Control Engineering for High School Students and Teachers: An Online Platform Development

Farhad Farokhi and Iman Shames

Contents

1 Introduction

Based on our teen years and feedback from many of our colleagues and friends, we believe that the control engineering, although being a major building block of automated system in many processes and infrastructures, is a fairly alien subject to the students, parents, and teachers. Therefore, there is a need for introducing feedback control and its application to high school students and their teachers to recruit the next generation of engineers and scientists in this field. We also believe that the academic community has a responsibility to disseminate the information cheaply, if not freely, to a wide range of interested audience, be it students, parents, or teachers, across the globe. This way, we can guarantee that people from different socio-economic backgrounds and in different countries can make informed decisions regarding their careers and those of their friends and families.

Motivated by these needs, in this project, we have attempted at developing an online platform for the students and their educators to read about the control engineering, watch lectures by researchers from academia and industry, access interviews with successful people in the control engineering community, and play online games to test their understanding and to possibly learn about the applications of the automatic control. The platform also contains a remote laboratory for the students to see control engineering in practice.

Figure 1 shows the landing page of the online platform. This page leads the students (or any other users) to various parts of the platform containing the relevant information, such as short courses on control engineering, online games, interviews, remote laboratory, and a news blog. In the rest of this report, these elements are discussed.

2 Courses

A fundamental part of a successful platform for the students to learn the basics of the control engineering, albeit in a simplified form. The covered concepts could contain introduction to

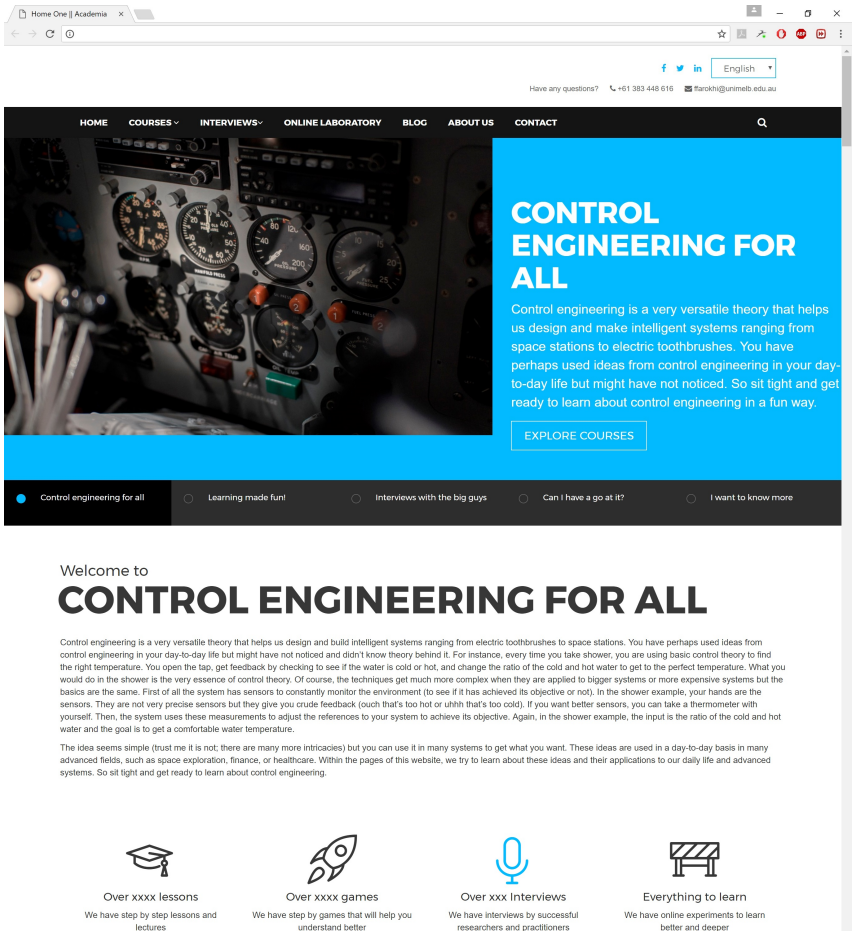


Figure 1: The landing page of the online platform.

feedback control, history of control engineering, modelling, sensors, actuators, on/off control, stability, performance, proportional control, etc. Figure 2 illustrates the menu from which the users can select the appropriate course.

An important issue when developing the courses is to keep the discussion at an appropriate level for senior high school students and undergraduate students. This way the material can engage the most number of people. Further, the material can excite younger students that are slightly ahead of their cohort. Each course is structured as in Figure 3. That is, it starts with an introduction on the topic. There are also videos, gathered from open sources, that discuss that topic. Then, the material is presented and examples, numerical or physical, follow that.

It should be noted that the undergraduate level of material does not mean that all mathematics should be avoided; however, the level of the mathematics should be tailored to the audience. In the platform, it is decided that the level of the mathematics is kept at high school and/or undergraduate calculus. Examples of the content of the platform and the level of the mathematics for first three courses of the platform are given in Appendices A–C.

Finally, graphical interfaces are utilized to help with the understanding of the concept. For instance, in the course discussing on/off controllers and their application, a simple graphical interface is used to show the effect of hysteresis or the band in the performance of the on/off controller. See Figure 8 in Appendix C.

