Code documentation

**Overview**

This testbed simulates a multizone building, ideally, with multiple stories and within each story there are 5 thermal zones. This testbed could simulate multi-unit residential buildings or commercial buildings, for example, hotels. The purpose of the first step of simulation is to obtain a realistic electricity consumption due to plug-in loads, lighting, cooling/heating equipment, etc. Since at this stage we only care about controlling electric load from cooling/heating, we intentionally split electricity consumption into 2 aspects, HVAC related, and non-HVAC related. This simulation testbed would only work on HVAC related loads. For non-HVAC related loads, we use Cornell dorm electricity data as a complement. Note that we intentionally select Cornell dorm electricity data because they have chilled water and steam plants to provide district cooling and heating to most of the buildings on campus, which means that the electricity consumption is independent of the outside weather condition.

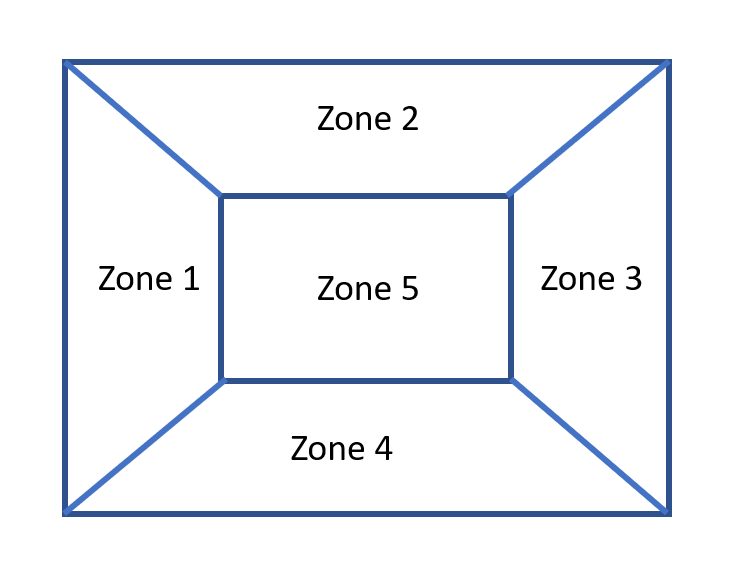
**Simulation period**: June – Aug (summer-cooling season)

**Location**: The testbed is hypothetically located in Ithaca, NY.

**Weather data**: Elmira, NY TMY file, the closest station to Ithaca (AMY file available)

**Cornell electric data:** released in ASHRAE Great Energy Predictor Competition 2019; Building name: Cockatoo-lodging-Homer; building total floor area: 12046.2 sqm; year built: 1941

**Floorplan drafted**: (dimension to be determined, drawing not to scale)



Four peripheral zones and one interior zone. Each zone has a dominant orientation. Depending on the orientation, solar irradiance is different for each zone.

TODO: use ladybug to simulate the actual solar radiation that each room would receive from TMY file. ~~Other option: using sinusoidal functions but not realistic.~~

Code

1. **Zone.py**

This file defines each zone within a building as an object that has its own features and methods.

Features:

**ID**: zone number  
neighbors: zone objects that have heat transfer with the current zone object  
**neighborlist**: list of zone IDs that have heat transfer with the current zone object  
**R**: thermal resistance ~5C/kW  
**C**: thermal capacitance ~2-3kWh/C  
**Pmax**: max heating/cooling system capacity ~2tons  
**COP**: system efficiency ~3  
**Qall\_schedule**: predefined schedule of exogenous heat flow, as a Series  
Qall: current exogenous heat flow (lighting, occupant, solar radiation, misc)  
**Tin**: current room temperature  
**orient**: orientation of the zone (window facing south etc); 1: south; 2: west; 3: east; 4: north  
P: electric load  
**condition**: +1 for heating and -1 for cooling  
**Tset\_schedule**: predefined setpoint temp schedule, as a Series  
Tset: current setpoint temp

The model is initialized with the features in bold.

TODO: COP could be varying according to outdoor temp and other parameters

Methods:

* **get\_ID**: returns the ID of a zone
* **add\_neighbor** and **add\_neighbors**: given a list of zone objects, adds neighbors to each other’s lists
* **get\_neighbors:** returns the neighbors of a zone
* **get\_current\_Tin:** returns current room temp
* **update\_Tin:** updates the current room temp with setpoint temp

Assume cooling system will meet the cooling demand and hit the desired setpoint accurately.

