In-Process Control in Thermal Rapid Prototyping

Nadya Peek
in MAS.864, May 2011

May 16, 2011

Abstract

This report compares thermal control algorithms for curing carbon fibre and epoxy resin composite parts using a networked mesh of embedded heater pads and temperature sensors. We compare 1. using proportional error correction, 2. PID control, and 3. an optimisation using the Nelder-Mead algorithm.

1. Initial Problem Setup

Suppose a mass M is to be heated up from an ambient temperature T_0 to a temperature T_1 with the temperature increasing linearly as a function of time from time $t = 0$ to time $t = t_1$, and and for time $t > t_1$ the temperature is to be kept at $T = T_1$. How much heat supply is required to achieve this? Let us denote the heat supply per unit time by q.

The heat flow q will consist of one part q_h required to heat the body, and another part q_r required to compensate for heat loss. I.e.

$$q = q_h + q_r \tag{1}$$

Rate of heating is $\frac{T_1 - T_0}{t_1}$, and to supply this one must have

$$q_h = c \cdot M \cdot \frac{T_1 - T_0}{t_1} \qquad \text{for } 0 < t < t_1 \tag{2}$$

$$= 0 \qquad \text{for } t > t_1 \tag{3}$$

Here c denotes the specific heat capacity of the material (this is assuming it is constant over the heated body; otherwise we should replace $c \cdot M$ by $\int c \cdot \delta M$, over body. In our case we will ignore the fact that we're working with a hybrid material.) We note that c may also be a function of the material's temperature, but we're going to assume it is constant.

The heat loss will depend on the temperature of the body, thus

$$q_r = q_r(T) \tag{4}$$

If heat loss is by conduction only, q_r will be a constant function,

$$q_r = K(T - T_0) \tag{5}$$

where K is a constant. Otherwise if radiation and/or convection are involved it becomes a nonlinear function, but in any case it is monotonically rising.

Thus in $q = q_h + q_r(T)$ for $0 < t < t_1$ the first term is constant, but the second is increasing with temperature. This means that the heat input must increase until the temperature T_1 is reached, and then it must drop suddenly by an amount $c \cdot M \cdot \frac{T_1 - T_0}{t_1}$ and remain constant, so that it continues to balance the heat loss without without further increase in temperature.

Now for the specific case of heating a part which includes uncured epoxy resin with an activation temperature T_a to begin the curing process, we will have to introduce a third q, which we will call q_e. Epoxy curing is an exothermic reaction, and we can expect the epoxy on the mould to start contributing to q through q_e once T_a is reached. In our case T_a has been set to be T_1, so we know q_e will be 0 when $t < t_1$.

Although using the specific heat capacity, the expected heat loss and the energy produced in the curing reaction we might be able to set out an energy input in advance, we are going to adhere to the temperature ramps required to cure the epoxy resin and carbon fibre part using closed loop temperature control.

Ideally, we will see a temperature controller following a ramp that we can imagine from the theory described above: monotonically increasing while the temperature is ramping, and then suddenly dropping when it is only required to hold temperature.

2. Closed loop temperature control

To test a heated mould with calorimetry for cure sensing, we built a system that controls a network of microcontroller nodes, each equipped with a temperature sensor (glass thermistor) and a heating element (NiCr wires controlled with PWM). The nodes can be set to different temperature profiles that can correspond to the thickness of the part at that location in the mould. The nodes communicate through Asynchronous Packet Automata, a communication protocol developed at the MIT Center for Bits and Atoms in collaboration with Spirit Aerosystems, where the geometry of the nodes connections determines how to address them. The nodes are controlled with a series of simple commands, including current up, current down and read temperature. The entire network can be addressed from a master computer running a controller algorithm. A graphic depiction can be seen in figure 1.