3 Interviews

Another important part of this platform is to introduce the biggest names in the control engineering community, both in academia and industry, to the young students. This way, the

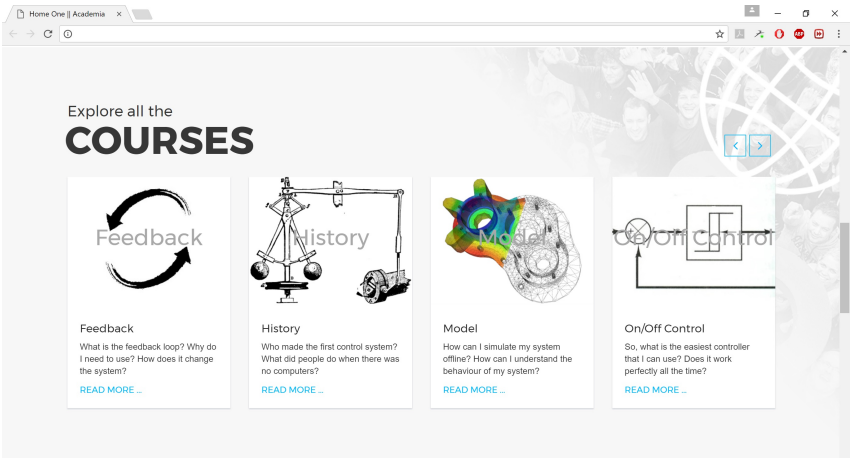


Figure 2: The list of courses offered on the online platform.

Figure 3: An example of course on the platform.

students can see the potential trajectory for their success in the future. Currently, the platform only features links to the IEEE Control System Society (CSS) interviews with the researchers in the field of control engineering. However, in the future the interviews will also be populated with many early-career researchers and engineers. The webpage also encourages the students to interview the control engineers in their vicinity. Figure 4 illustrates the interview panel of the online platform.

4 Educational games

In the online platform, the use of educational games is also highly encouraged to help with improving the understanding of the students. Understanding the effect of the delay in the closed-loop system is a subject for which a game is developed. In this game, the objective of the user is to control the distance of a satellite from the earth. The satellite is effected by the gravitational forces as well as space objects that might hit it (i.e., disturbances). In this game, the level of the difficulty is a function of the delay between sending a command by the user and observing its effect on the satellite's movements. As expected, the users have a difficult task at hand as the delay grows. Figure 5 illustrates the starting page of the game.

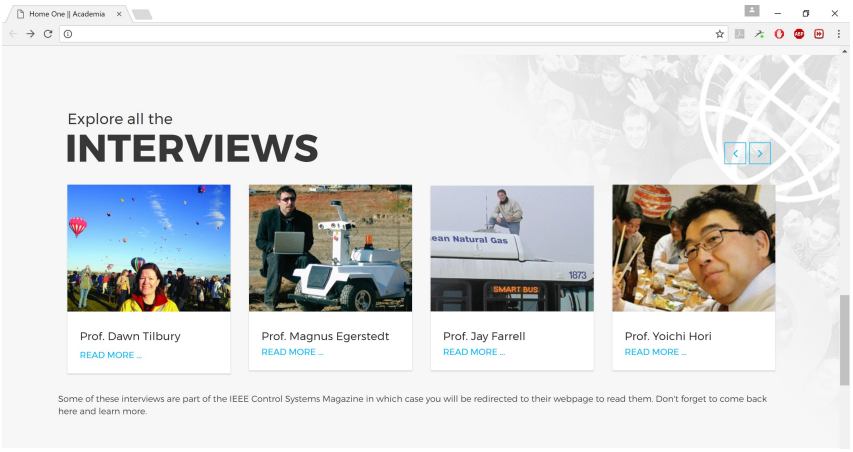


Figure 4: Interview panel of the online platform.

5 Remote laboratory

Remote laboratory (RLAB)[1] is a software package designed to aid development of remote-access laboratory experiments by providing a web interface to laboratory apparatus that manages the access, queuing and data-retrieval for experiments. In what follows, an overview of the design of RLAB is presented. Instructions on how to install and develop such laboratory experiments is provided in Appendix D.

The term *apparatus* is used to refer to a physical equipment on which an experiment is run. The term *experiment* is a particular procedure that can be performed on an apparatus. Note, each apparatus could perform any number of experiments. It is assumed that every remote-access experiment is designed such that it can be initialized, controlled, and measured via a digital computer. Here, we presume that a *Raspberry Pi* (RASPI) is the digital computer interface used by all apparatus, however, the RLAB software could be modified to be run on any digital computer. The apparatus that is going to be deployed in this remote laboratory is presented in Figure 6.

As shown in Figure 7, the RLAB software runs on Raspberry Pis that are co-located with a particular lab apparatus. The RLAB software serves webpages to allow users and administrators perform certain tasks as described in the following subsections.

Queuing experiments

The webpage `/queue_expr/<expr_id>/` provides a form that shows all the parameters that can be modified when performing the experiment named `<expr_id>`. By submitting this form, the user's particular experiment setup is placed into a queue. In the background the RLAB software then queues each experiment sequentially and sends a notification email on completion to the user. This email contains a link which instructs the user where to download the results, which includes the experiment setup, results, score and a video.

