Spring 2021 ABEE2031 Control System for Built Environment

Coursework

"Control System Design and Simulation for Building Systems"

Introduction

This is an individual coursework for ABEE2031 Control System for Built Environment. It aims to use common control simulation and programming software to design, implement and analyze some simulated building systems. It will assess and enhance students' understanding on dynamic system modeling, feedback control loop, supervisory control of HVAC systems and the use of common software and programming languages in today's control industry.

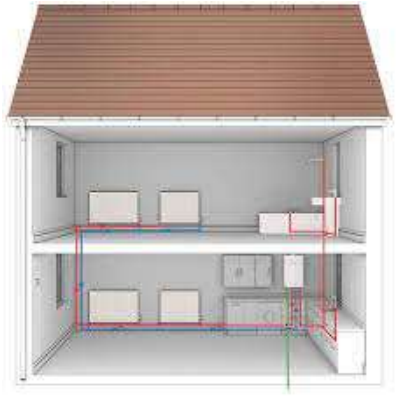
Type: individual
Weight: 30% of the total marks of this module
Release date: Apr 7th, 2021
Due time: 4:00PM May 7th, 2021 (Beijing Time)
Submission method: via Moodle

PART 1: Feedback Control Simulation (70 %)

- **Objectives**

This part aims to implement and analyze a feedback control system for a simple and ideal dynamic system via SIMULINK.

- **Methodology/Step-by-Step Guide**



Heating system is very common in residential, commercial and industrial buildings. This part will build a control system for a simple and ideal heating system in SIMULINK.

1. **Build the dynamic model of the controlled process, i.e., the heated room (30%).**

In SIMULINK, build the dynamic model of the process shown in Figure 1:

- Set the simulation stop time to be 200
- Use $T = 20, \sigma = 3, K_1 = 300, K_2 = 0.05, \tau = 5$
- Use a step signal with the final value 1 for the input Q
- Use a constant signal with the value 0 for OAT
- Use fixed-step solver with fixed-step size 0.01.
- Plot the time response of IAT, and comment on:
  i. steady state response (the IAT value when it becomes steady)
  ii. rising time (time it takes for the response to rise from 10% to 90% of the steady-state response)
  iii. Settling time (time it takes for the error e(t) = |y(t) – yfinal| between the response y(t) and the steady-state response yfinal to fall below 2% of the peak value of e(t))
- Try different values for T, K_1, τ, comment on how do the changes affect the time response of IAT?

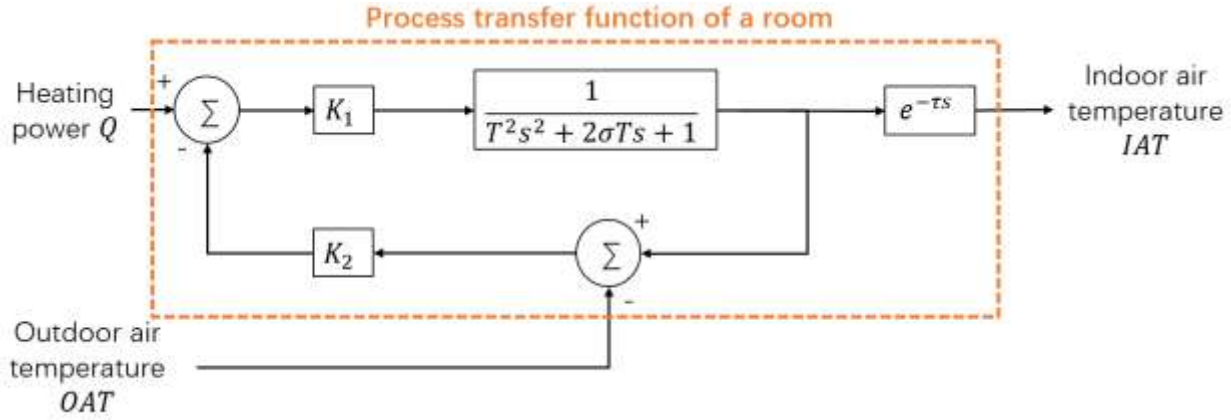


Figure 1 Dynamic model of a heated room (hint: $e^{-\tau s}$ can be represented by using the Transport Delay block in SIMULINK)

Submit your **SIMULINK model file (.slx)** and a **Word file (.docx)** with your responses to the above questions.