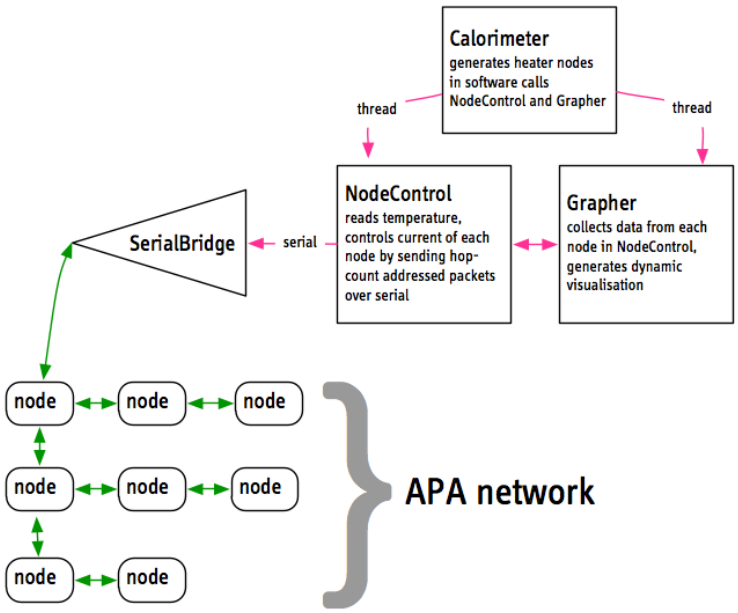

Figure 1: Overview of the system setup with APA nodes and master control from an external computer.

2.1. Proportional control

In the first experiments, the algorithm implemented was simply to change the input energy at a rate proportional to the deviation in temperature. The first experiment done using this control method can be seen in figures 2, 3 and 4.

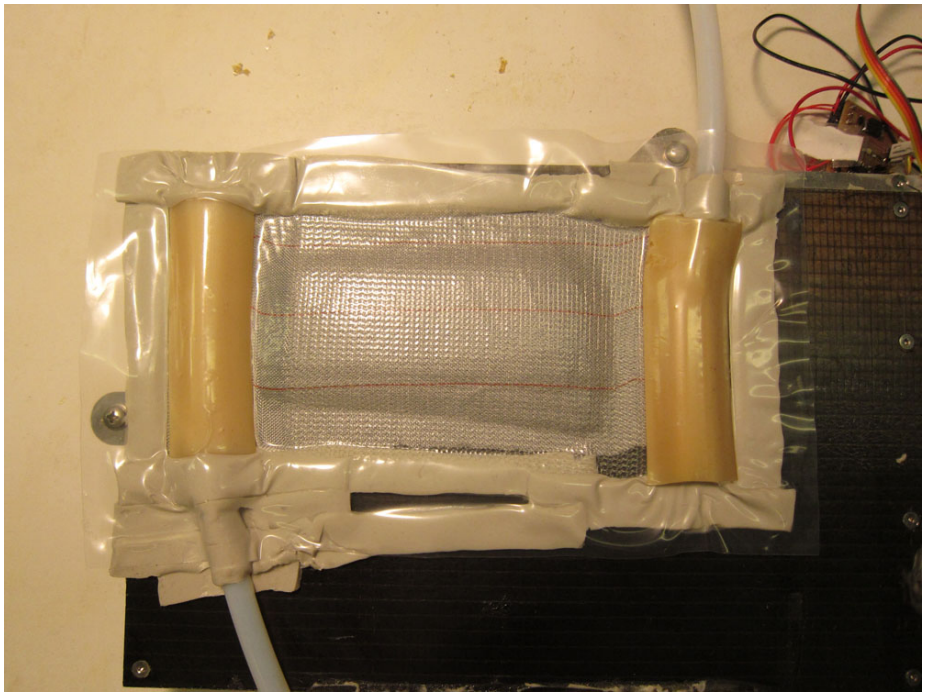

Figure 2: Setup for infusing carbon fibre with epoxy resin on a mould with temperature sensors and embedded heating pads. The epoxy enters the mould already heated to 80 C. On top of this we have positioned a heating lamp to maintain ambient temperature around the mould at around 80 C. Node 1 in figures and is on the left, and node 2 is on the right.

The mould that we are using in this experiment was developed at Spirit Aerosystems using their composite production facilities. It is made with the same materials that it is meant to cure, which should minimise the difference in specific heat capacity between the mould and the part, so that our q_h will remain predictable.

