Software Systems Final Project Deliverable: Case Study

Nicole Rifkin, Rahil Dedhia

May 7, 2014

Problem

The behavior of a thermostat is a classic control theory problem. We are going to explore a similar model, where we want to control the temperature of a room. The twist is that we want the temperature to be variable between 0 and 100°F. As a result, we are using a controller that sends a voltage to a plant with both heating and cooling actuators. The controller decides whether we want to use the heater or the cooler, and how much voltage we want to sent to that actuator in order to change the air temperature. Initially, we will build a simpler model of the system by abstracting out delays. Then, we will try to see how the system behaves when we take the delays of the actuators into account, and adjust our model accordingly.

Defining the Model

Brief Summary of P, PI, and PID Control

The three types of controller gains we use are proportional, integral, and derivative. Proportional control is generally used in all systems, but by itself is not enough. Once the error gets too small, the proportional gain is no longer helpful, and increasing the gain often causes overshoot. Integral control is added for this purpose, because it can eliminate steady-errors by continuously adding up the error. The problems with integral control is that by smoothing the signal, it responds more slowly, and also that it tends to cause oscillations. The slow response can cause major problems with setpoint changes. Derivative control can make systems respond faster to changes, but due to this it strongly enhances noise, which is why a lot of controllers choose to use PI control instead of PID control.

Describing the System

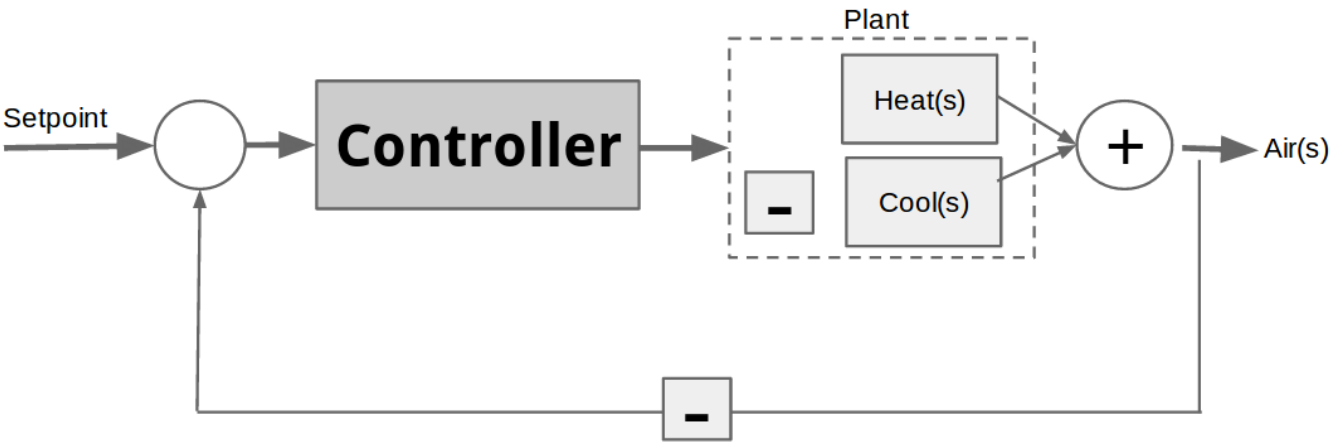

Figure 1: Control Diagram of our System

Figure 1 shows a diagram of our system assuming no delays. The controller sends a variable voltage between -1 and 1 Volt to the plant depending on the error and the setpoint. If the voltage is positive, it is send to the heater. If it

is negative, it is inverted, and then sent to the cooler. For the controller, we use a PI controller. For the actuation of the system in the plant, we use Newton's law of cooling.

Simulation

Figure 2 shows the output response of our system with proportional gain at .9 and integral gain at .5. The system takes more iterations to respond to large changes in the set-point than smaller changes. The system oscillates around the set-point in steady state. Figure 3 displays the output under the same conditions as above. However, the setpoint stays constant, and only a small number of iterations are shown. From the graph, it is evident that the system will reach steady state after about 15 iterations.

