MEC3079S: Control Systems

Chapter 1 — Introduction

Table of Contents

1.1 Live scripts

1.1.1 Using live scripts

These course notes have been written in `MATLAB Live Scripts` and comprise text, imagery, and mathematical development, as well as code that is used to generate a variety of useful visualisations. The intention is that you play around with the provided code to get a better understanding of the various concepts that we will cover.

The live scripts have been *sectioned off* (indicated by the thin blue horizontal lines such as just before Section 1.1.2) so that you can run code on a section by section basis. You can use `Run Section` using CMD+ENTER (macOS) or CTRL+ENTER (Windows), or using the `Run Section` button under the `LIVE EDITOR banner` at the top of your screen, as shown in Figure 1.1.

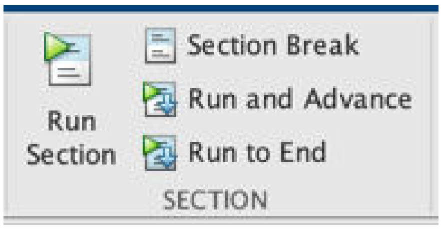


Figure 1.1: `Run Section` button that can be found in `LIVE EDITOR` banner at top of screen.

If you want to view the code, select the `Output inline` (figures appear below the text) or `Output on right` (figures appear on the right of the script) button on the top right of the script (see Figure 1.2). Otherwise, select `Hide code` (which will hide the code!).

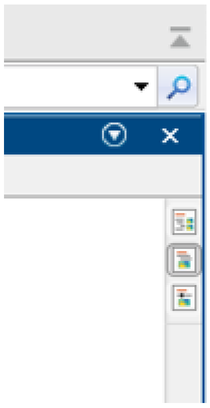


Figure 1.2: Live script view options, found at the top right corner of the script: Output on right (top button), Output inline (middle button), Hide code (bottom button).

This setting can also be adjusted when in the `VIEW` banner at the top of the screen, as seen in Figure 1.3. You can zoom in/out in `MATLAB live Scripts` using CMD+scroll (macOS) or CTRL+scroll (Windows). You can also zoom using the buttons under the `VIEW` banner at the top of the screen, as shown in Figure 1.3. Zooming may help if text/images are too big or small on your monitor.

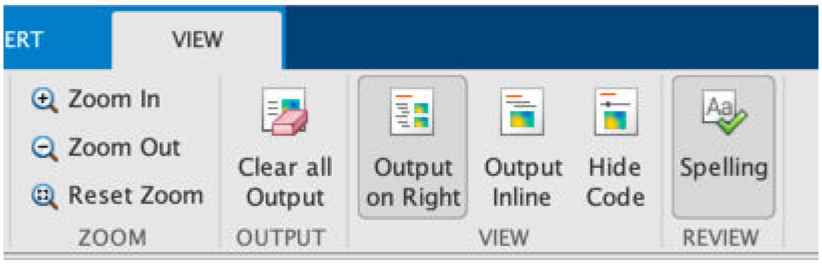


Figure 1.3: One can change the zoom settings or Live Script view options, such as hiding code when in the `VIEW` banner at the top of the screen.

1.1.2 MATLAB code blocks

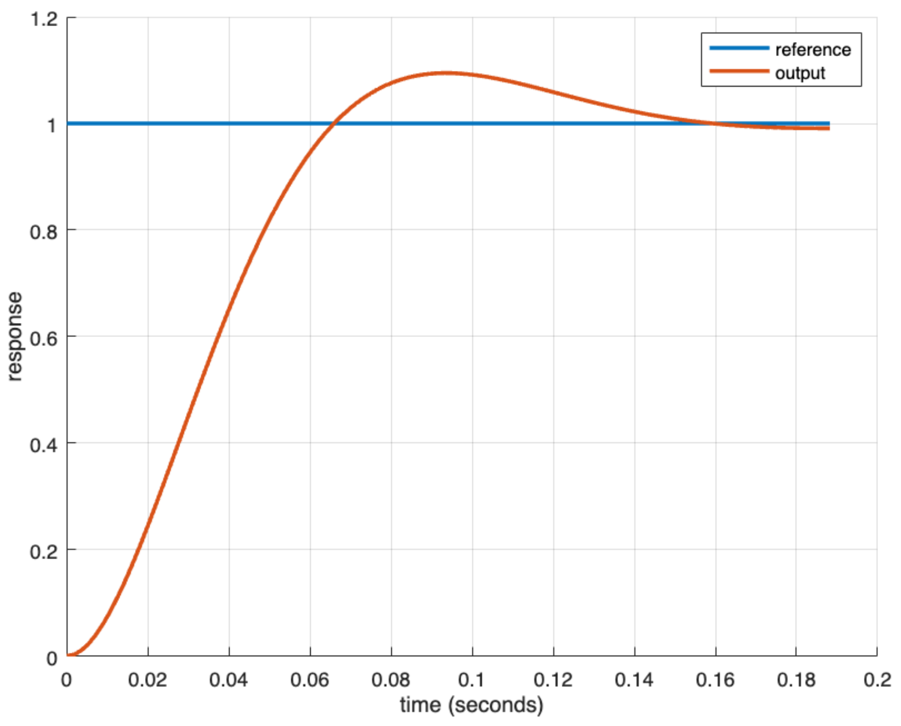
An example of a `MATLAB` code block is shown below. Code blocks will typically consist of parameters that you can change to observe how this affects some kind of visual response (in this case there are numerical sliders that you can play around with). Note that when you alter the slider, all the code blocks within the section will run automatically. You are also welcome to change other parts of the code as part of your experimentation!

```
s = tf('s');
```

```
wn = 42;
zeta = 0.6;

L = wn^2/s/(s+2*zeta*wn);
T = feedback(L,1)
```

```
T =

         1764
  -------------------
  s^2 + 50.4 s + 1764

Continuous-time transfer function.
Model Properties
```

```
[y,ty] = step(T);
[r,tr] = impulse(1/s,ty(end));

figure, hold on
plot(tr,r,lineWidth=2)
plot(ty,y,lineWidth=2)
grid on
xlabel('time (seconds)')
ylabel('response')
legend('reference','output')
```