* **update\_Tset:** updates the current/next timestep setpoint temp from Test\_schedule
* **update\_Qall:** updates the current/next timestep exogenous Q temp from Qall\_schedule

1. Building.py

This file defines a system/building as an object that has its own features and methods.

Features:

zonelist: records all zone IDs in this building/system  
syssize: length of the zonelist, determines the size of matrix operation  
Tout: outdoor temperature (drybulb temp)  
htgload: building heating load  
clgload: building cooling load

all\_zone\_Tin: records room temps for all zones  
all\_zone\_P: records electric load for all zones  
all\_zone\_R: records thermal resistance for all zones  
all\_zone\_C: records thermal capacitance for all zones  
all\_zone\_COP: records COP values for all HVACs  
all\_zone\_Qall: records current exgoneous Q for all zones   
all\_zone\_Tset: records current setpoint temp for all zones

Methods:

* update\_Tout: updates real-time outdoor temp
* update\_zonelist: collect all zone objects into one list, update system size (number of zones in the system); gather zone parameters (Tin, P, R, C, COP, etc)
* generate\_all\_zone\_params: given a zone parameter of interest (Tin, P, R, C, COP, etc), record its corresponding values in system features (all\_zone\_Tin, all\_zone\_P, all\_zone\_R, all\_zone\_C, all\_zone\_Qall, all\_zone\_Tset)
* generate\_all\_zone\_COP: collect all zone COPs, + for heating and – for cooling

Notation used in continuous-time system dynamics equation

Tdot = AT + BP + w

T: zone indoor temp, P: power

* generate\_Rij and init\_A: generate A matrix with wall insulation parameter, size of the wall, and adjacency matrix for all zones

(self, wallR=70): # Assume R-13 wall with an effective area of 30m2

* Rij = np.zeros((self.syssize, self.syssize))
* for zone in self.zonelist:
* # define diagnal as 1/Ri
* Rij[zone.ID - 1, zone.ID - 1] = 1. / zone.R
* # fill in the adjacency matrix from neighbor info
* for zone\_j in zone.neighbors:
* Rij[zone.ID - 1, zone\_j.ID - 1] = 1. / wallR
* for i in range(self.syssize):
* Rij[i, i] = -Rij[i, :].sum()
* return Rij
* def init\_A(self):
* C = self.get\_all\_zone\_C()
* C = np.broadcast\_to(C, (self.syssize, self.syssize)).T
* A = np.multiply(1./C, self.generate\_Rij())
* self.A = A
* def get\_B(self):
* COP = self.get\_all\_zone\_COP()
* C = self.get\_all\_zone\_C()
* bi = np.squeeze(np.divide(COP, C))
* B = np.diag(bi)
* return B
* def get\_W(self):
* theta = self.Tout \* np.ones((self.syssize, 1))
* R = self.get\_all\_zone\_R()
* Q = self.get\_all\_zone\_Qall()
* C = self.get\_all\_zone\_C()
* W = np.divide(theta, np.multiply(C, R)) + np.divide(Q, C)
* return W
* def discrete\_systemT\_update(self, deltaT):
* A = self.get\_A()
* B = self.get\_B()
* W = self.get\_W()
* Ad = linalg.expm(np.multiply(deltaT, A))
* Bd = np.dot(np.linalg.inv(A), np.dot((Ad - np.identity(self.syssize)), B))
* Wd = np.dot(np.linalg.inv(A), np.dot((Ad - np.identity(self.syssize)), W))
* Tk = self.get\_all\_zone\_Tin()
* U = self.get\_all\_zone\_P()
* Tk1 = np.dot(Ad, Tk) + np.dot(Bd, U) + Wd
* return Tk1
* def load\_calculation(self, deltaT):
* A = self.get\_A()
* B = self.get\_B()
* W = self.get\_W()
* Ad = linalg.expm(np.multiply(deltaT, A))
* Bd = np.dot(np.linalg.inv(A), np.dot((Ad - np.identity(self.syssize)), B))
* Wd = np.dot(np.linalg.inv(A), np.dot((Ad - np.identity(self.syssize)), W))
* Tk = self.get\_all\_zone\_Tin()
* Tset = self.get\_all\_zone\_Tset()
* P = np.dot(np.linalg.inv(Bd), (Tset - np.dot(Ad, Tk) - Wd))
* return P