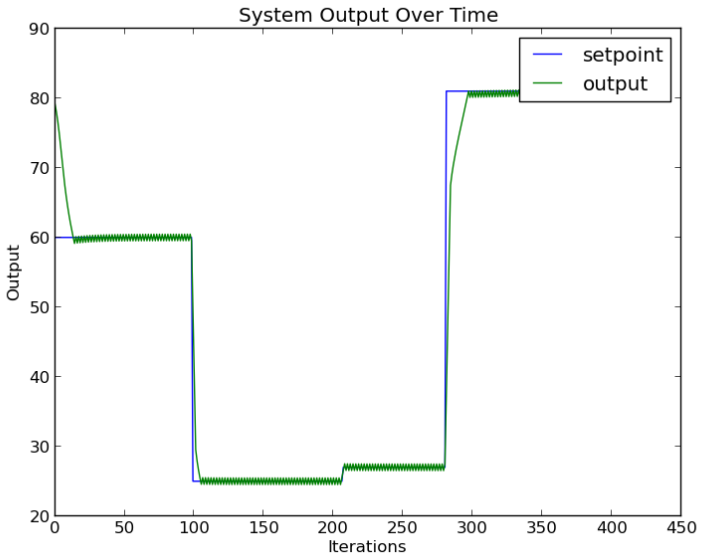

Figure 2: Shows the output of our system as a function of the number of iterations through the control loop.

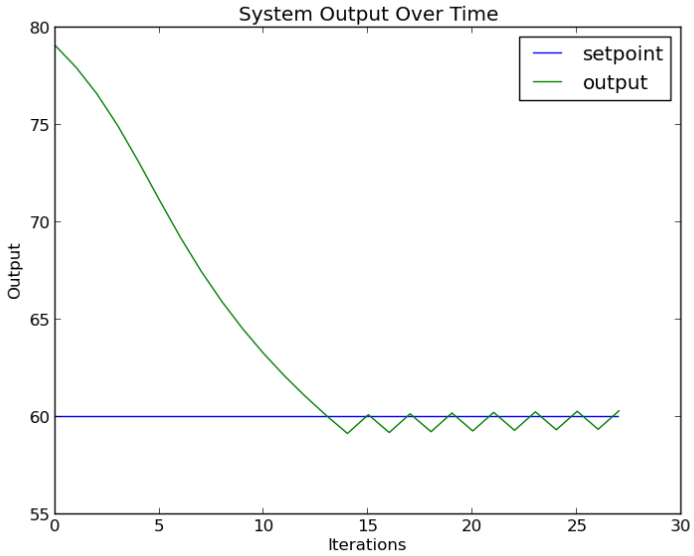

Figure 3: Shows the output of our system for a small number of iterations.

Figure 4 shows the error in the system under the same conditions. The instantaneous error experiences spikes when the set-point changes but reaches zero in steady state. The cumulative error reaches a constant value in steady state.

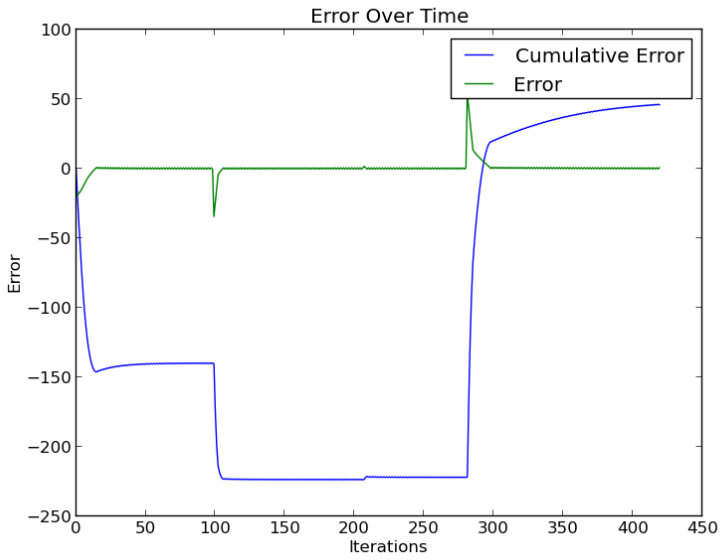

Figure 4: Shows the error in our system as a function of number of iterations.

Figure 5 exhibits the system with no integral control. The system is still oscillating, which shows that oscillations are inherent to the system and not a result of control action. It is likely that this is due to the nature of a heating/cooling plant, which constantly switches from one source to another. In steady state, the system's lowest point is still just above the set-point. This is due to proportional droop, and is fixed when integral control is introduced.