You are not expected to understand the code above just yet, but hopefully in a few weeks this will make some sense to you! Note that you can find more information about a `MATLAB` function by using the `help` function:

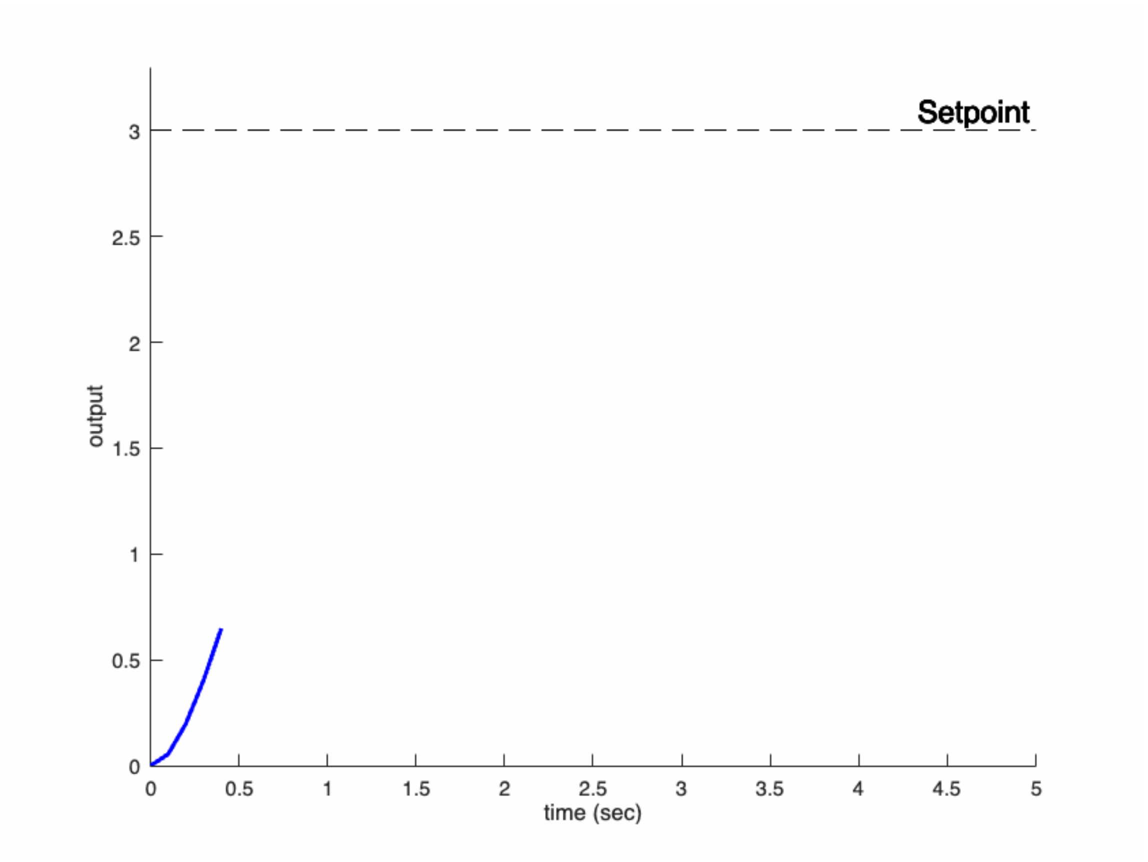
```
%Example
help feedback
```

```
 feedback  Feedback connection of two input/output systems.

    M = feedback(M1,M2) computes a closed-loop model M for the feedback loop:

           u --->O---->[ M1 ]----+---> y
                 |               |           y = M * u
                 +-----[ M2 ]<---+

    Negative feedback is assumed and the model M maps u to y. To apply
    positive feedback, use the syntax M = feedback(M1,M2,+1).

    M = feedback(M1,M2,FEEDIN,FEEDOUT,SIGN) builds the more general feedback
    interconnection:
                       +-------+
           v --------->|       |--------> z
                       |  M1   |
           u --->O---->|       |----+---> y
                 |     +-------+    |
                 |                  |
                 +-----[  M2  ]<---+

    The vector FEEDIN contains indices into the input vector of M1 and
    specifies which inputs u are involved in the feedback loop. Similarly,
    FEEDOUT specifies which outputs y of M1 are used for feedback. If SIGN=1
    then positive feedback is used. If SIGN=-1 or SIGN is omitted, then
    negative feedback is used. In all cases, the resulting model M has the
    same inputs and outputs as M1 (with their order preserved).

    If M1 and M2 are arrays of models, feedback returns a model array M of
    the same dimensions where
       M(:,:,k) = feedback(M1(:,:,k),M2(:,:,k)) .

    For dynamic systems SYS1 and SYS2,
       SYS = feedback(SYS1,SYS2,'name')
    connects SYS1 and SYS2 by matching their I/O names. The I/O names of
    SYS1 and SYS2 must be fully defined.

    See also lft, parallel, series, connect, InputOutputModel, DynamicSystem.

    Documentation for feedback
    Other uses of feedback
```