Unfortunately, there is some variation in the temperature sensors we embedded in the mould, and we have found they do not report consistent temperatures across sensors. To remedy this, we calibrated each node in software with reference temperature readings made with an infrared camera on the surface of the mould.

Ambient temperature in the room that we used for the experiments is set at 21

C. The mould is heated to 80 C before the introduction of the epoxy, which is also already heated to 80 C (without the heat, the epoxy has too high a viscosity to flow through the part).

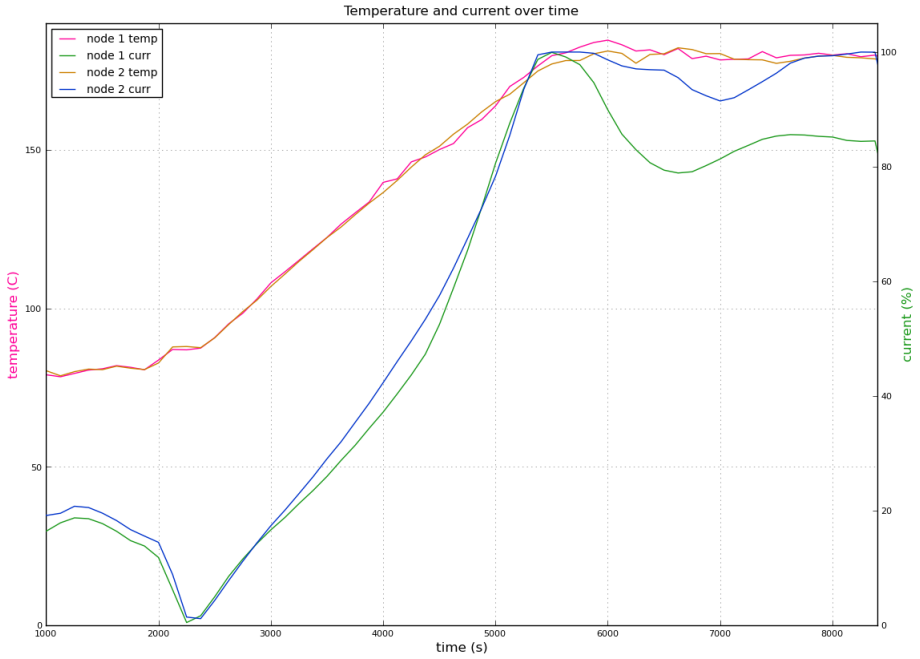

Figure 3: Curing Cytec epoxy with a ramp from of 2 C per minute from 80 C to 180 C. This figure shows two separate nodes embedded in a single carbon fibre mould. The two separate input current and temperature lines are for each heating pad and corresponding thermistor. The maximum current is 0.5 A per node. The decreasing input energy through $t = 2200s$ is due to the repositioning of the external heat lamp.

In figure 3 you can see a drop in input current from $t = 5700s$ which is rectified at $t = 7500s$. Just looking at this graph, we might think that what we are observing is simply our control algorithm being slow to respond, not immediately dropping to only supply q_r and then overshooting when it does. To better understand the dynamics of this plot, we ran a control experiment which can be seen in figure 4. In the control we re-ramped the part we had previously already cured, after manual inspection that it had indeed cured correctly.

Comparing the two plots, we notice that it is not a resonance. We could subtract the first plot from from the second to show only the different in $q_h + q_r$, which we will assume to be q_e.