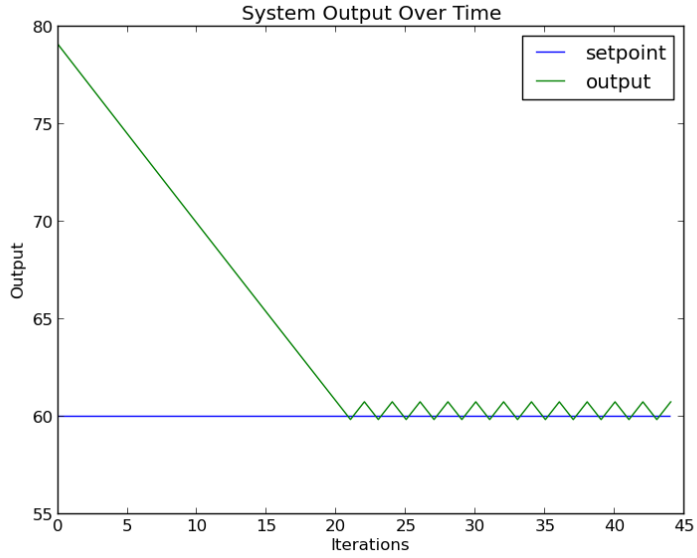

Figure 5: Shows the output of the system without integral control.

Figure 6 shows the behavior of a system with no control. The temperature of the system drops linearly. This

behavior is expected initially, but should drop off at a certain temperature. This is a limitation of our model.

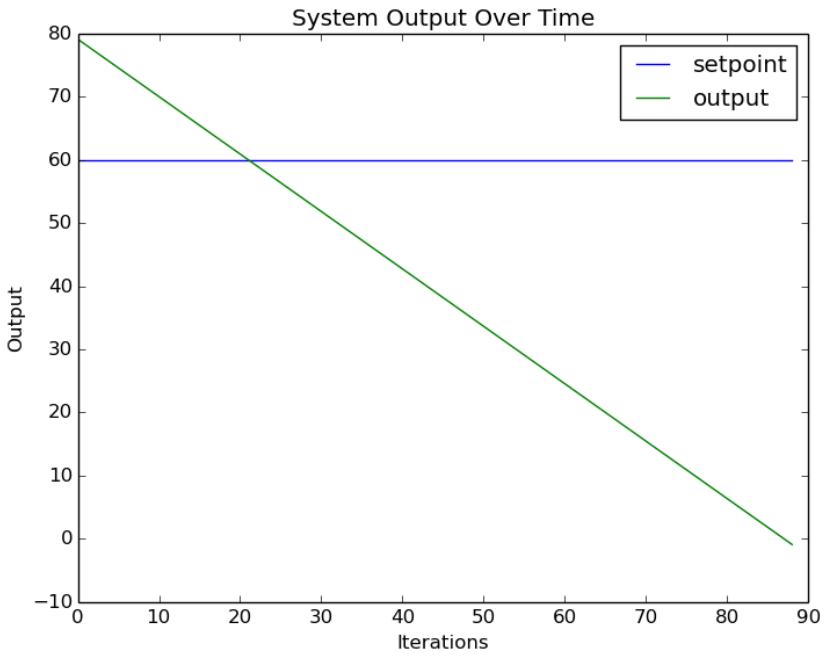

Figure 6: Shows the output of the system without a controller.

Improving the Model

Integrator Windup

If the heating or cooling system needs to make a very large change, it is possible that the system will not be able to produce as much as it needs to. This causes actuator saturation, where the control signal and corrective action no longer correspond directly. As a result, due to the limitations of the actuator, tracking errors will not be corrected, so the integrator will continue to add them up until it catches up and overshoots. This leads to a cycle where the integrator has to "unwind" after the actuating "catches up" to the input. A solution to this is integrator clamping, where we stop adding to the integral term during actuator saturation. While this phenomenon is common in many systems, in our simulation we did not observe any integrator wind-up, even when the integrator term was cranked up to a gain of .9. We believe that this is because the system can quickly oscillate since it provides both a heating and cooling mechanism. This might explain why any wind-up was quickly corrected and was not apparent on our graphs.

Delays

Time delays serve as a major practical inconvenience, because they make transfer functions irrational. Shifting a function by "T" in the time domain introduces an e^{-sT} factor in the frequency domain. This shift in a control problem can describe a delay. They pose a major problem to a controller just like actuator saturation does. When the controller tries to make a corrective action based on the output of a plant with a delay, it becomes very easy to overshoot the setpoint value. Because the output is delayed, the controller can overcompensate by providing a greater control signal than it should. This can lead to a great deal of oscillations, and sometimes even causes a system to diverge permanently.