Retrieve experiment results

Once an experiment has completed, the RLAB software serves the files at the web address `/downloads/<filename>/`, a link to this location is provided in the experiment complete notification email. By default the expiry time for this file is 24 hours, however, this time can be modified by the admin.

[1]This part is developed by Mark M. Fabbro, a casual engineer at the Department of Electrical and Electronic Engineering at the University of Melbourne, hired with the outreach grant provided by the IEEE CSS.

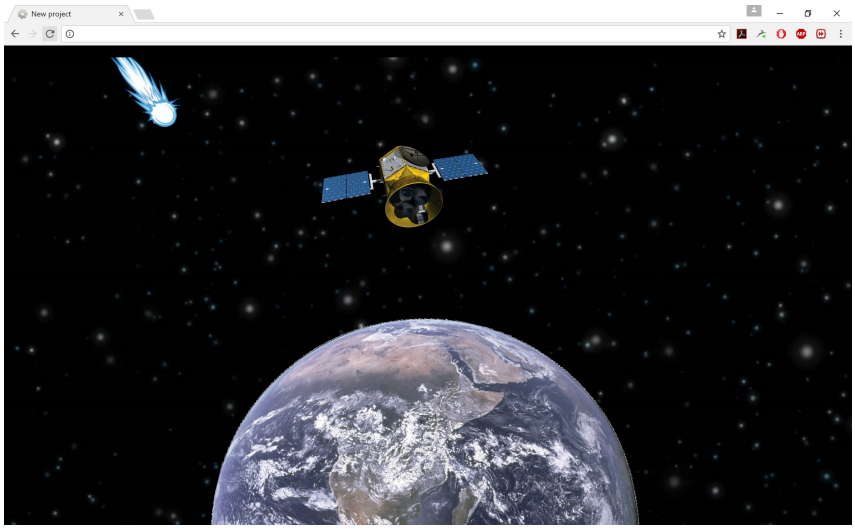


Figure 5: An educational game for demonstrating the effect of the delay in control of systems.

Compare results via scoreboard

Some experiments have a suitable notion of a *score*. When this is the case the RLAB software serves a web page located at `/topscores/<expr_id>` that provides that top 5 scores and an average score for the experiment. This feature is used to make the remote laboratory competitive and more interesting to the users.

Administrative tools

The RLAB software provides a web page `/admin_tools/` which allows the administrator to (i) enable/disable experiments, (ii) restrict experiment access and, (iii) reset the scoreboard. The enable/disable experiments feature allows the administrator to stop users from queuing a particular experiment on an apparatus. This is useful in the context of university workshops, where certain experiments can be made available only after a certain date in semester. Moreover, if an experiment fails the RLAB software immediately disables the experiment and notifies the administrator via email. The restrict experiment access feature allows the admin to submit a CSV list of all user names that have permission to submit a particular experiment. This feature was included for the case where lab equipment is sensitive and should only be used by researchers or specific students.

6 Conclusions and future work

Unfortunately, the project's implementation has been delayed due to multiple reasons including technical difficulties (regarding hosting the online remote laboratory platform and cyber-security issues regarding the use of the university's network). In addition, due to the initial delay in review of the grant, the preliminary plans of the investigators were changed and further delayed. Furthermore, since the webpage of the online platform is being developed internally, lack of access to proper rights for the use of pictures and videos has also stalled the completion of the web front. Hence, at the current time, the project is still not finished and is ongoing. However, recently, the progress has been steady. It is anticipated that the project finishes within 6-9 months.

An important step is to finish the web front for the online platform. This allows us to make the webpage live. Further, a physical apparatus needs to be integrated into the remote laboratory so that the users can implement various control strategies on the robot for assessing

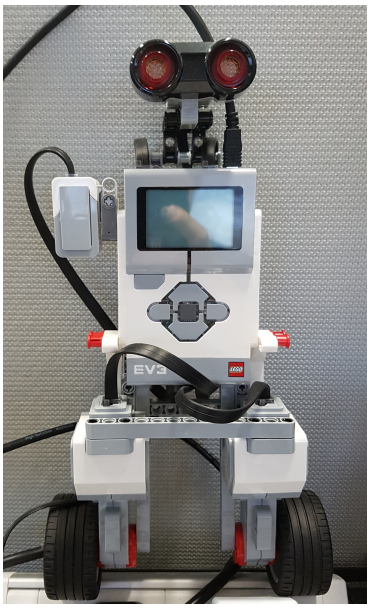


Figure 6: A Lego segway is used as an apparatus in the remote laboratory.

their knowledge of the subject. Experiments need to be designed for this part that are in the range of the knowledge of the students.

A Example course 1: Feedback theory

Feedback Theory

Control engineering is a versatile theory that helps us design and build intelligent systems ranging from electric toothbrushes to space stations. You have perhaps used ideas from control engineering in your day-to-day life but might have not noticed or didn't know the theory behind it. For instance, every time you take a shower, you are using basic control theory to find the right temperature. You open the tap, get feedback by checking to see if the water is cold or hot, and change the ratio of the cold and hot water to get to the perfect temperature. This is the very essence of the control theory. Of course, the techniques get much more complex when they are applied to bigger or more expensive systems (there is no room for a mistake there) but the basics are the same. First of all, we use a sensor to constantly monitor the environment and the underlying system (to see if we have achieved our objective or not). In the shower example, your hands are the sensors. They are not very precise but, nonetheless, they act as sensors. They give you a crude feedback of the system (ouch that's too hot or uhhh that's too cold). If you want better sensors, you can take a thermometer with yourself. Then, the controller uses these measurements to adjust the references to the system to achieve our objectives. Again, in the shower example, you and your brain are the controller, the input or the reference is the ratio of the cold and hot water, and the objective is to get a comfortable water temperature.