This is already quite promising a result, but to make more certain what we're seeing has nothing to do with our current input, we would like to implement a

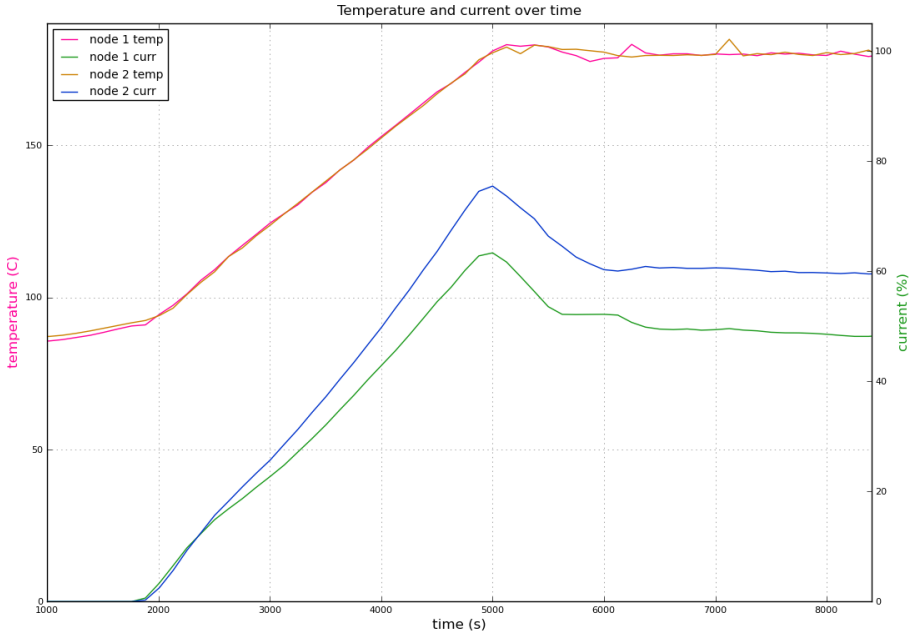

Figure 4: This is the same setup as shown in figure 3, except now the mould and part have already gone through one heating cycle and the epoxy has hardened.

better control algorithm which will more closely approximate the energy input we are theoretically expecting to see, which includes a discontinuity at T_1 instead of the decline proportional control must give us.

2.2. PID control

PID control has been widely used for nearly a half century of control. It is a control loop mechanism that takes the proportional error, the rate of change in error and the accumulated error into account when calculating how to adjust the output at any given step.

If we let $e(t)$ be the error at time t, then the output current at time t can be given by:

$$out(t) = K_p \cdot e(t) + K_i \cdot \int_0^t e(t)dt + K_d \cdot e(t)\frac{d}{dt}$$

In the sum, the first term is the proportional, the second the integral, the third the derivative. Here the gains K_p, K_i and K_d are tuning parameters. We will elaborate on selecting values for these parameters in 2.2.1.

In algorithm form, the PID controller can be implemented as follows:

PID-control(*target*, *current*, *prev_error*)

1 *error* ← *target* – *current*
2 *integral* ← *integral* + *error* · *elapsed*
3 *derivative* ← (*error* - *prev_error*) / *elapsed*
4 *output* = K_P·*error* + K_I·*integral* +K_D·*derivative*
5 *prev_error* ← *error*

2.2.1. Tuning

Manually selecting good values for the K_p, K_i and K_d gains is somewhat tricky, and entire bookshelves have been written about it [1]. When manually tuning a PID loop, first the K_i and K_d gains are set to 0. The K_p is increased until the output oscillates. Then K_p is decreased from that value, and its decrease is redistributed over the integral and derivative terms. Determining exactly how to redistribute can be guided by many different rules of thumb, such as *increasing K_p will increase overshoot but decrease steady-state error*.

We've chosen a heuristic tuning method known as the *Ziegler-Nichols Method*, developed by Ziegler and Nichols in the 1940s by observing helmsmen steering ships. Like with manual tuning, the K_i and K_d gains are initially set to 0. K_p is increased until the output starts to oscillate with constant amplitude, at value K_u. The period of the oscillation is T_u. Then the parameters are selected from the table 1:

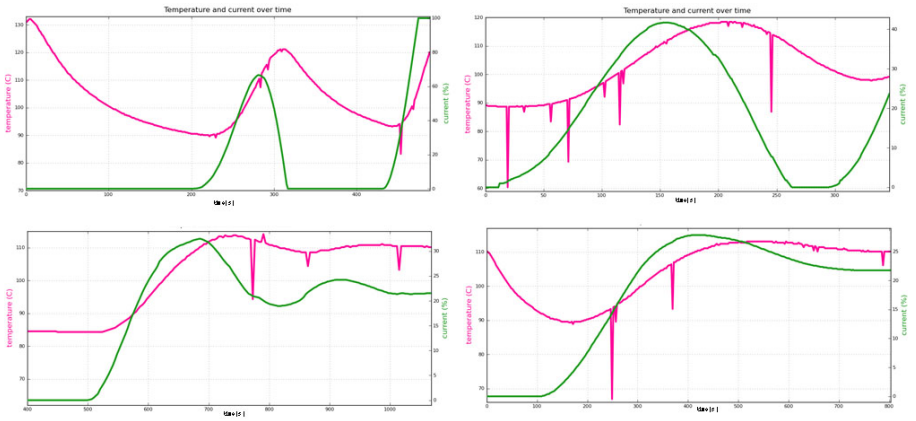

Figure 5: Temperature (pink) and input current (green) with four consecutive tunings of PID control. The node was given a temperature ramp of 5 degrees per minute, from 90 C to 110 C. In a, K_p is too large and the output is oscillating, and K_i is too large so the changes are delayed. In b, this is somewhat reduced, but the control is still largely overshooting. Plots c and d show better results but fail to produce the angular temperature we expect.

Control Type	K_p	K_i	K_d
P	$K_u/2$	-	-
PI	$K_u/2.2$	$T_u/1.2$	-
PID	$K_u/1.7$	$T_u/2$	$T_u/8$

Table 1: Ziegler-Nichols PID tuning heuristic

This method is tricker than it may seem– even with the Ziegler-Nichols method, it is difficult to get values for the gains that are accurate for a variety of temperature ramps. Often, a timely response in one part of the temperature ramp is paired with an oscillation in another.

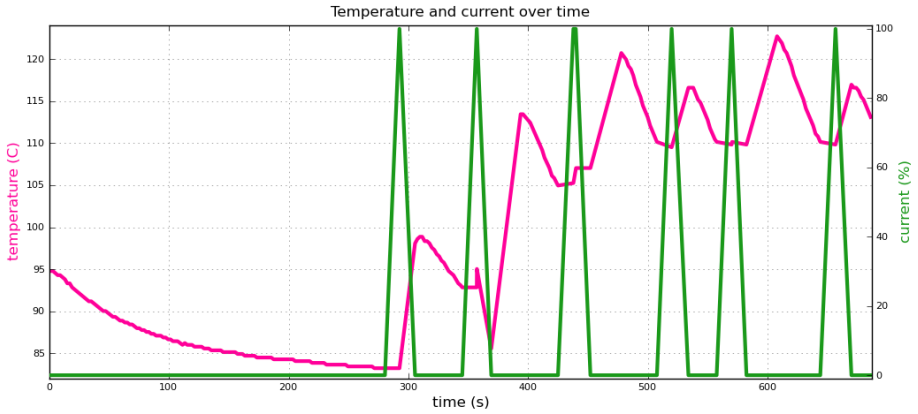

Figure 6: With too aggressive tuning, the output from PID overshoots so much and also takes some time to update all the nodes in the network that horrible awful oscillation occurs.

2.3. Nelder-Mead Algorithm

This algorithm, first proposed by Spendley et al [4] and later developed by Nelder and Mead [2], uses a heuristic search strategy to find a minimum of a function of n variables. It depends on the evaluation of the function at the vertices of a *simplex*, where a simplex is defined to be a polytope with one vertex more than the space it is searching in. The method described below follows the implementation of amoeba in *Numerical Recipes for C* [3].

The function is evaluated at each vertex of the simplex. The values are sorted from smallest to largest

$$f(x_1) \geq f(x_2) \geq ... \geq f(x_{n+1}) \tag{6}$$

The worst point is omitted when calculating the centroid of the polytope (the mean of the columns), and then the worst point is reflected across the face towards the best point:

$$x_r = x_0 + \alpha(x_0 - x_{n-1}) \tag{7}$$

Here α is a reflection coefficient (we use 1). If $f(x_1) \geq f(x_r) < f(x_n)$, then x_r becomes the new x_{n+1} in the simplex, and the algorithm loops.