Smith Predictors

A Smith predictor is a control strategy that is useful for handling delays. This strategy can be used if we know the length of the delay T, and have a model for the system assuming no delay. Displayed in Figure 7, a Smith predictor

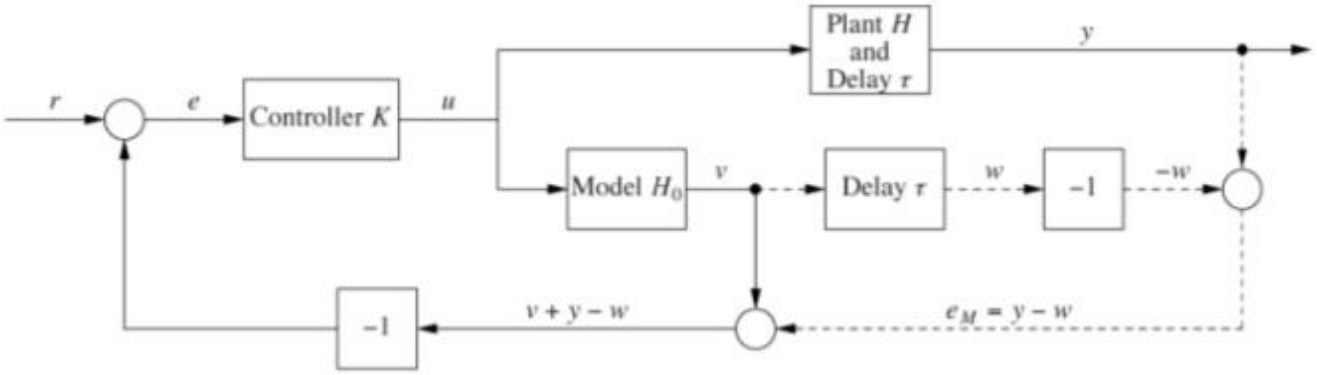

Figure 7: Diagram Explaining the Behavior of a Smith Predictor

has a normal feedback path from the plant, but with two more terms factored in. It also takes into account a predictive model assuming no delays. Finally, this model is put through the delay T, and that is subtracted from the output of the predictive model. The delay factors in as a complex exponential e^{-sT}. A good model for a Smith predictor allows the controller to respond to setpoint change much quicker than with a normal PI controller. Figure 8 shows a control diagram of using Smith predictor for our air temperature model. 8.

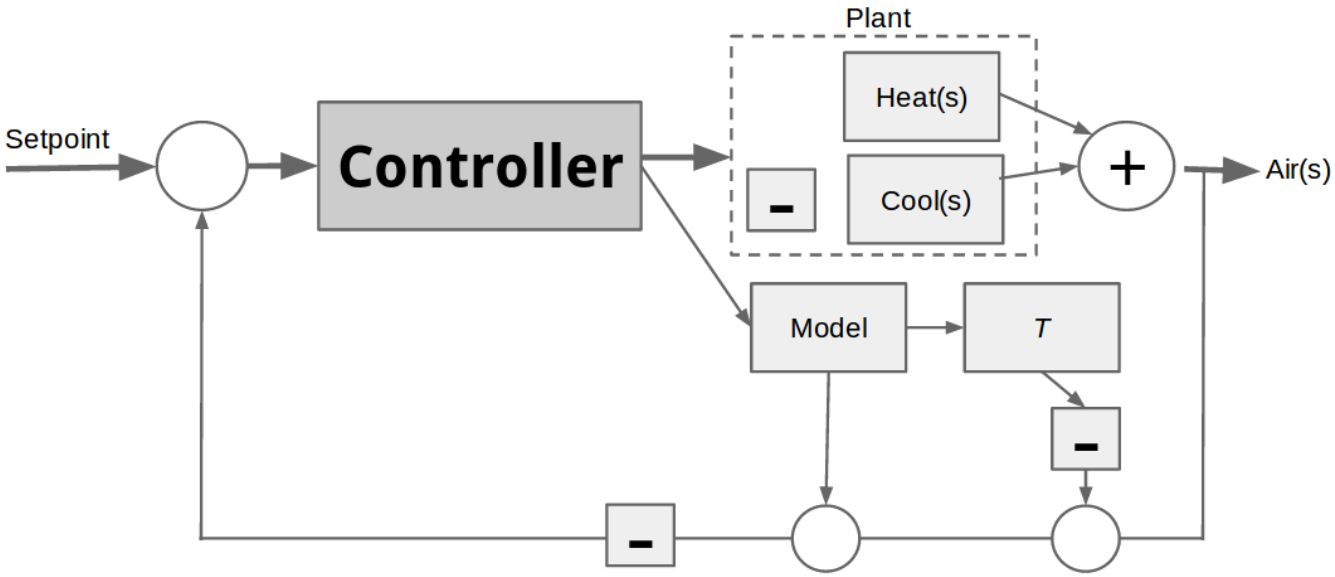

Figure 8: Diagram of our System with a Smith Predictor

Future Work

In the future, we would like to implement the smith predictor that we specified for our system in the model above. In addition, there were some parameter tuning methods specified in the book like Ziegler-Nichols tuning that we could explore to better optimize our system.

