Week 1 Slides

Instructure: Prof Tim Gordon (tgordon@lincoln.ac.uk)

50% Coursework, 50% TCA. Coursework will be introduced in week 3

Main Topics of the Module

- **Introduction to Control**
- **Introduction to Block Diagrams and Simulink**
- Mathematical methods – Laplace transforms and Transfer functions
- Transient and steady-state responses
- Stability criteria and PID controllers
- Root locus techniques
- Frequency response and Bode plots

What is Control?

We can think of many everyday examples of a person (or thing or group or nation) controlling something.

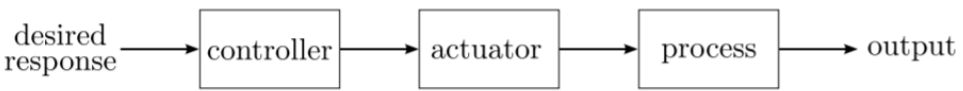
- Governments want to control inflation
- Health professionals want to control disease
- Pilots want to control cruising altitude
- Drivers want to control speed and direction
- …..

If you can control something "with your eyes closed" it's probably an open-loop control system. If you need to "watch" it carefully it's probably a feedback control system.

Open-Loop Systems

Simply

for example in indstrial automation where a series of electronic switches trigger actions such as valve opening or for heating circuits. What other processes are like this? A more detailed picture:



The desired response might be “valve open”. The controller (electronic circuit performing logic operations) then triggers a valve motor. Control is open-loop because it **does not monitor the response** (is the valve open?).

Advantages of open-loop control

- Inexpensive (saves on sensor costs)
- Simple
- Does not have the stability concerns of closed-loop systems

Open-loop systems need to work **predictably** and **reliably** - the valve must actually work (almost) always.

In control jargon the system being controlled is called the **plant** and the plant. If the plant is highly predictable then open-loop control may be an option

An open-loop control system does not make corrections, so **disturbances** can be a problem.

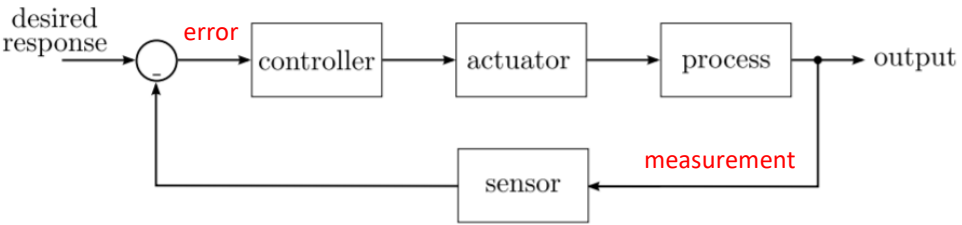
- Open-loop driving on a straight road: point the car in the right direction and then hold the steering wheel straight!
- Would this approach work for speed control?

Real-world examples of open-loop control are commonplace.

Microwaves and toasters work on timers (probably reliable) – they normally don't check whether the food is still frozen or the toast is golden brown.

As sensors (and logic circuits) become cheaper, closed-loop control of everyday devices becomes more realistic commercially.

Closed-Loop Control



- A “closed-loop toaster” would need a sensor that detects bread colour and would need an algorithm in the logic circuit to deal with different bread types (white, brown, bagel, crumpet, …)
- In simple terms, the desired response in a number (temperature, reflectivity, position, velocity, ….) and the process is operated until this desired result is achieved.
- Then repeat!
- There are many everyday examples of closed-loop control.
- Note the little negative sign in the feedback loop – it’s important!