The idea seems simple (believe me when I say there are many intricacies) but you can use it in many systems to get what you want. These ideas are used on a day-to-day basis in many advanced fields, such as space exploration, finance, healthcare, and robotics. Within the pages of this website, we try to learn about these ideas and their applications to our daily life and advanced systems. So sit tight and get ready to learn about control engineering.

> Control engineering is a versatile theory that helps us design and build intelligent systems ranging from electric toothbrushes to space stations.

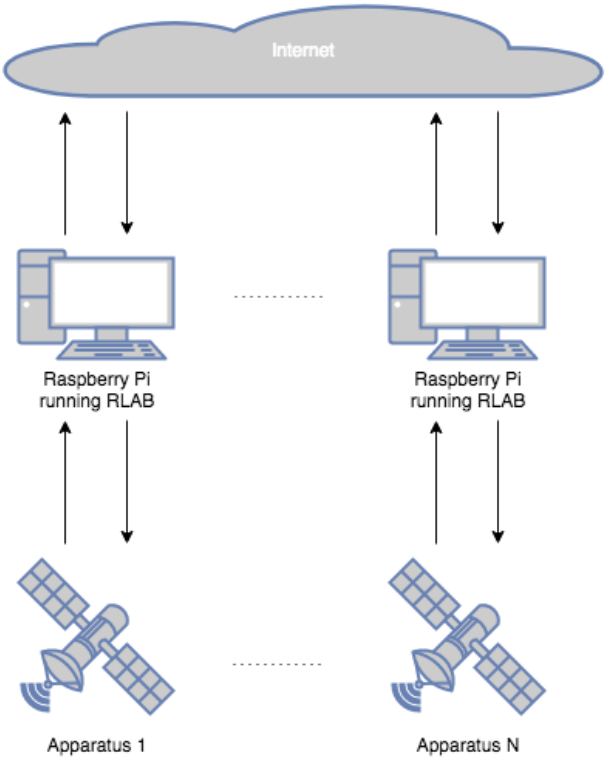


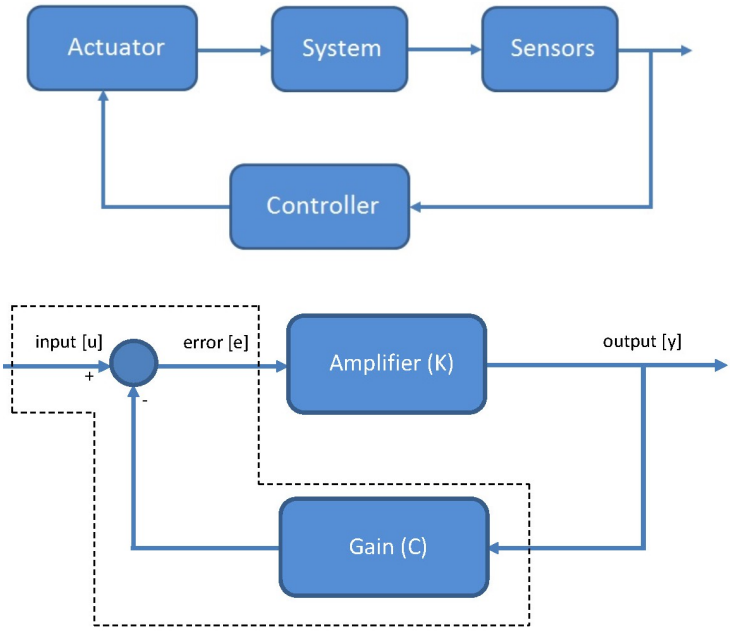
Figure 7: Remote laboratory software architecture.

Open loop versus closed loop

The terms open loop and closed loop are technical terms (jargon) that are used on by control engineers. Open loop simply means that we do not use measurements from the sensors to adjust the inputs of the systems. In these systems, we typically sit down and exactly calculate "how much" input is enough to get to where we want and then provide that input to the system. To give you a visualization, open loop control is like driving without looking. That metaphor probably gives you the sense that open loop control is dangerous, which is a good intuition! Open loop control works if the environment is not changing a lot (for instance when you are driving on a completely empty straight highway) and you know the system very good (you are using a car that you have driven for the past several years). But, in practice, there are always uncertainties: wind, tire pressure imbalances, etc. In that case, we would like to close the feedback loop by constantly monitoring the sensors and adjusting the inputs or references. This methodology is, perhaps unsurprisingly, called closed loop control. In the driving example, if you keep your eyes open, you are constantly using your sensors, your eyes, to measure your distance with pedestrians and other vehicles and adjust your input (accelerating or braking).

> ... in practice there are always uncertainties ... [therefore we should] close the feedback loop by constantly monitoring the sensors and adjusting the inputs ...