2. Build a feedback controller (40%)

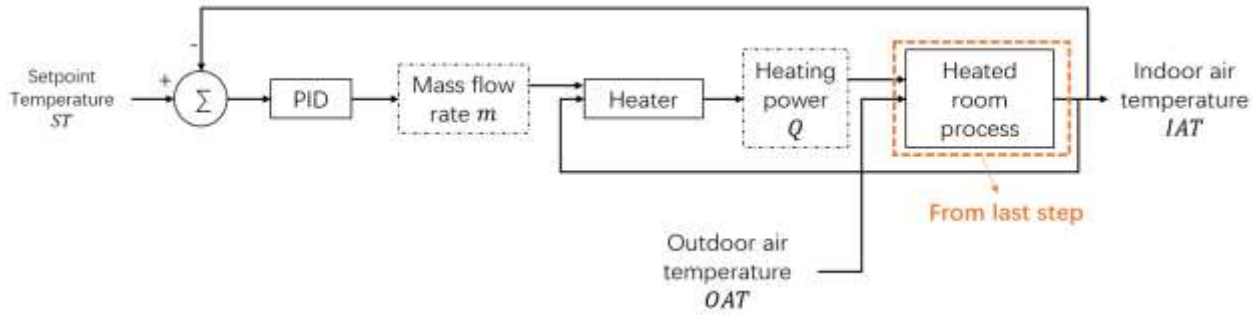


Figure 2 Block diagram of the feedback control system for the heated room (black dashed blocks represent the intermediate outputs)

- Figure 2 shows the block diagram of the feedback control system.

- The block "Heated room process" is the one you just built in the last step, with two inputs: Heating power Q & Outdoor air temperature OAT, and one output: Indoor air temperature IAT. OAT is still a constant with the value 0.
- The "Heating power Q" is generated from another process named "Heater". Heater simulates a convective electric heater. Heater has two inputs: Mass flow rate m & Indoor air temperature IAT, and one output: Heating power Q. The relationship between the inputs and output is:

$$Q = c_p m(T_{hotair} - IAT)$$

  where $c_p = 1$ and $T_{hotair} = 35$.
- The block "PID" is a PID controller. Its output is "Mass flow rate m". Initially, $K_p = 0.001, K_i = 0.001, K_d = 0$. Due to the physical limitation of the heater, the value of "Mass flow rate m" cannot be larger than 0.2, and of course, mass flow rate cannot be negative.
- Simulation end time is 1500, use fixed-step solver with fixed-step size 0.01.
- Setpoint Temperature ST is initially 20, and it changes to 22 at t=500, and then it changes back to 20 at t=1000.
- Run the simulation with above settings, plot the time response of IAT and comment on it.
- You will find the time response of IAT is not satisfactory. We need to tune K_p K_i K_d of the PID controller. A general experience-based tuning rule is shown below in Figure 3. Tune your K_p K_i K_d for a few times and try to get a better time response of IAT. **Explain** why you make such tuning actions and **show** the time response plot of IAT after each time you make the tuning.

  NOTE: No need to spend too much time on the tuning, 5-6 times are sufficient. I will not grade based on how well is your final tuning results.

  HINT:
  - Each time you tune your K_p K_i K_d, just make a very small change. A small change may have very big effects.
  - K_d can help reduce signal oscillations. But sometimes a too large K_d will make the oscillations even worse. So try to use a very small K_d in the magnitude of 0.0001.

Submit your **SIMULINK model file (.slx)** and a **Word file (.docx)** with your responses to the above questions.

- **Submission**
  1. Two SIMULINK model files
  2. One WORD file answering the above questions.