Terminology

- Feedback control = closed-loop control (= error correction)
- The input to the controller is the **error signal** $e(t)$
- The output of the controller is the **control signal** $u(t)$
- The desired response (desired temperature, position etc.) is the reference, or **reference input** $r(t)$
- The actual response, $y(t)$ is controlled to minimise $e(t)$, that is we want $y(t) \approx r(t)$
- When the reference input is constant the control system is a **regulator** (system).
  Examples: regulating an oven temperature, regulating vehicle speed by cruise control,
- When the reference input changes over time, $y(t)$ is expected to **track** the changes in $r(t)$ – so it's called **tracking control** (or **servo control**)
  Examples: steering control in a self-driving car (track the changing curvature using a steering motor), steering an research submersible to follow a pod of whales, …

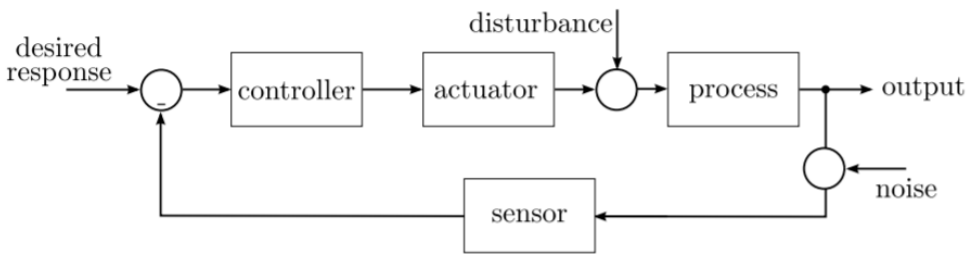
Feedback control has several advantages over open-loop control:

- **requires less knowledge of the plant** e.g., we just need to know that increasing the motor speed on a heat pump will increase the temperature of the house. It helps to know an approximate model for designing the controller, but this does not need to be accurate.
- the effects of **disturbances** $d(t)$ can be minimized. For example a cruise control signal automatically compensates for hills and headwinds. $d(t)$ affects the plant directly and is usually not measured by the control system.
- feedback is capable of much **greater accuracy** than manual of open-loop control. For example feedback is used when controlling a robot arm; sensors (encoders) are used to determine the precise position of the arm. Of course the sensor needs to accurate and the control system needs to work well.

On the other hand

- feedback often works best when the controller has “high gain” but this can lead to **unstable behaviour**.
- **time delays** in the feedback loop can be a particular problem for stability
- **costs** of sensors and signal transmission need to be considered

The system block diagram here includes a **disturbance input** and **sensor noise**.



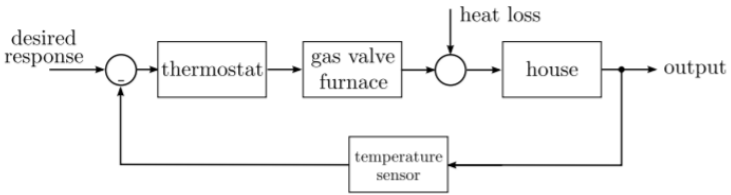
We want our closed loop control system to **insensitive to disturbances**. Later we will use the concept of a **transfer functions** to evaluate control system properties, including this senstivity.

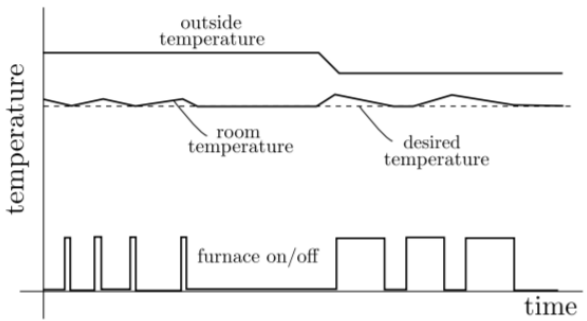
Sensor noise might be due to electrical interference, vibrations of machinery and is the worst enemy of a control engineer!

For obvious reasons, “feeding back noise into a high gain controller” produces bad results, and sensor quality (and reliability) is critical in closed-loop control.

It’s like reacting strongly to meaningless information in real-life – never a good idea!

Example – temperature control of a house (regulator problem)





We should be able to identify the following: $y(t), e(t), d(t), r(t)$.
And we should be able to identify the plant, the sensor, the controller and the actuator. Why is the negative sign important?!