The help function also provides links to online documentation, which will give additional context and often examples.

1.2 Control systems

This course will introduce you to the wild and wondrous world of **control systems**. The intention is to expose you to various fundamental elements that can be used to both *analyse* and *design* control systems. A control system is a fairly vague term that can apply to a range of scenarios. We can think of a control system as a collection of various subsystems that interact with each other to achieve some pre-defined objective. The objective will vary depending on the particular system under consideration, and the means to achieve the goal will be dependent on specifications (usually supplied by a client) and the physical and/or financial limitations attached to the project.



Control systems are all around us and are a fundamental part of our daily lives. Some common examples of control systems are:

- the temperature control within an oven;
- the water level control in a toilet cistern;
- voltage regulation of the AC voltage supplied to houses (e.g. 220 volts);
- heading and velocity control of an aeroplane;
- the position and orientation control of an autonomous drone.

1.2.1 Control system definition

A control system consists of subsystems and processes (or *plants*) assembled for the purpose of obtaining a *desired output* with *desired performance*.

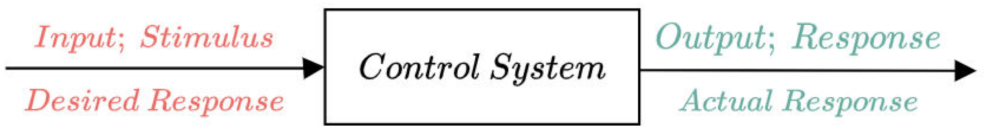


Figure 1.4: Visualisation of a high-level control system that takes in an input signal as a desired signal and outputs the actual response.

We represent

- **signals** (e.g. inputs and outputs) with an arrow, where the direction of the arrow indicates the directional flow of the signal,
- **systems** (processes or plants) with a block. The output of a system is a result of the interaction between the input signal and system.

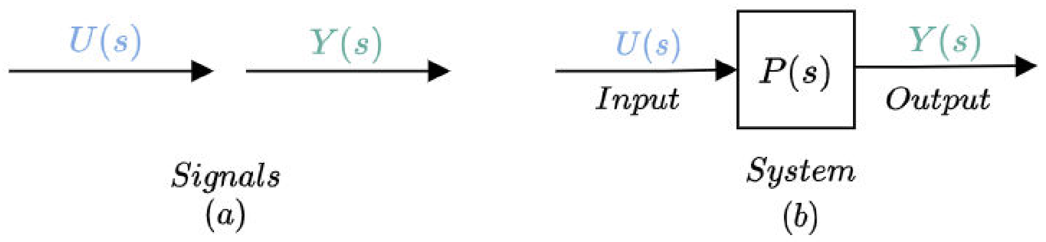


Figure 1.5: Visualisation of (a) signals using arrow and (b) systems using blocks.