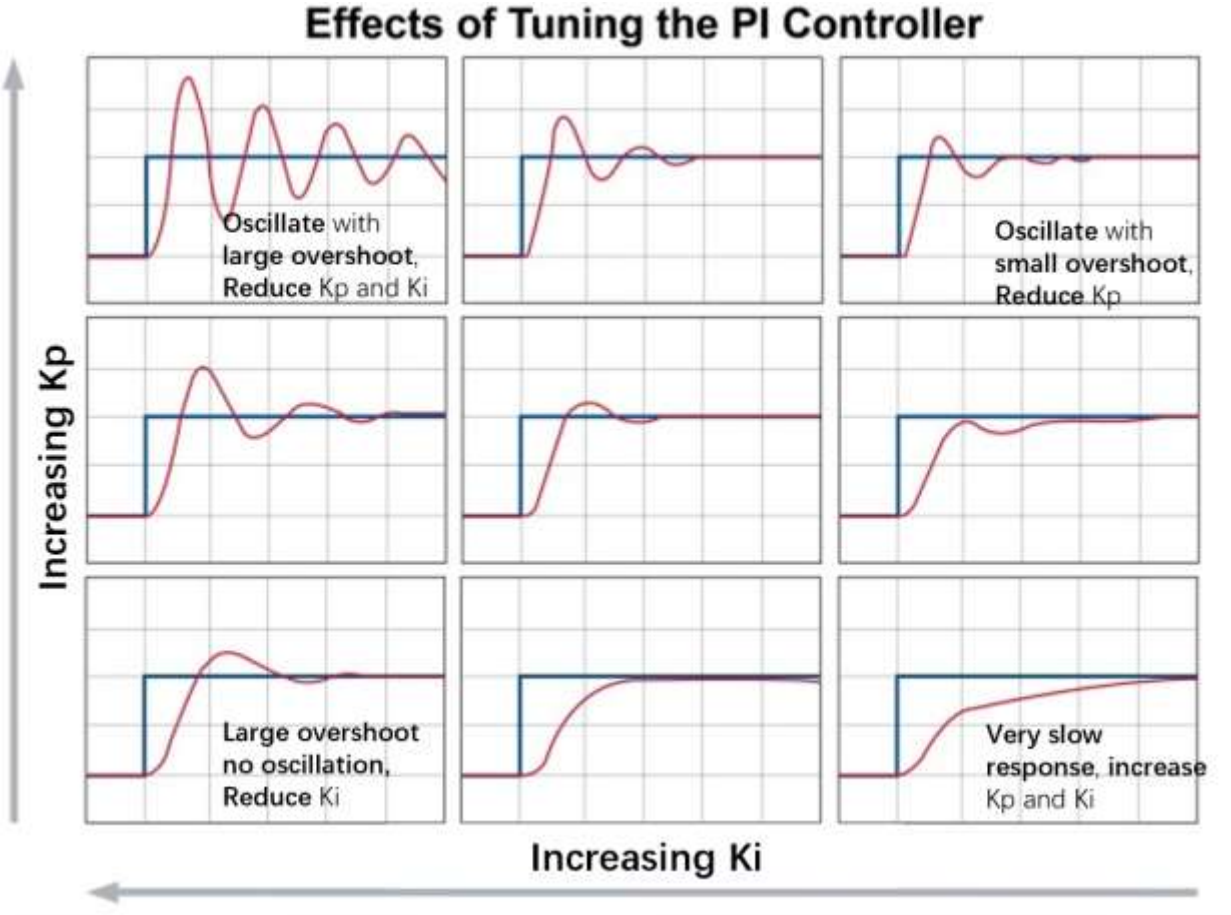


Figure 3 PID tuning rule of thumb

PART 2: Rule-based Control Simulation (30 %)

- **Objectives**

This study aims to design, implement and analyze a rule-based control strategy for a heating system based on a simulation-based building control test-bed.

- **Background**

You are given a simulation-based building control test-bed (named Gym-Eplus, source code can be found at here if you are interested to learn more: https://github.com/zhangzhizza/Gym-Eplus), where the simulator is a radiant heating system, as explained in appendix 2. You need to firstly setup the computing environment to use this control test-bed. The instructions can be found in appendix 1.

- **Methodology/Step-by-Step Guide**

1. *Problem identification (5%):* Run the baseline control strategy "iw_right_time_starter.py", then identify ONE problem of the baseline control strategy **by analyzing the baseline simulation output results**. You should clearly state your control problem and show the analysis in a Word file.