Figure above shows a schematic diagram of the closed-loop control. System is what we would like to control like the car. Sometimes, people call them plants (as in power plants, not trees). Sensors are the elements that measure various variables in the environment and the system. They tell us what is happing in reality. In the driving example, our eyes are acting as sensors. Some advanced cars have proximity sensors to alert us if you are getting too close to an obstacle when driving in reverse. Those are also sensors. The controller is the element we are using to convert the data from the sensors to the adjustments of the inputs to manipulate the behaviour of the system towards what we want. So far, in all the examples that we discussed, you and your brain have acted as a controller but the controller can be a computer, an especially designed



chip, or even a piece of metal. We will see examples of all these controllers in what follows. Then, the command from the controller is passed to an actuator. Actuator is a device that acts on the system. In the driving example, our feet and hands are actuators. They change the inputs to the system. Actuators can be motors, valves, or other objects. We will see examples of the actuators in the future too. Sometimes the actuator and the controller are packed in a single box as the controller.

Effect of uncertainty

In fact, feedback loop was popularized, in part, due to the work of Hendrik Wade Bode while working with amplifiers at Bell Labs. Amplifiers, if we model them at a very basic level, are static gains. You will feed them a signal (like your voice in a microphone) and they increase its amplitude to a level that can be transmitted over wires and then fed to a speakers. The problem that Bode was facing was that the gains of these amplifiers was highly variable due to their design process and the environmental factors (such as temperature) among many other factors. The gain could change significantly and that would make their products unusable (a volume of the voice on your phone could have deafen you one day while being fine another day). So what he did was to use feedback. Consider the elementary feedback loop in the figure below. This feedback loop is a bit different from the one that we discussed eariler but it has the same structure.

First, the system here is the amplifier. The system multiplies everything that it receives, in this case e, with the gain K. Here, for simplicity, we have dropped the actuators and the sensors. The controller is contained inside the dashed lines. The controller gets the output y, multiplies it with the gain C, and then compares that with the input u to construct the error $e := u - Cy$. Substituting $e = u - Cy$ inside $y = Ke$, we get

$$y = K(u - Cy).$$

Now, if we solve this equation for y, we get

$$y = \frac{K}{1 + KC}u.$$

So, the gain of the closed-loop system is $T := K/(1 + KC)$. Let's have a numerical example. Let say $K = 100$ and $C = 0.1$. The gain of the closed-loop system, based on our calculation, is

$$T = \frac{100}{1 + (100)(0.1)} \approx 9.091.$$

Now, let's say something happens and the gain of the amplifier significantly changes to $K = 1000$. So, if we were using this amplifier, our ears would suddenly ring. In this case, the gain of the closed-loop system becomes

$$T = \frac{1000}{1 + (1000)(0.1)} \approx 9.901.$$

That significant change does not change the gain of the system with a feedback loop noticeably. In fact, for all large K, we have that

$$T \approx \frac{1}{C},$$

which is independent of K. That was great news for Bode because the feedback gain C for his amplifiers were designed using resistors and they were very accurate. We can define the sensitivity S as the changes in the gain of the closed-loop system T over the changes the gain of the amplifier K. We get

$$S = \frac{9.901 - 9.091}{1000 - 100} = 0.0009 = 0.09\%.$$

If there was no feedback the sensitivity was 100% (meaning the changes in the gain were completely observable in the output) but now the sensitivity is less than a tenth of the percentage point! These calculations were done for a very simple system but the observations can be generalized to many more complex systems.

B Example course 2: Models

Introduction

Models are mathematical objects that we develop to be able to replicate the behaviour of the system in a computer (called simulation) or to understand the behaviour of the system (like what variable excites what part of the system). We can develop these models by laws of physics (first principle models) or we can try to fit a model to a system. For instance, if you have a robot that moves freely in the space, we can model it using Newton's laws of motion. Let's say we have a robot that can only move left and right and its position is given by x. Doing so, we get

$$\ddot{x}(t) = \frac{1}{m}F(t),$$

where $F(t)$ is the force applied to the robot by its motor (so, technically, the input the system) and m is its mass. If we wanted to digg deeper and model the wind friction, we would get

$$\ddot{x}(t) = \frac{1}{m}[F(t) - b\dot{x}(t)].$$

That is because the force excreted by the wind friction is proportional to the velocity (which is the derivative of the position) $\dot{x}$. These models are called continuous time models (because we look at the object continuously in time). But let us consider a different case, where we look at our robot only once per second. This way, we get

$$x[k + 1] = x[k] + (F[k]/m)\Delta t,$$

where $x[k]$ denotes the positions of the robot in the k-th second and Δt is the time between two consecutive updates, which is 1 sec in this example (but can be different in other cases). The

expression above is correct, of course, if the force F is constant between two updates. It is good approximation even if F is slowly varying (certainly slow considering the sampling time ΔT). This is called a discrete-time model. Sometimes, it is easier to work with the discrete-time model and, sometimes, it is easier to work with the continuous-time model. There are cases that you have to work with a discrete-time model. For instance, inside a computer, since computations are done in sequence and time is discreteized, discrete-time models should be used.

What does the model tell us?