Consider an elevator. When the fourth-floor button is pressed on the first floor, the elevator rises to the fourth floor with a speed and floor-leveling accuracy designed for passenger comfort. The push of the fourth-floor button is an **input** that represents our **desired output** (where we want the elevator to eventually get to), shown as a step signal in the figure below. The **performance** of the elevator can be seen from the elevator response curve in Figure 1.6.

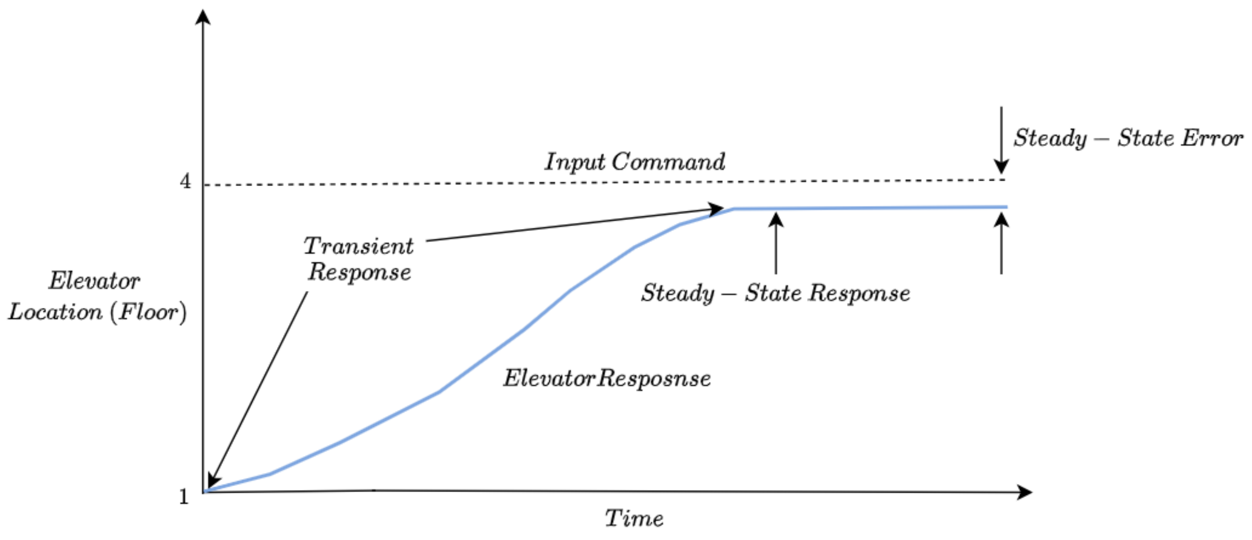


Figure 1.6: Time-domain response of an elevator after the 4th-floor button has been pressed.

There are two major measures of performance of any system in general:

1. Transient response (this is the behaviour of the elevator when in motion).
2. Steady-state error (the difference between the desired elevator position and actual elevator position after the elevator stops).

In our example, passenger comfort and passenger patience are dependent upon the transient response. If this response is too fast, passenger comfort is sacrificed; if too slow, passenger patience is sacrificed. The steady-state error is another important performance specification since passenger safety and convenience would be sacrificed if the elevator did not level properly. A third implicit consideration in control systems is the concept of stability. The transient response and steady-state error conversation would be rendered pointless if the elevator had an unstable response and ended up crashing into the ceiling of the elevator shaft! We will discuss all these topics in detail in the coming chapters.

1.2.2 Advantages of control systems

Control systems have many advantages. Some of the main advantages are summarised below.

1. Power amplification and signal conditioning — amplifying and filtering signals such that the best properties of the signal is maintained while avoiding amplifying signal noise (e.g. audio amplifier)
2. Automating systems — enabling systems to autonomously complete tasks without the need for human intervention (e.g. assembly line pick and place robot).
3. Uncertainty reduction — reducing the effect of parameter variations on the performance of a system (e.g. cruise control of a vehicle with an unknown number of passengers).
4. Compensation for disturbances — reducing the effect of disturbances on the control system (e.g. an aircraft autonomously compensating for crosswinds while landing).
5. Stability — maintaining stability of a system that would otherwise be unstable (e.g. angle control of an inverted pendulum).

1.3 History of control systems

Control systems date back thousands of years, with the first instances being purely mechanical in nature. Some notable milestones are summarised below.

1. Liquid-level control (~300 B.C)
2. Steam pressure and temperature control (1681)
3. Speed control (1745)
4. Stability and steering (1868)
5. Wright brothers (1903)
6. Space Race (1955-1972)

1.4 System configurations

There are two major configurations of control systems:

1. Open-loop control systems (often also referred to as feedforward control systems),
2. Closed-loop control systems (often also referred to as feedback control systems).

1.4.1 Open-loop systems

An input (often called a reference signal if it defines a desired value) is applied to the system via an input transducer. The controller drives the plant with the intention of achieving a particular output (controlled variable), as shown in Figure 1.7.