At this point you should know the very basics of what a control system looks like and several of the important idea. Of course we need various tools to be able to **predict the performance** of a control system and **design** controllers in a systematic way.

Toolbox

- Mathematical models
- Block diagrams
- Matlab and Simulink
- Laplace transforms and transfer functions
- Stability analysis (s-plane poles)
- Design methods – especially root locus
- Frequency response – especially Bode plots
- …. and textbooks

Some reference text books (dates and editions are not important)

- Nise, Norman S., 2008. *Control systems engineering*. 5th ed. Hoboken, N J: John Wiley & Sons.
- Ogata, Katsuhiki, 2009. *Modern Control Engineering*. 5th ed. Prentice-Hall.
- Richard C. Dorf, Robert H. Bishop, 2011. Modern Control Systems. 12th ed. Pearson Education, Inc.
- Gene F. Franklin, J. David Powell, Abbas Emami-Naeini, 2010. Feedback Control of Dynamic Systems. 6th ed. Pearson.
- Katsuhiko Ogata, 2008. MATLAB for Control Engineers. Pearson.

Never try to read a control textbook from cover to cover – life is too short! But do dip into them to improve understanding and check examples.

Tutorial Question

1. In the similar way that the example considered temperature control of a house, consider cruise control of a car. The driver sets the target speed and the controller does the rest!

    a. draw a block diagram similar to the temperature control example
    b. identify the plant, the sensor, the controller and the actuator (you may need to check some facts online)
    c. identify possible sources of disturbance
    d. identify $y(t), e(t), d(t), r(t)$
    e. sensor error can be dangerous – can you explain why?

Block Diagrams

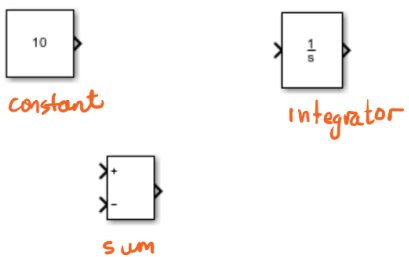
We have used these already in a general sense to represent the sub-systems of a control system. They show connections as signals and blocks to represent sub-systems.

They are also a useful tool for modelling and simulation (also design and analysis – we come to these later).

Here are some basic blocks (lego bricks of control).

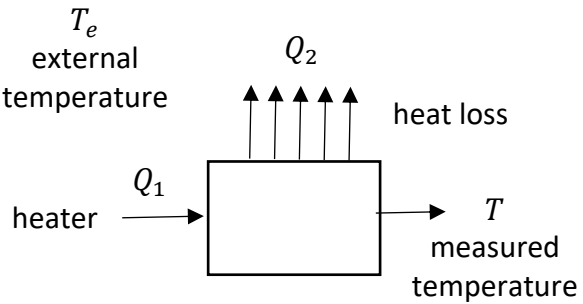
The integrator block performs integration of the input to find the output (area under the curve).

Think of s as differentiation. Then $1/s$ is the inverse = integration.



Example – Temperature Control of an Object

Here is the general situation and the resulting model equations.



$Q_1 = u$ is the rate of heat input (control input) and $T = y$ is the measured temperature output.

Heat loss is proportional to temperature difference

$$Q_2 = kA(T - T_e) \qquad (1)$$

Change in temperature is proportional to the net heat input

$$mc\,\Delta T = (Q_1 - Q_2)\Delta t$$

where Δt is a short time interval and ΔT is the corresponding temperature change. Alternatively