Let us consider a very simple model and study its properties. Since dealing with continuous-time models is more difficult with a limited mathematics knowledge, we consider a discrete-time model. Consider a single room and a air-conditioning (AC) unit in

$$T[k+1] = T[k] - \frac{1}{R_i C_i}(T[k] - T_o + u[k])\Delta t,$$

where $T[k]$ is the temperature of the room at k-th time instant, T_o is the ambient (outside of the house, the other rooms, etc.) temperature at that instant, $u[k]$ is the effect of the AC unit, R_i is the thermal resistance (^{o}C/kW–read this as Centigrade degree per kilo Watt) of the room, and C_i is the thermal capacitance (kWh/^{o}C–read this as kilo Watt hour per Centigrade degree). The last two, R_i and C_i, are constants that relate to the size of the room, the material that is used to build and insulate the walls, and the furnitures inside the room. The same as before, Δt is the time between two consecutive updates. Let define a parameter $a := \Delta t/R_i C_i \leq 0$ to make the model easier to write down. We get

$$T[k+1] = (1-a)T[k] + aT_o - au[k].$$

This model allows us to simulate the behaviour of the room as a response to the AC's input $u[k]$. We can write a code in the computer to simulate this behaviour. The model however give us other insights about the system without the need for simulation.

Assume that the AC is off. That is, $u[k] = 0$. The model becomes

$$T[k+1] = (1-a)T[k] + aT_o.$$

An interesting point is an equilibrium. An equilibrium of the system T is a point that if we initialize the system there $T[0] = T$, it says there for ever $T[k] = T$ for all $k \geq 0$. Alternatively, we can say that T is an equilibrium if $T[k] = T$ implies that $T[k+1] = T$. What's the equilibrium of the simple system above? By the definition, if T is an equilibrium, we get

$$T = (1-a)T + aT_o.$$

If we solve this equation for T, we get that $T = T_o$. This shouldn't be surprising to you. This means that if the temperature of your room is the same as the outside temperature, the temperature of your room stays the same as the outside temperature for ever. Let us try to understand more about the dynamics of the room temperature. With a little bit of mathematics, we can see that

$$\begin{aligned}
T[1] &= (1-a)T[0] + aT_o \\
T[2] &= (1-a)T[1] + aT_o \\
&= (1-a)((1-a)T[0] + aT_o) + aT_o \\
&= (1-a)^2 T[0] + ((1-a)a + a)T_o \\
T[3] &= (1-a)T[2] + aT_o \\
&= (1-a)((1-a)^2 T[0] + ((1-a)a + a)T_o) + aT_o \\
&= (1-a)^3 T[0] + ((1-a)^2 a + (1-a)a + a)T_o.
\end{aligned}$$

If we follow all the calculations up to the k-th time instant, we can write $T[k]$ as a function of the initial condition $T[0]$ and the ambient temperature T_0:

$$T[k] = (1-a)^k T[0] + ((1-a)^{k-1} + \cdots + (1-a)^2 + (1-a) + 1)aT_o.$$

Using the properties of the geometric sequences, we get

$$(1-a)^{k-1} + \cdots + (1-a)^2 + (1-a) + 1 = \frac{1-(1-a)^k}{1-(1-a)} = \frac{1-(1-a)^k}{a}.$$

Therefore, we have

$$\begin{aligned} T[k] &= (1-a)^k T[0] + \frac{1-(1-a)^k}{a} aT_o \\ T[k] &= (1-a)^k T[0] + (1-(1-a)^k)T_o. \end{aligned}$$

If we selected Δt small enough so that our model is realistic, $0 < 1-a < 0$, which means that

$$\lim_{k\to\infty} (1-a)^k = 0,$$

and, as a result,

$$\begin{aligned} \lim_{k\to\infty} T[k] &= \lim_{k\to\infty} (1-a)^k T[0] + (1 - \lim_{k\to\infty} (1-a)^k)T_o \\ &= T_o. \end{aligned}$$

Again, not surprisingly, this means that no matter what is room temperature now, as time goes by the temperature of the room converges to the ambient temperature. This is something that we see everyday. If it is a hot day outside, when we turn off the AC, the room gets warmer until its temperature becomes the same as the outside. These observations are however extremely valuable if we model a very complex system. The model can also be used to design a controller and to evaluate how good our controller is.

Modelling uncertainty

When we model a system, there are many phenomena that we might no consider. This could be because we do not know about them or because if we consider them the model becomes so complex that no one really can use it. For instance, in our model for the temperature of the room, we can go extremely detailed and consider the fact that the temperature of the air at different locations in the room is different. When we pump warm air into the room, we need to wait until diffusion (a fancy name describing the fact that the warm air moves and mixes with the cooler air) does its magic and the temperature of the whole room increases. We also do not consider the effect of the furniture in the room. They absorb energy and their temperature rises. People also change the model. On top of all these neglected effects, we might not be completely sure about some of the parameters in the model. We can perform experiments to measure these parameters but, at the end of the day, we might only have access to noisy versions of these parameters.

All these simplifications and randomness creates a model uncertainty, that is, our model might not completely reflect the behaviour of the room. For instance, using our model, we might expect that the temperature of the room reaches a certain value after half an hour but in practice if we turn on the air conditioner we might need to wait a few more minutes. This however doesn't matter that much if we are using a feedback loop to regulate the temperature of the room. As we discussed (and to some extent saw) in the lesson about feedback theory closing the control loop makes the system much more robust to the changes of the parameters and uncertainties.