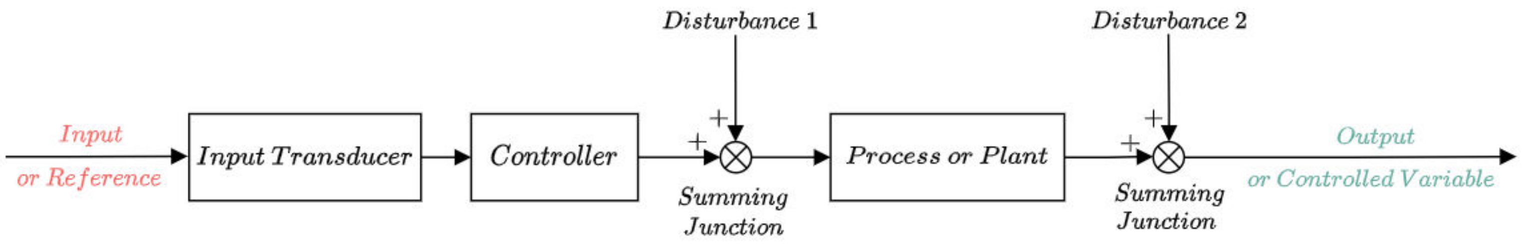


Figure 1.7: Block diagram of an open-loop system.

An **open-loop system** is cheap to implement, but it cannot compensate for any disturbances that act on the system. For example, toasters are open-loop systems, as anyone with burnt toast can attest. The controlled variable (output) of a toaster is the color of the toast. The device is designed with the assumption that the toast will be darker the longer it is subjected to heat. The toaster does not measure the color of the toast; it does not correct for the fact that the bread is rye, white, or sourdough, nor does it correct for the fact that toast comes in different thicknesses. We set the reference signal, which on a toaster usually corresponds to the "toasting time", and then hope that our guess of this input signal will result in a favourable output (an appropriately toasted piece of bread).

1.4.2 Closed-loop (feedback control) systems

In addition to the open-loop configuration, an output transducer (or sensor) measures the output response and converts it into a form that can be used by the controller, as shown in Figure 1.8.

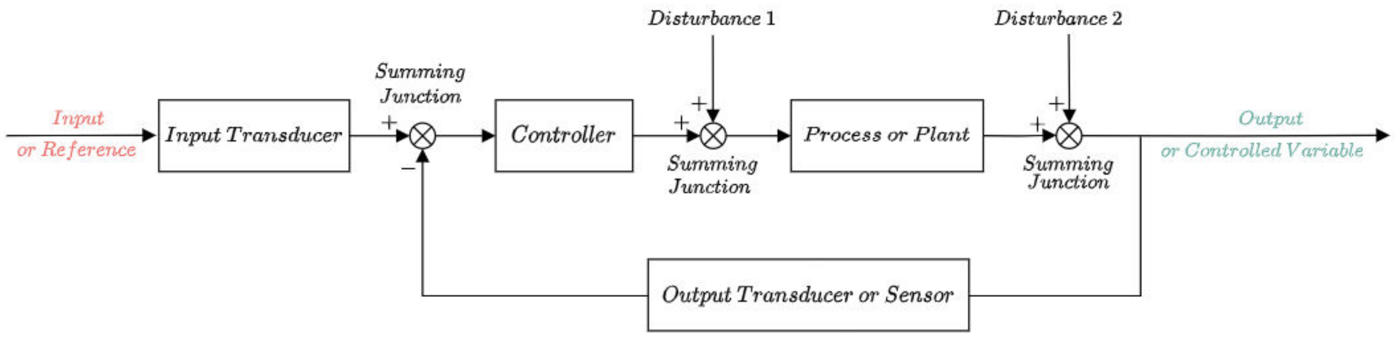


Figure 1.8: Block diagram of a closed-loop system.

A **closed-loop system** compensates for disturbances by measuring the output response, feeding that measurement back through a feedback path, and comparing that response to the input at the summing junction. If there is any difference between the two responses, the system drives the plant, via the actuating signal, to make a correction. If there is no difference, the system does not drive the plant, since the plant's response is already the desired response.

In the case of the toaster example, if we were able to measure the colour of the toast (e.g. with a camera), then we could feed this information back to the toaster in order to tell it when to turn off. This would constitute closed-loop control (albeit of a simple form).

Note that if we continually check the toast colour visually then we are in fact closing the loop on this operation as we are providing feedback in terms of when to stop the toaster.

1.4.3 Open-loop vs closed-loop

Another good comparison of open-loop and closed-loop control is to consider how modern toilets with cisterns refill after the toilet has been flushed. We can imagine that the task is to keep the water level at the desired level, but when the toilet is flushed this acts as a disturbance that results in the cistern quickly emptying. The question is then: how do we determine how long to open the valve from our water supply in order to refill the cistern, and how do we know if the water has reached the correct level?