   Note:

- for students with their student ID ending with 0-3, change line 33 of iw_right_time_starter.py file to
  env = gym.make('IW-right-time-v1');
- for students with their student ID ending with 4-6, change line 33 of iw_right_time_starter.py file to
  env = gym.make('IW-right-time-v2');
- for students with their student ID ending with 7-9, change line 33 of iw_right_time_starter.py file to
  env = gym.make('IW-right-time-v3');

2. *Control logic design (5%):* Based on the identified problem, design and implement (via Python) a **simple** rule-based control logic. Explain your control logic in the Word file. Control inputs and control output(s) must be clearly shown.
3. *Evaluation (25%):* Run your control logic using the test-bed, and quantitively analyze the control simulation results in the following sectors in the Word file:
   1) Indoor thermal comfort performance compared to the baseline control (you should clearly define your thermal comfort metric(s) based on the PMV information)
   2) HVAC energy performance compared to the baseline control (you should clearly define your energy performance metric(s))
   3) Limitations of your control strategy

NOTE: You are not graded based on the "correctness" or complexity of your results. You are graded based on your analysis, i.e., do you have a reasonable explanation to support your design/conclusion. Therefore, DO NOT spend too much time on the control logic design. As long as you give reasonable explanations, you can even just change a few parameters of the given baseline control strategy to make your control logic.

- **Submission**
  1. Python code (.py file) of your control strategy
  2. Word file with your analysis

APPENDIX 1: The use of Gym-Eplus

Gym-Eplus is a home-brewed open-sourced Python-based software that wraps the EnergyPlus-v-8-3 into the OpenAI Gym environment interface. EnergyPlus is one of the most popular building energy simulation engine. It can simulate the energy and thermal performance of various types of buildings and HVAC systems. OpenAI Gym is an open-sourced simulation environment for control simulation. Gym-Eplus combines the two: it uses EnergyPlus as the core simulation engine to achieve control simulation of building HVAC systems.

➢ **Computing environment setup**

Gym-Eplus is written in Python and is developed in Linux OS. There are two ways that you can setup your computing environment. You can choose either of the following two ways to setup yours.

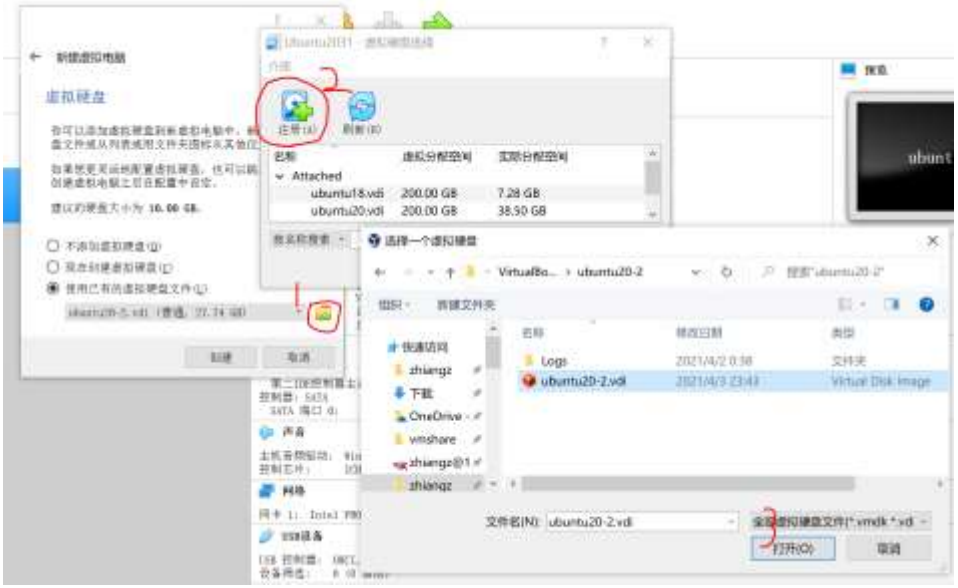
1. Via a pre-built virtual machine image.

1) Download the pre-built image ubuntu20-2.vdi file from this link: https://nottinghamedu1-my.sharepoint.com/:f:/g/personal/z2020054_nottingham_edu_cn/EnTzuSorvxpElgUDr7h2DHIBu6VroGh0jvrKzz4tYZJSfA?e=Ygv5J6 (about 10 GB large).
2) Download and install Oracle VirtualBox in your computer. Download at https://www.virtualbox.org/wiki/Downloads. Windows users select "Windows host", Mac users select "OS X hosts" (NOTE: Mac computers with M1 chip cannot use this software).
3) Run Oracle VirtualBox. Windows users, right click the VirtualBox icon and "Open as Administrator"; Mac users run the "terminal" program, and type "sudo virtualbox".
4) Click "New" button as shown in the image below (Mac users may see a slightly different interface) to create a new virtual machine.