If the reflected point is the best found, or $f(x_r) < f(x_1)$, then grow the reflected point out to see if it improves even more:

$$x_r = x_0 + \gamma(x_0 - x_{n+1}) \tag{8}$$

Here γ is the growth coefficient (we use 2). If $f(x_e) < f(x_r)$, then a new simplex is formed where x_e is the new x_{n+1}, else x_r is the new x_{n+1}.

If the reflected point is the worst found, or $f(x_r) \geq f(x_n \forall n)$, then shrink x_r by:

$$x_c = x_{n+1} + \beta(x_0 - x_{n+1}) \tag{9}$$

Here β is the contraction coefficient (we use 0.5). If the contracted point is better than the worst point or $f(x_c) < f(x_{n+1})$, then loop with a new simplex with x_c.

If none of the above steps helped, shrink the entire simplex towards the best result, or for all i in $\{2, ..., n+1\}$:

$$x_i = x_1 + \rho(x_i - x_1) \tag{10}$$

Where ρ is a shrinking coefficient (we use 0.5). To make sure our simplex does not get stuck in a local minima, we have to ensure that the initial simplex is of sufficient size.

To apply Nelder-Mead to our specific problem, we need to determine the function we are trying to minimise. Our aim is to optimise our controller's parameters to best approximate the temperature ramp we have specified. To determine our search space, we performed $k+1$ experiments where we varied our parameters. The experiment with the worst result is discarded, and a new one is substituted according to the Nelder-Mead method.

The coordinates we select for our starting simplex can be defined by the following matrix, where the links between coordinates are given by

$$x_{ij} = x_{1j} + X_{ij} \cdot \delta x_j \tag{11}$$

here i ranges between 1 and $k+1$ and j ranges between 1 and k.

$$\begin{Bmatrix} 0 & 0 & 0 & 0 & 0 \\ p & q & q & q & q \\ q & p & q & q & q \\ q & q & p & q & q \\ q & q & q & p & q \\ q & q & q & q & p \end{Bmatrix}$$

where

$$p = \frac{1}{k\sqrt{2}} \cdot (\sqrt{k+1} + k - 1) \tag{12}$$

and

$$q = \frac{1}{k\sqrt{2}} \cdot (\sqrt{k+1} - 1) \tag{13}$$

We use the response function to a desired increase of 10 C as an optimisation criterion (noting that this may not be so optimal, but doing experiments with a more realistic desired increase of 100 C would take weeks unless someone wants to buy me a fancy Peltier block). It is a weighted sum that considers both the error deviation and the duration and amplitude of the oscillations:

$$F_r = \sum S_i + a \cdot t_{osc} + b \cdot E_i \tag{14}$$

Where S_i denotes the surface of oscillations around the set point, t_osc is the duration of oscillations in seconds, and E_i the error in degrees. The weights a and b were both set to 1.

The variables we have determined to effect the outcome are the integral time (in seconds), the derived time (in seconds), the tuning parameters K_p, K_i and K_d and of course the input energy in %. Therefore our simplex has 6 variables and will be represented by a polytope in 6-dimensional space with 7 vertices.

The starting coordinates use the k that has parameters set by previous Ziegler-Nichols tuning attempts.

3. Issues

Somehow the incantations that I normally use to get my APA network online hate me. After much debugging and re-making of controller boards, I have determined that 1: one of the heater pads in the 2x2 is burnt out, 2: related to 1, I want new heater nodes and that 3: we need to do something about this weird state the boards can come up in (...Neil?).

References

[1] K. J. Äström. *PID Controllers: Theory, Design, and Tuning*. International Society for Measurement and Control, 1995.

[2] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[3] W. Press and S. Teukolsky. *Numerical Recipes in C*. Cambridge University Press, 1992.

[4] W. Spendley, G. R. Hext, and F. R. Himsworth. Sequential application of simplex designs in optimization and evolutionary operation. *Technometrics*, 4:441–443, 1962.