An *open-loop* implementation, as shown in Figure 1.9a could make use of an estimated flow rate from our water supply to determine the time required to open the valve to let water in, in order to fill the tank to an appropriate level (this require basic integration based on the shape and dimensions of the tank). This seems like a plausible idea, but in reality the flow rate from the water supply will vary depending on the demand in the rest of the house (for example if someone else is showering at the same time). Therefore, we should expect that there will be some non-zero amount of steady-state error when relying on an open-loop control scheme to refill our toilet cistern. Note that this may still be acceptable if a non-zero steady-state error is within the specifications of our toilet.

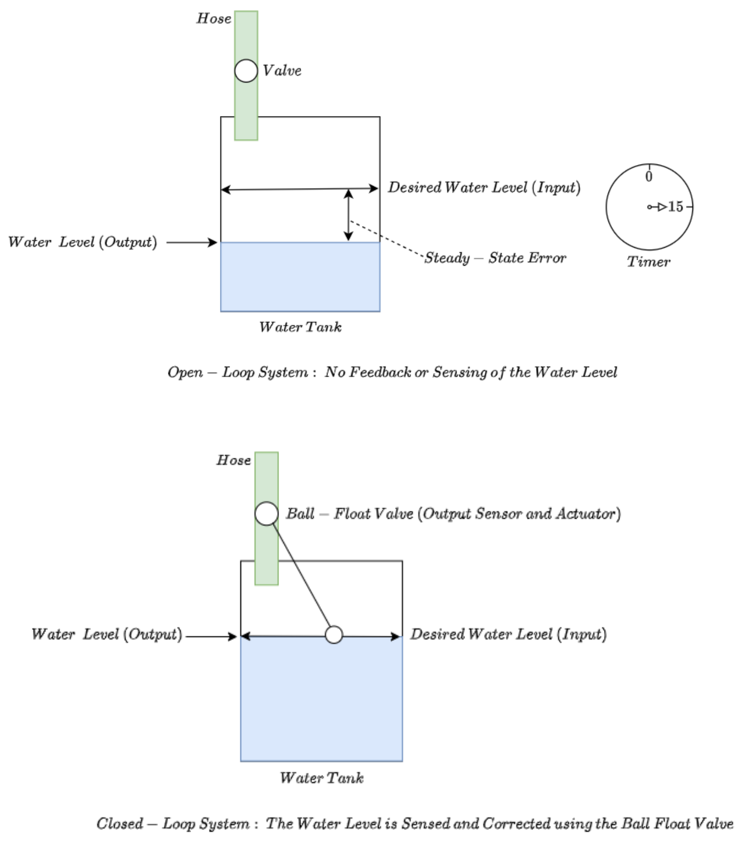

Hose
Valve
Desired Water Level (Input)
0
15
Timer
Water Level (Output)
Steady – State Error
Water Tank
Open – Loop System : No Feedback or Sensing of the Water Level
Hose
Ball – Float Valve (Output Sensor and Actuator)
Water Level (Output)
Desired Water Level (Input)
Water Tank
Closed – Loop System : The Water Level is Sensed and Corrected using the Ball Float Valve

Figure 1.9: Visualisation of (a) an open-loop toilet cistern and (b) a closed-loop counterpart.

A *closed-loop* implementation makes use of a ball-float valve as shown in Figure 1.9b. When the water level is equal to the desired water level, the float sits at the desired height and keeps the valve closed. When the toilet is flushed and all the water is emptied out the cistern, the float drops to a near-vertical position, which opens the valve. The water then rises based on the open valve, and will only close again when the water reaches the desired height. Note that while the time it takes to fill the cistern may vary depending on the other water loads in the house, the tank will eventually refill to the correct height. Note that while the closed-loop solution definitely seems more practical (and tolerant of uncertainty with regard to flow rate), we did have to add additional components (a ball-float in this case), so our closed-loop solution is more expensive financially. Whether the additional expense is justified would come down to the client requirements.

1.4.4 Computer-controlled systems

Modern control systems make use of digital computers, as opposed to using analogue components (e.g. resistors, capacitors, operational amplifiers). Both analogue and digital control systems exist today with varying advantages and disadvantages. The advantages and disadvantages of using digital control systems are summarised below.

Advantages:

- a single computer can control multiple control systems and control loops
- adjusting control parameters can be done "instantly" via software
- simulation and implementation can be tightly coupled using the same hardware/software environment

Disadvantages:

- digital sampling effects can cause performance and stability degradation
- at the mercy of software/hardware related shortcomings (e.g. stack overflows, processor stall, signal amplitude/bandwidth limits, ADC quantisation)

We will cover methods of converting control schemes into implementable algorithms at the end of this course.