5) Name your new virtual machine as "Ubuntu2031", click next.

6) Select the memory size of this virtual machine, better to select drag the bar to the end of the green limit, and then click “Next”.

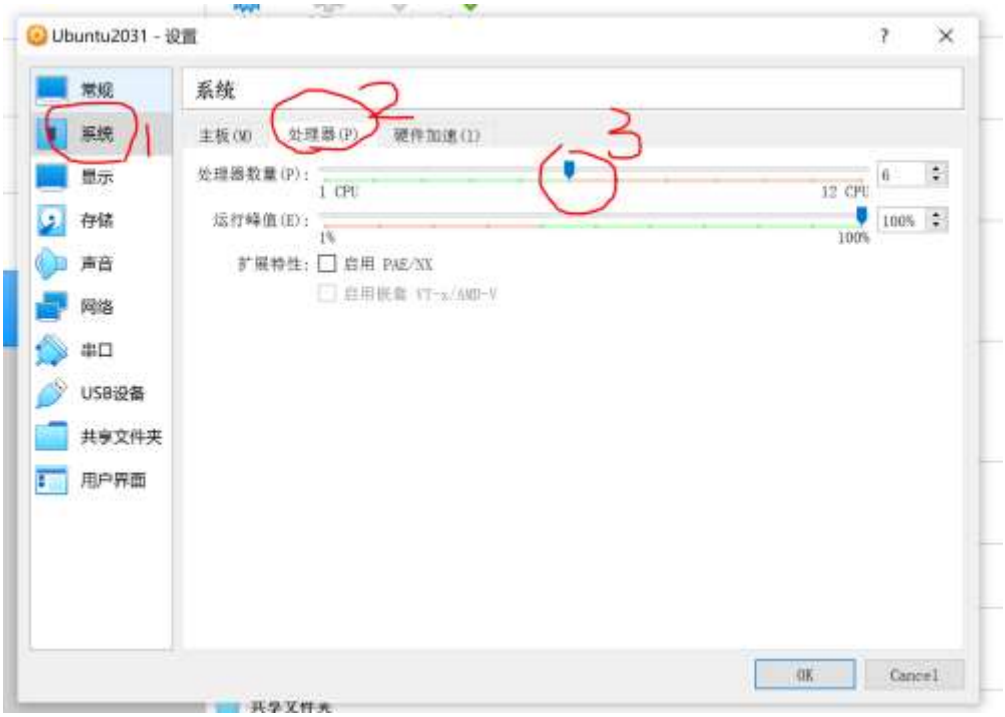
7) Import the VDI image file that you downloaded before.



8) Click “Create” button. You will then see a new virtual machine appears on the left pane.

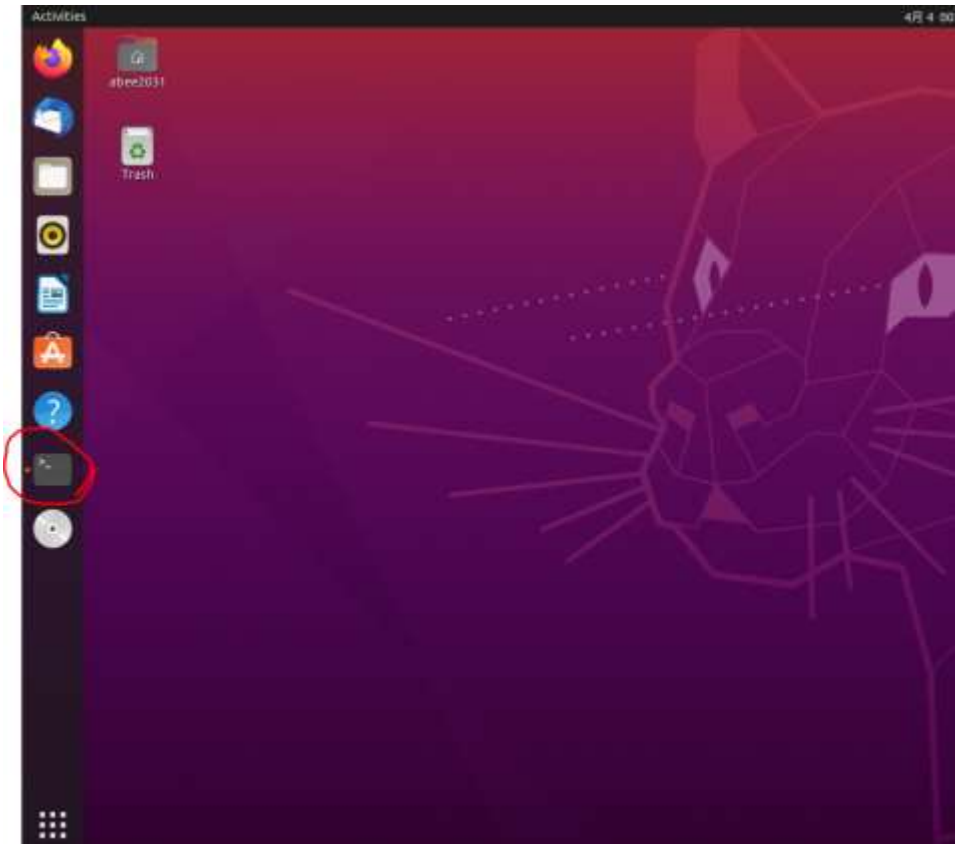
9) Click the new virtual machine “Ubuntu2031”, and then click “Setting” button.