$$\dot{T} = \frac{Q_1 - Q_2}{mc} \quad (2)$$

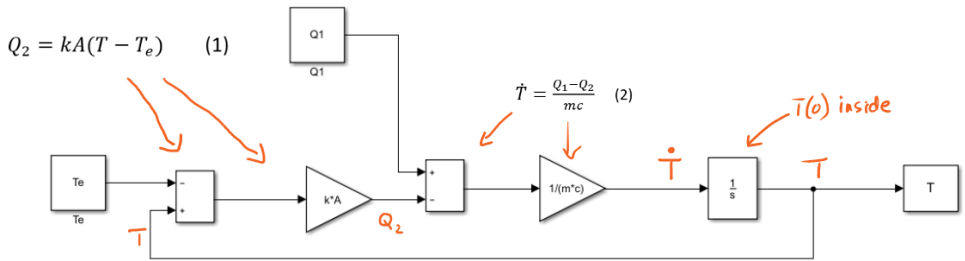
Note: m=mass, c is specific heat capacity, k is the heat transfer coefficient, and A is surface area.

Equations (1) and (2) are enough to determine the 'time evolution' of the (open-loop) system using Simulink.

We will assume the heater is turned on and left on, so Q1 = constant.

The figure below shows how the equations relate to a block diagram.

The temperature is output and, after parameters are set (in Matlab) the temperature profile can be found by running the model.

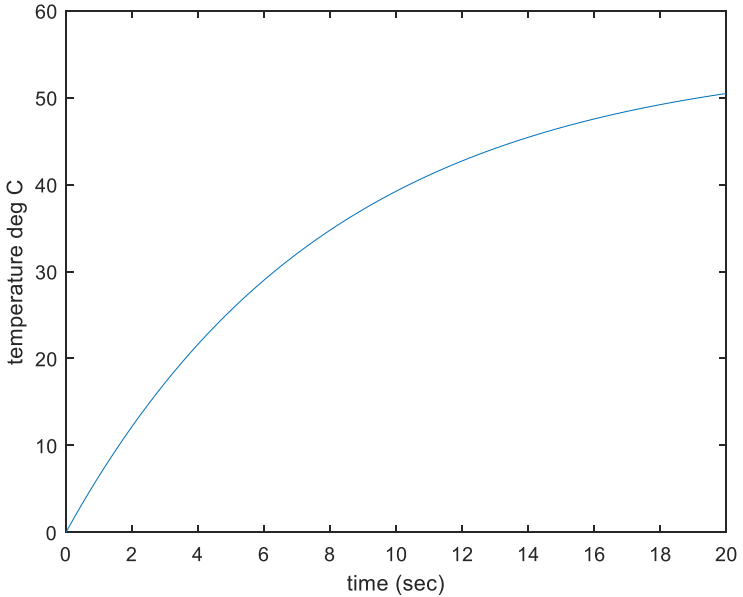


You will learn to create this yourself shortly! We create two files, one is called Temperature_01.slx and contains the Simulink diagram above. The other one is a Matlab script file, CallTemperature_01 which sets parameters, tells the simulation to run and plots the temperature result. It's simple but incredibly powerful!

Matlab script file

```
clear
close all
%set parameter values
model='Temperature_01';
m=1; %kg
A=0.25; %m^2
k=0.5; %heat transfer coeff
c=1; %specific heat capacity
Q1=5; %constant heat input
Te=15; %deg C environment temperature
T0=0; %initial temperature (see integrator block)
%run the simulation and plot the result
sim(model,20)
figure
plot(tout,T)
xlabel('time (sec)')
ylabel('temperature deg C')
```

The result is as follows – it gets hotter quickly at first but heads towards a constant temperature that is somewhere between 50 and 60 deg C.



Notes

- the variable **tout** is set internally in Simulink for convenience. You will need to know a bit about some of the internal settings of Simulink (exercise in class).
- The simulation time is set in the **sim** command.
- Simulink has it's own 'run' button, but here (since parameters need to be set) it's convenient to let Matlab take control.

We can also understand the system by rearranging Equations (1) (2) to eliminate Q_2. Some simple algebra yields

$$\dot{T} + aT = aT_e + bQ_1$$

where $a = \frac{kA}{mc}$ and $b = \frac{1}{mc}$. This is a first-order linear differential equation that can be solved in terms of the given parameters, noting that the right-hand side is constant. In the tutorial section you can check that the solution of this matches the Matlab plot, but the Simulink diagram is the most important tool for us here.