C Example course 3: On/off control

On/Off Controller

This is the easiest controllers that one can employ. It is also one of the most used controllers. This control scheme is used when inputs to the system can only take two values: being on or off. For instance, let's say our control problem is to regulate the temperature of our house. In our house, we only have an old fashioned air conditioning unit, where we can't set a temperature reference. So our input to the system is either to turn the air conditioning system on or to turn it off completely. How can we regulate the temperature of the house around our desired point, say 20 degree Celsius? You know the answer perhaps but let me right it. You would turn the air conditioning unit on until the temperature of the house is close to your desired value. Then, you would turn it off. You repeat the procedure again as the temperature gets above your desired point. As you would guess, if you do this, you would have to turn the air conditioning unit on and off quiet rapidly. So, you cool down the room to a temperature slightly below your desired point and would turn the unit on again after the room temperature has passed a slightly higher temperature than your desired point. This way, the room temperature is never at the desired point but it is always in a band around it. The narrower the band, the more frequently you need to switch between on and off positions. This is a very simple and effective mechanism for control but it has a couple of cons. First, it cause wear and tear on the actuators (turning your air conditioning on and off all the time in rapid successions can make it break faster). Second, the error in achieving what you want can be too high. Yes, it is fine to have this controller for regulating the temperature inside a house (a couple of degrees above and below are not even noticeable by us) but these levels of error are not justifiable in regulating the temperature inside a clean room in a chip manufacturing factory.

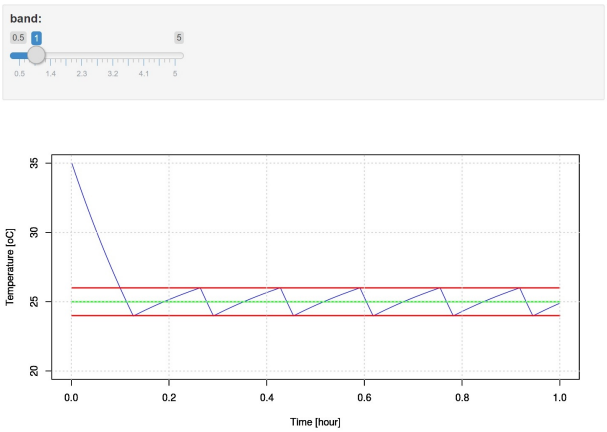


Figure 8: Example of a simple on-off controller for a temperature control setup. It is easy to see the trade-off between the performance and the distance between two consecutive change of command from on to off and vice versa.

There are also other important issue that we need to consider when using an on/off controller. In the temperature control, turning the actuator, the air conditioning unit, on and off would directly influence the temperature, the variable that we want to control. That's a good thing; it makes the implementation of the controller easy. However, in my real systems, the changes in the actuation influence another variable (or many variables in sequence) and those variables end up influencing the variable that we want to control. In those cases, we have to me more careful in selecting the band. This is because after we turn the system off, the system might be influenced by the secondary variable for a while until that effect fades.

D Remote laboratory (RLAB) manual

Installing, configuring and deploying RLAB

This section documents how to install, configure and deploy the RLAB software using RasPi 2, and the RasPi camera. The installation guide is for unix/linux/macosx users, but can also be performed on windows via the PuTTY SSH client.

Installing RLAB

Connect the Raspi to your local network using a wired LAN connection. From your router, or alternative method, locate the ip address for the Raspi, then from any computer on the same network, open up a terminal and type the following:

```
>> ssh pi@<rpi_ip_address>
% A prompt for password will pop up, type 'raspberry'
>> sudo apt-get update
>> sudo apt-get upgrade
>> sudo apt-get install unzip
>> sudo apt-get install screen
>> sudo pip install virtualenv
>> cd ~
>> wget -O rlab.zip http://<<rlab_download_location>>
>> unzip rlab.zip -d ~/rlab
>> virtualenv flaskenv
>> cd flaskenv
>> source ./bin/activate
>> pip install -r ~/rlab/requirements.txt
>> mv ~/rlab/code/ ~/flaskenv/code/
```

Configuration RLAB

Next, edit the configuration file located at

```
/home/pi/flaskenv/code/conf/application.yaml
```

An example configuration file:

```
admin:
    user: adminusername
    pass: adminpassword
    eaddr: scopelabmelb@gmail.com
    epass: emailpassword
    smtp_server: smtp.gmail.com
    smtp_port: 587

web:
    external_addr: 120.160.80.10

expr_data:
```

```
    path: ./expr_data/
    expiry_age_in_hrs: 12

apparatus_name: Test Apparatus

experiments:
    one:
        pars:
            - par1
            - par2
    two:
        pars:
            - par1
            - par2
            - par3
```

A specific description of the keys is now given.

admin:user Username for the admin. user that has access to `admin_tools` page.

admin:pass The password for the admin user.

admin:eaddr The email address that notification emails will be sent from.

admin:epass The password associated with this email address. Needed to send emails from the smtp server.

admin:smtp_server Location of SMTP server for this email.

admin:smtp_port The port for the SMTP server for this email.

web:external_addr The IP or domain name of this raspberry pi for devices outside of the LAN. Note, that a sysadmin will need to port forward port 80 HTTP from this external IP address to port 5000 of local LAN address of the raspberry pi.

expr_data:expiry_age_in_hrs The length of time a file sits on the server before it is deleted. This needs to be short since raspberry pis do not have very large storage.

apparatus_name the name of this apparatus, will be used on web pages for identification.