10) Click “System”, and then go to “CPU” tab, and drag the number of CPUs bar to the end of the green limit. Click “OK”.



11) Click “Start” to start this virtual machine. If you are asked for a password, type 2031.

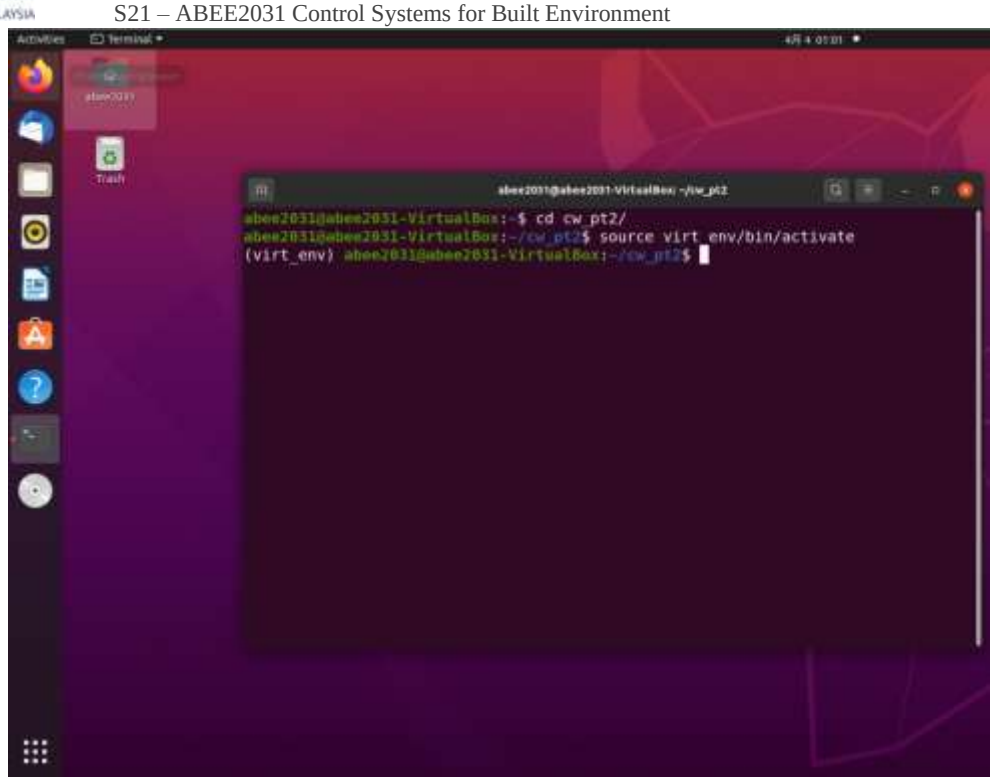
12) Once the virtual machine starts, click “Terminal” on the desktop.



13) On the terminal, type the following commands line by line with hitting “Enter” after each line.

```
cd cw_pt2
source virt_env/bin/activate
```

14) You will see something like below. The environment is all set for run now!