1.5 Analysis and design objectives

1.5.1 Analysis

Analysis is the process by which a system's performance is determined and understood. For example, we evaluate a system's transient response and steady-state error to determine if they meet the desired specifications (usually provided by the client). At this stage, we also identify whether the system requires intervention (maybe the system already meets the client's specifications, thereby requiring no intervention) and if intervention is required, what type of control configuration is required (open-loop vs closed-loop).

1.5.2 Design

Design is the process by which a system's performance is adjusted as required. For example, if a system's transient response and steady-state error are analyzed and found not to meet the specifications, then we change control parameters or add additional compensators to meet the specifications.

Key considerations:

- Transient response — how quickly does the output response settle after a reference signal is applied, and how oscillatory/erratic is the response during this period.
- Steady-state response — what is the extent of the difference between the reference signal and output signal after the transient period has completed.
- Stability — does the output respond in a stable (non-divergent) manner as a result of all reference signals of interest.

Other considerations:

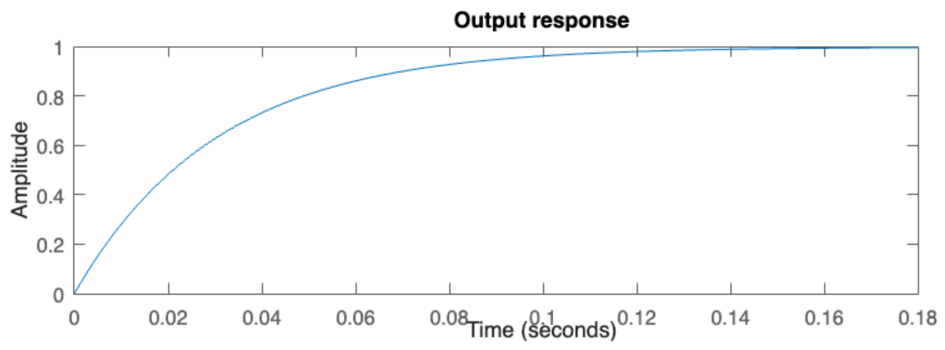
- Hardware selection — how do we select components (e.g. sensors, microcontrollers, actuators) such that the system specifications will not be hindered.
- Cost of design — balancing the performance specifications against the cost of the additional components required to achieve the specifications.
- Robust design — designing the control system such that it will perform within the required specifications regardless of disturbances and parameters variations in the system.

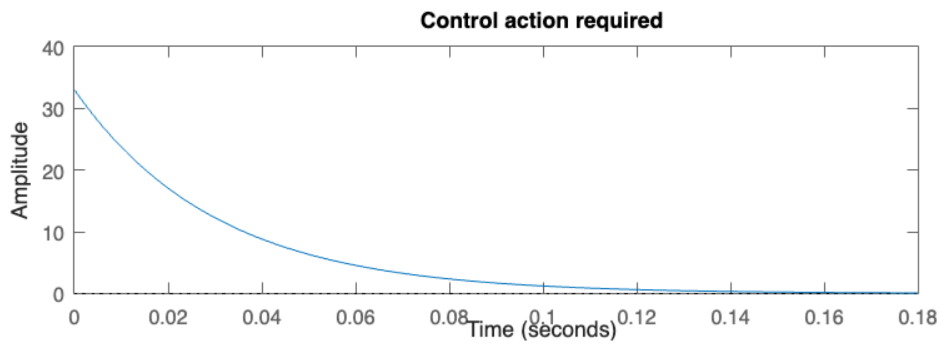
1.5.3 Design trade-offs

As we will see later in the course, we will have to make a trade-off between the performance we can achieve, and the amount of energy that we can inject into our system. For example, we can try design a control system for a Toyota Corolla that will make it go from 0 km/hr to 100 km/hr in a few seconds, but we will end up being fundamentally limited by the amount of power that our engine can produce relative to the car's mass.

The example below shows that we can make our output response as quick as we want, but at the cost of increased control action. The control action is usually a physical signal, such as a voltage, force/torque, flow rate, or pulse width modulated signal. As with any real-world system, there will be limitations on the control action that is achievable — e.g. the voltage supplied to a system is limited by the power supply available, or the torque produced by an engine is limited by its design.

```
s = tf('s');
P=1/s;
G = 33;
Ty = P*G/(1+P*G);
Tu = G/(1+P*G);

figure
subplot(2,1,1), step(Ty),title('Output response')
subplot(2,1,2), step(Tu),title('Control action required')
```





1.6 The design process

The **design process** often takes the form as detailed below:

1. Convert user requirements into engineering specifications.
2. Mathematically model the system under consideration and represent the model using a block diagram (or some other equivalent visualisation).
3. If available, confirm the model representation using measurement data.
4. Analyse the system using control analysis techniques against the required specifications and identify the required control scheme.
5. Design a control scheme and test its viability.
6. If the designed control scheme meets the specifications, then the design process is complete, otherwise return to Step 4.