More on Feedback Control

In our project proposal, we also mentioned that we want to understand more about control theory, and its applications to computer systems. In the section below, we describe some more topics we learned about while

reading Janert.

Discrete Sampling

Computer systems are generally discrete, so they rely on a sampling interval to respond to change. Faster intervals are better, because it is better to make many small control actions than a few large ones. Small control actions tend to be a lot safer than large ones, because there is a lower chance of overshoot and oscillation as a result. That said, sampling too quickly can lead to round-off errors with derivative control. Ideally, the sampling interval should be at least 5-10 times faster than the fastest process we want to control.

In addition, a PID controller produces floating-point output, which can be a problem in a discrete system. For example, 6.4 server instances might be required, where 6 is not enough but 7 won't be fully utilized. A solution is the give the correct output "on average" by letting the controller switch between 6 for 60% of the time and 7 for 40% of the time.

Instead of a continuous time model, discrete systems can have a *discrete-time model*, which uses difference equations instead of differential equations and *z-transforms* instead of Laplace transforms. The theory is similar to that of a continuous time model, but some of the implementation details are different.

PID Customizations

Sometimes, using nonlinear PID controllers are beneficial. For example, if we control the fill level of a buffer, but don't care abbout fluctuations unless they become too large, we can use an *error-square* controller, which multiplies the original controller response by the absolute value of the error. Another solution is a *gap controller*, which treats errors as 0 unless they exceed a certain threshold (gap).

Feedback loops can also have a few sets of gain parameters instead of one, to make them more customizable. This is mainly used for nonlinear systems, where the response can be qualitatively different for different gain values. By having multiple sets of values for controller gains, nonlinear systems can achieve better performance across all inputs. Another customization is nested control loops, which are useful when there are two processes with very different timescales.

Poles and Zeros

Zeroes are important because they block signal transmission. Poles are important because the transfer function in the time domain (the inverse laplace transform) is a linear combinations of exponentials for each pole. Real poles behave exponentially, and imaginary components add oscillation. Purely imaginary poles are labeled as *marginally stable*, and as the imaginary coefficient increases, poles oscillate at higher frequencies.

Sometimes, analysis of higher order systems can be simplified by only focusing on the *dominant poles*, which are essentially the slowest poles. Poles that are very far to the left on the s plane decay very quickly, and poles very close to a zero have a very low amplitude, so usually these poles are not dominant and can be ignored.

By understanding a system's poles, we can use *pole placement* to design a system with ideal behavior by "moving its poles". Controller tuning is a way to do this by using controller gains. In addition to changing control gains, additional elements can be inserted into the control loop as transfer functions by *loop shaping*. For example, the transfer function $G(s) = \frac{1+as}{1+bs}$ is a lead/lag compensator. The function has a pole at $-1/b$ and a zero at $-1/a$. The zero can be used to coincides with an unfavorable pole to remove its influence from the system. The value of b can be adjusted to place the compensator pole into a favorable position.

Other Control Methods

State-space methods linear algebra to do fun vector to allow total controller synthesis instead of just tuning. A flaw in these methods is that they are not robust, because they are very precise for a specific problem, and in reality, there is a certain degree of "model uncertainty".

Writer's Notes

We were hoping to do a control study on a computer system. However, Janert already covered several examples in case studies including cache hit rate and server networks. We chose a heating problem because we knew that there was already a transfer function that we could analyze and test with. This is not important for implementing control on real systems, but was necessary for our simulation. We also chose this problem because it is applicable to computer fan cooling systems, and it includes a delay which is found in a number of controlled software systems. Our overall understanding of control theory after this project can now be applied to many physical or software systems.

Our repository can be found at `www.github.com/rdedhia/SoftwareSystems/hw06`

References

1. Janert's *Feedback Control for Computer Systems*
2. `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=348347`
3. `http://www.mathworks.com/help/control/examples/control-of-processes-with-long-dead-time-the-smith-predictor.html`
4. `http://137.132.165.75/matlab/Smit/theory3.html`