2. Setting up the environment from scratch (you must have an Ubuntu 20.04 or Mac OS system).
   1) Download and install Java on your computer. For Ubuntu 20.04, type "sudo apt install default-jre" in a terminal. For Mac, follow this article https://www.java.com/en/download/help/mac_install.html
   2) Download and unzip the starter package to your computer from Moodle.
   3) Open a terminal and "cd" to the directory of the starter package.
   4) Run the following commands in your terminal window line by line:
      python3 -m venv virt_env
      source virt_env/bin/activate
      pip install -r requirements.txt
   5) Download and install EnergyPlus-8-3-0.
      For Ubuntu 20.04: Download the installer at https://github.com/NREL/EnergyPlus/releases/download/v8.3.0/EnergyPlus-8.3.0-6d97d074ea-Linux-x86_64.sh, and run this installer in terminal. When asked for the installation location, just type "." for the current directory. After installation finished, copy the EnergyPlus-8-3-0 folder to "{the_starter_package}/eplus-env/eplus_env/" directory.

      For Mac: Download the installer at https://github.com/NREL/EnergyPlus/releases/download/v8.3.0/EnergyPlus-8.3.0-6d97d074ea-Darwin-x86_64.dmg, and run this installer to install. After installation finished, copy the EnergyPlus folder to "{the_starter_package}/eplus-env/eplus_env/" directory.

- **Usage**

Gym-Eplus is a Python package to realize interactions between some Python logics and EnergyPlus simulation, as shown in Figure 4.

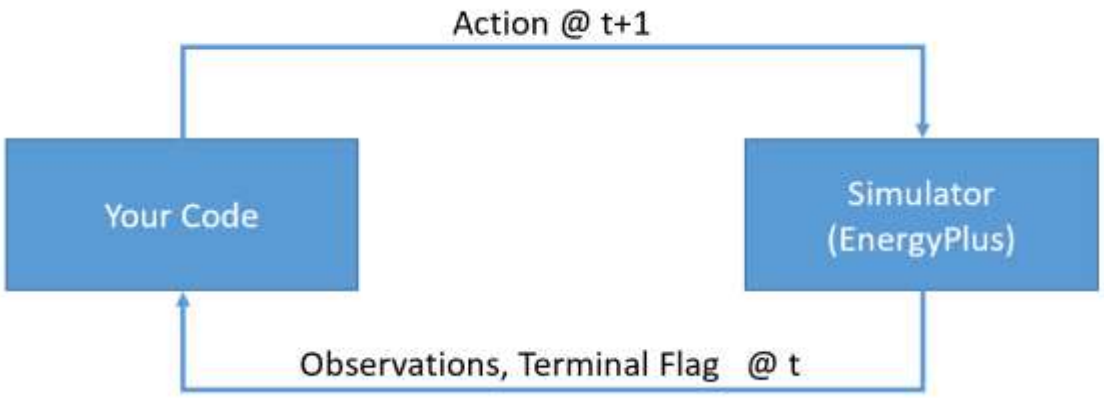


Figure 4 Gym-Eplus realizes Python-EnergyPlus interactions

Now I will use "iw_right_time_starter.py" file (you can find it in the starter package) to explain the usage of Gym-Eplus. Before using Gym-Eplus, you have to import gym and eplus_env. Gym-Eplus has three major functions:

1. env = gym.make("an environment name string")

```
import gym, eplus_env, logging
 Thunderbird Mail
logger = logging.getLogger('Ctrl-Tester');
logger.setLevel(logging.DEBUG);
ch = logging.StreamHandler()
ch.setLevel(logging.DEBUG);
logger.addHandler(ch);
logger.info('Running the simulation test...')
logger.info('Environment warm-up may take time.')
env = gym.make('IW-right-time-v3');
ob, is_terminal = env.reset();
logger.info('Environment warm-up is done.')
total_steps = 8928;
step_i = 0;
last_perct = 0;
```

   This function creates a new simulation environment. The input argument is a string, representing the environment name. In this coursework, you will be given the environment name.