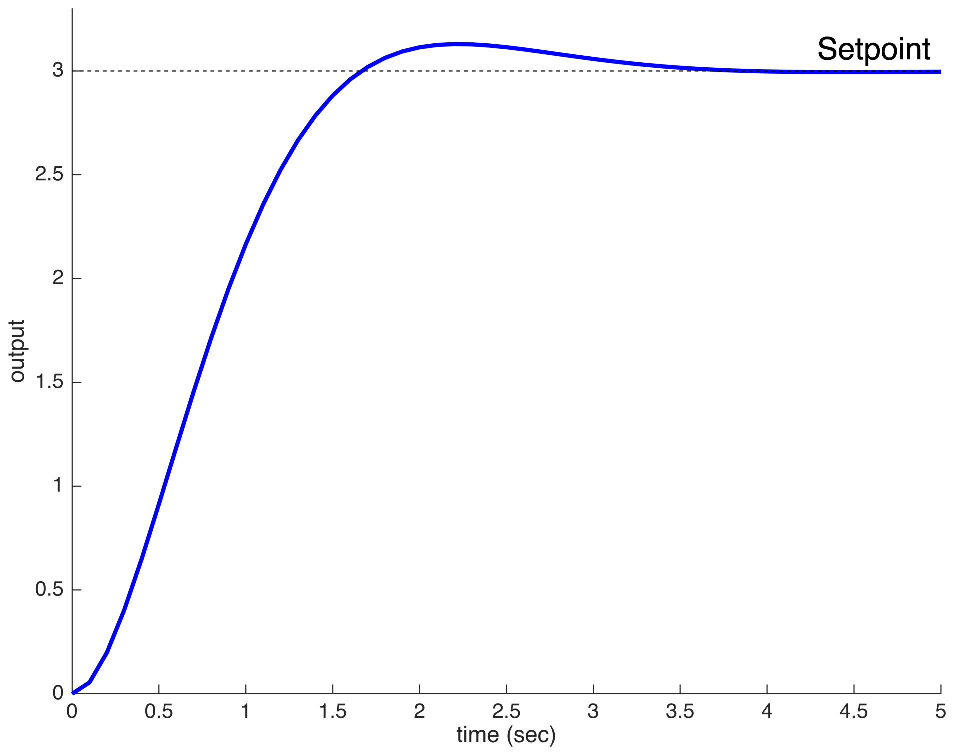
1.7 Computer-aided design

The computer is an integral part of modern control system design, and many computational tools are available for your use. In this course we use `MATLAB` and the `MATLAB Control System Toolbox`, which enables us to quickly analyse, design, and test control systems.

The code block below provides a simple example of computer-aided design.

```
s = tf('s');

w = 2;
z = 0.7071;
P = 1/(s^2/w^2+2*z/w*s+1);
```

```
dT = 0.1;
ty = 0:dT:5;
U = 3;
[y,~] = step(P*U,ty);

timer = rateControl(1/dT);

figure, hold on,
axis([0 5 0 1.1*U])
xlabel('time (sec)'),ylabel('output')
for i=1:length(ty)
    plot(ty(1:i),y(1:i),'b-',lineWidth=2)
    yline(U,'--k','Setpoint',FontSize=15)
    drawnow
    waitfor(timer);
end
```



1.8 The control systems engineer

Control systems engineering is an exciting field in which to apply your engineering talents, because it cuts across numerous disciplines and numerous functions within those disciplines. The control engineer can be found at the top level of large projects, engaged at the conceptual phase in determining or implementing overall system requirements. These requirements include total system performance specifications, subsystem functions, and the interconnection of these functions, including interface requirements, hardware and software design, and test plans and procedures.

1.9 Summary of course content

The course contains the following chapters:

1. Introduction
2. Signals and Systems
3. Block diagrams
4. Modelling in the Laplace domain
5. Prediction of system response
6. System characterisation
7. Frequency response techniques
8. Feedback control systems
9. Stability of closed-loop systems
10. Steady-state error
11. Transient design specifications
12. Frequency-domain design
13. Digital control systems

Chapters 1-4 constitute *fundamental* theory development that will be required to tackle the latter chapters.

Chapters 5-10 are involved with the *analysis* of control systems, with **Chapter 8-10** focusing on feedback control systems.

Finally, **Chapters 11-13** (and **Chapter 10** to an extent) are concerned with feedback control *design* methodologies.