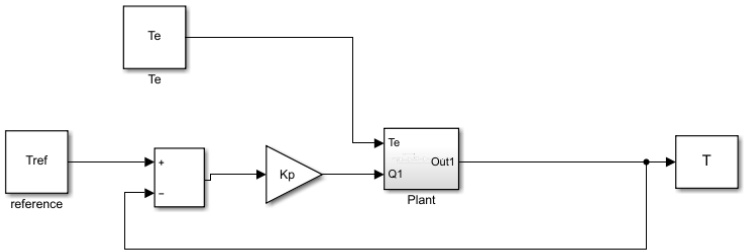
More simply, the steady-state temperature comes from setting $\dot{T} = 0$. Then

$$T_{ss} = T_e + (b/a)Q_1$$

Using the parameters above we can calculate this as 55 deg C which is in general agreement with the plot.

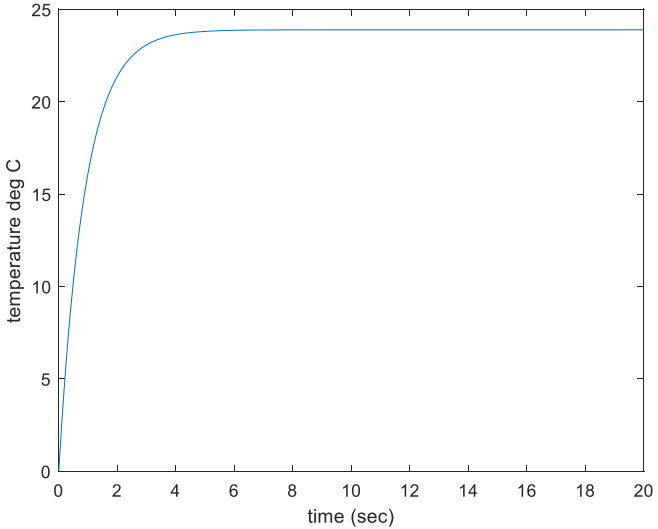
Closing the Loop

The plant is now turned into a subsystem and the 'loop is closed' outside of it. This is a simple process you will be shown in class.



Q_1 is no longer constant but we need to set the reference temperature (Tref in the figure, set to 25 deg C) and also the 'controller gain' Kp (set to 1). The new model is called **Temperature_02** and the script (now **CallTemperature_02**)is slightly modified with the new parameters.

The result is below



It seems reasonable, but there is an error in the steady-state value, it never quite gets to the required temperature! We will need to understand why and how to fix it!

Tutorial Questions (continued – mainly Simulink)

2. Create a simple Simulink model, that takes a constant value of 1 as input and integrates it to give the output $y(t)$. Call the model 'simple.slx' and run it without using a Matlab script. Use the blocks 'constant', 'integrator' and 'to workspace'. Use different values for the initial value $y(0)$. Check the result makes sense. Compare with the theoretical result $y(t) = y(0) + \int 1\, dt$.
3. Now you have had a chance to build a simple model yourself, switch attention to the thermal model introduced about. Build the open model (Temperature_01.slx) and check you get the same result as shown in the slides. You can copy/paste the Matlab commands given above and put them in your own m-file (CallTemperature_01.m).
4. Introduce the proportional feedback controller, preferably putting the plant into its own subsystem block (select the relevant blocks and right-click the mouse, select 'create subsystem').

5. Adjust the value of K_p to see if you can reduce steady-state error. Also modify the model and plot $Q_1(t)$. What's the effect of using a larger value for K_p? Explain what you see.
6. [optional] Change the controller so it has both proportional and integral terms are used – a 'PI controller'. This means the block K_p is replaced by a subsystem $K_p + K_i \times \frac{1}{s}$. You can choose $K_p = 1, K_i = 0.1$. What do you notice? Can you explain what happens?