2. ob, is_terminal = env.reset()

```
import gym, eplus_env, logging
 Thunderbird Mail
logger = logging.getLogger('Ctrl-Tester');
logger.setLevel(logging.DEBUG);
ch = logging.StreamHandler()
ch.setLevel(logging.DEBUG);
logger.addHandler(ch);
logger.info('Running the simulation test...')
logger.info('Environment warm-up may take time.')
env = gym.make('IW-right-time-v3');
ob, is_terminal = env.reset();
logger.info('Environment warm-up is done.')
total_steps = 8928;
step_i = 0;
last_perct = 0;
```

Reset the simulation. This function does not have any input arguments, and it gives two outputs: ob is a python list of float representing state observations; is_terminal is a Boolean value representing whether or not the simulation is terminal. The physical meanings of each item in ob will be explained in Appendix 2.

3. ob, is_terminal = env.step([some_act])

```
while is_terminal == False:
# !!DO NOT change the above lines
#########################################################
# The following lines are the baseline control strategy
# You should implement your control strategy here
        weekday = ob[2];
        hour = ob[0];
        # For all the weekends
        if weekday >= 5:
        # Step down the setpoint
                act = 1;
        # For all the weekdays
        else:
        # Before 7:00 AM or after 8:00 PM, step down the setpoint
                if hour < 7 or hour >= 20:
                        act = 1;
        # For other time, step up the setpoint
                else:
                        act = 0;
        #logger.info('This observation is: %s'%ob);
        ob, is_terminal = env.step([act])
# Control strategy ends
#########################################################
```

This function sends a control action back to the simulator and get the resulting state observation of this time step. The input argument must be a python list, and the physical meaning of the action is environment-specific. For this coursework, the physical meaning of the action will be explained in Appendix 2. The outputs of this function are still ob and is_terminal, which is the same as the reset function.

Actually, you do **NOT** need to use the first two functions because I have already coded it for you. You only need to use the step function, i.e., determine your control action and send it back to the simulator.

After you run the starter python script, you will see a new directory created in the package root directory named “Eplus-env-IW-right-time-v{x}-res{x}”. This is the directory where your simulation results are

stored. The simulation results file is "Eplus-env-IW-right-time-v{x}-res{x}/Eplus-env-sub_run1/output/eplusout.csv". You should analyze the results in this CSV file to finish the coursework.

APPENDIX 2: IW-right-time simulation environment

"The right time" (Intelligent Workplace optimal start/stop problem)

The Intelligent Workplace (IW) is a green and smart building located in Pittsburgh, PA, USA. It is the home of Center for Building Performance and Diagnostics of Carnegie Mellon University. It has a unique radiant heating system, which has very slow thermal response (e.g., it may take hours to heat up the space). The IW occupants are complaining that IW is too cold in some winter mornings. In addition, IW is too warm in some winter evenings, when no one occupies the space.


Control action:

You should design a control strategy for "step-on/step-down" of the indoor average air temperature (IAT) setpoint. When "step-on", the IAT setpoint is 24 ^{o}C, when "stepdown", the IAT setpoint is 18^{o}C.

In your Python implementation, step-on is integer 1, stepdown is integer 0. That means, when you use the step function of Gym-Eplus, you should use "ob, is_terminal = env.step([act])" where act is either 1 or 0 depending on your control logic.

Control input (observations):

You are provided with the following state observations (you do **NOT** need to use all of them to design your control logic):

- Index 0: hour of the day (0-23)
- Index 1: minute of the hour (0-59)
- Index 2: day of the week (0-6, 0 is Monday)
- Index 3: current outdoor air temperature (^{o}C)
- Index 4: current outdoor air RH (%)
- Index 5: current diffuse solar radiation (W/m^2)
- Index 6: current direct solar radiation (W/m^2)
- Index 7: current IAT setpoint (^{o}C)
- Index 7: current IAT (^{o}C)
- Index 9: current average PMV (if the space is not occupied, then return -999)

- Index 10: predicted occupant first-arrival time (1.0 hour resolution) of the day (e.g., 8.0 is 8:00 AM); it is updated at every 0:05 AM; if IW won't be occupied for the day, -999 is returned
- Index 11: predicted occupant last leave time (1.0 hour resolution) of the day (e.g. 17.0 is 5:00 PM); it is updated at every 0:05 AM; if IW is not occupied or for the day, -999 is returned
- Index 12: IW heating demand (kW)
- Index 13-36: hourly outdoor air temperature forecast of the next 24 hour (provided at every 0:00 midnight, for other timesteps, -999 is returned)

Control timestep:

5 min.

Control simulation period:

Jan 1st – Jan 31st. (winter)