Finally, the last item `experiments` is the most import. In the example above, the configuration file says that this apparatus currently serves two experiments named `one` and `two`. The `one` experiment requires two parameters from the user, named `par1` and `par2`, and the experiment named `two` requires three parameters named `par1`, `par2`, and `par3`. Obviously these are just generic names for generic experiments. However, when naming experiments and parameters use one word names without spaces.

Starting RLAB Webserver on raspberry reboot

If the user would like the web server to be started at the boot-up of the raspberry pi, execute the following.

```
>> chmod +x /home/pi/flaskenv/code/boot_server.sh
>> sudo crontab -e
% Select NANO editor
```

Then insert the following line at the bottom of the file.

```
@reboot /home/pi/flaskenv/code/boot_server.sh
```

Creating experiments

Finally, in this section we address how experiments are created using the RLAB framework. The steps are as follows: (i) Add experiment to the configuration file, as shown in Subsection D, (ii) Add a HTML template that gathers the parameters for your experiment, (iii) Write the python method that performs the experiment.

Note, for the remainder of this section suppose we wish to design an experiment named `trackingrobot` that takes two numerical parameters: `p1` and `p2`.

Add experiment to Config

As shown in a previous subsection, open the file `/home/pi/flaskenv/code/conf/application.yaml` and insert:

```
experiments:
    one:
        pars:
            - par1
            - par2
    two:
        pars:
            - par1
            - par2
            - par3
    trackingrobot:
        pars:
            - p1
            - p2
```

This configuration allows three experiments to be queued, `one`, `two`, and now `trackingrobot`.

Gathering user data via experiment template

Next, create a new HTML template file called `expr_trackingrobot_form.html` under the directory

```
~/flaskenv/code/templates/expr/
```

Note, the name of this file is important, it must be of the form

```
expr_ + experiment_name_as_in_config_file + _form.html
```

Remark: This is why it is important to only use single word names for parameters and experiments in the config file as they will be used for file names.

Next, construct the HTML form using this template as an example. If the parameters are not of number-type, then select any valid HTML input type.

```
<html>
    <head>
        <title>Queue Experiment</title>
        <link rel=stylesheet type=text/css href="{{ url_for('static',
        filename='style.css') }}">
    </head>
    <body>
        <div id="container">
            <div class="title">
                <h1>Queue Experiment: Tracking Robot</h1>
            </div>
            <div id="content">
                <form method="post" action="{{ url_for('expr_submit',
                expr_id='trackingrobot') }}">
                  <label for="user">Please enter your name:</label>
                  <input type="text" name="user" /><br />
                  <label for="email">Please enter your email:</label>
                  <input type="email" name="email" /><br />
                  <label for="p1">Parameter 1:</label>
                  <input type="number" name="par1" /><br />
                  <label for="p2">Parameter 2:</label>
                  <input type="number" name="par2" /><br /><br />
                  <input type="submit" />
                </form>
            </div>
        </div>
<a href="{{ url_for('index')}}">Index</a>
    </body>
</html>
```

Remark: It is critical that the

```
action="{{ url_for('expr_submit', expr_id='trackingrobot') }}
```

line is modified to include the experiment name, and that each parameter specified in the `application.yaml` configuration file is collected in the form with the same names.

Write Python method that performs experiment

The final and most difficult step is to write the python code that performs the experiment given the parameters from the user. This code needs to be written in the file `~/flaskenv/code/experiment.py`, and needs to take the form of the following skeleton code.

```
import os, time
from picamera import PiCamera

#
#
def run_experiment(expr_id, pars, wdir):
    # Add a line here for each experiment on this apparatus
    if expr_id == 'one':
```

```
        return run_experiment_one(pars, wdir)
    elif expr_id == 'two':
        return run_experiment_two(pars, wdir)
    elif expr_id == 'trackingrobot'
        return run_experiment_trackignrobot(pars, wdir)

def run_experiment_trackingrobot(pars, wdir):
    # Parameters stored in pars dictionary, with keys
    # as specified in config file. ie.
    pone = pars['p1']
    ptwo = pars['p2']


    # Perform Initialization Procedure for
    # trackingrobot here. If the initialization fails,
    # call the following exception:
    #   raise 'Expr. Init. Failure'

    # After Initialization if no problems Perform Experiment
    # Start Video
    with PiCamera() as camera:
        camera.resolution = (640, 480)
        fname = os.path.join(wdir, 'vid.h264')
        camera.start_recording(fname)

    # Perform Work in Here, if any issue occurs call the
    # following exception:
    # raise 'Expr. Runtime Failure'


    # When experiment is finished: stop video
    camera.stop_recording()

    # Save experiment results to wdir directory. Call the files
    # anything except 'score.txt'  'setup.txt'  'vid.h264'
    results  = []
    with open(os.path.join(wdir, 'results.csv') as f:
        f.write(results)

    # If the experiment can have a score, calculate score here, otherwise
    # return 0
    score = 0
    return score

# Handle experiment 'one'
#def run_experiment_one(pars, wdir):
# Handle experiment 'two'
#def run_experiment_two(pars, wdir):
```

After everything above has been completed, the `trackingrobot` experiment can be queued remotely using the RLAB interface by hitting the webpage `/expr_queue/trackingrobot/`.