Laboratoire d'automatique, EPFL

OpenBuild: Instructions

Tomasz Gorecki, Faran Qureshi

April 21, 2015

1 Introduction

2 Installation

This section summarizes software that need to be installed on your machine to be able to use OpenBuild.

- OpenBuild 2.0 has been tested on Mac OS X 10.8 Mountain Lion, OS X 10.9 Mavericks, OS X 10.10 Yosemite, and Windows 7 32 bits and 64 bits.
- OpenBuild works in close integration with the building simulation software EnergyPlus. EnergyPlus is an open-source program available on every platform. OpenBuild 2.0 has been tested with EnergyPlus v.8.0. We believe the toolbox should work with more recent versions of EnergyPlus but tests are required. Due to syntax changes in EnergyPlus, Openbuild 2.0 does not work with EnergyPlus 7.2 and older. It should work with more recent versions of EnergyPlus. We recommend the use of EnergyPlus 8.0 since examples are provided for this version. Visit `http://apps1.eere.energy.gov/buildings/energyplus/` to install EnergyPlus and find more information on EnergyPlus.
- Ruby : using OpenBuild requires a Ruby interpreter. It exists by default on recent Mac OS X installations. Ruby scripts of Openbuild are not compatible with Ruby 2.0 and subsequent. In OS X 10.9, the system version of Ruby has been changed from 1.8 to 2.0 so the user needs to provide the path to the binaries of Ruby 1.8 manually. In OS X

10.9, ruby 1.8 is still available as part of the system install. From OS X10.10 onwards, it is not the case anymore, and it needs to be installed. Openbuild 2.0 has been tested with Ruby 1.8.7. We recommend installing. Visit `https://www.ruby-lang.org/` for information and downloads.

- Java : Co-simulation requires Java. Openbuild 2.0 has been tested with Java JRE 7 but should work with different versions of Java. You have to make sure on Mac that Java is the Mac supported version of the JRE. For OS X 10.10 users, refer to `https://support.apple.com/kb/DL1572?viewlocale=en_US&locale=en_US` in case of malfunction of the default Oracle JRE.

- To be able to read SQL files produced by EnergyPlus with Matlab, the toolbox *mksqlite* has been used. Visit `http://sourceforge.net/projects/mksqlite/` and follow installation instructions there. Precompiled MEX files are provided with recent installs of *mksqlite* but you might need to recompile it for your platform. Openbuild 2.0 has been tested with mksqlite 1.14 and may not be compatible with too old versions of mksqlite due to inconsistent syntax.

- The file `installOpenBuild.m` in the root of the project needs to be run once. Open this script and give the path to the EnergyPlus, Ruby and Java executables. Some default are indicated depending on the OS. If you do not save the path after executing this file, then you will need to run it everytime you want to use OpenBuild.

3 Tutorial for MPC design

As an introduction to the cosimulation environment, a tutorial example is proposed to guide the user through the main steps of creating a state-space model for a building, and run cosimulations for a particular building. All the material necessary for the tutorial is found in the subfolder `Tutorial` in the `examples`. After installation, you should readily able to run the `tutorial.m` file.

The first section of the tutorial introduces the process of creating a model for a building. All information about a building is regrouped in the class `building`, which has two attributes, an instance of the class `buildingData` and one of the class `buildingModel`.

The first step is to create an empty `building` object.

```
b = building('tutorial');
```

Next we will need to import all data required to create the model. This includes one EnergyPlus `.idf` file and one weather file `.epw`. At this point, some things have to be noticed.

- One of the first step of the data gathering is to do one run of EnergyPlus with the files provided to collect the data processed by EnergyPlus. Therefore, the `.idf` file has to work in the sense that it can be run in EnergyPlus without modification. You can verify this by running the commands `runenergyplus filename.idf weather.epw`.

- For the purpose of modeling, the choice of the weather file is not crucial but will have an influence since data from simulation is used to produce the model (especially figure out which convection coefficients should be used). Therefore it is advised to use the same weather file for the modeling and the simulation itself.
- the weather file has to be in the weatherfile folder of the EnergyPlus installation.

```
idfFile = [pwd '/OfficeBuilding.idf'];
epwFile = 'USA_IL_Chicago-OHare.Intl.AP.725300_TMY3.epw';
b.loadData(idfFile,epwFile);
```

Next, the command createModel, will construct the model of the building. For details of all the steps, please refer to the comments in the code. Here is a rough list of the steps taken to create the model:

- Modify the `.idf` file to add a range of outputs required in the sequel. In particular, the creation of some particular reports and the `.sql` simulation report are required.
- Run the `.idf` file once.
- Collect the .sql and .eio report files from the simulation and import the all the data required there into MATLAB. This includes thermal characteristics of the material, zones, surfaces, constructions, simulation data (timestep, duration,...), and data about internal gains specified in the `.idf`
- This step consists in describing the building as a set of thermal nodes with thermal capacities, connected to some thermal resistances. Each layer of each surface will form one node and each zone also is described by one node. Outside is also represented by a couple nodes. (Sky, outside air, ground)
  - identify all "links", *i.e.*which zones are connected to which zones through which surfaces
  - Starting from the set of nodes representing the zones, successively add the nodes representing each link and accordingly augment the connectivity matrix of the graph.
  - Connect the surfaces which interact through longwave radiation by appropriately modifying the connectivity matrix of the graph.
- Identify the effect of all disturbances on the nodes (internal gains and solar gains)
- At this point it is easy to derive the differential equations describing the building from the connectivity matrix. The only difficulty is that some of the nodes, namely the windows, have no thermal capacities and therefore have to be eliminated. A simple substitution procedure is applied to do so.

All of these steps are performed when the following command is called:

```
b.createModel;
```

Now the building model is almost ready. To allow flexible HVAC modeling in MATLAB and overcome the limitations of the EnergyPlus external interface. For this purpose, custom objects are added to the idf file, which allow to control gains in the rooms in a flexible way.

```
b.addCustomInputs;
```

This commands modifies the `.idf` file to add one schedulable gain object in each room of the building and adds the external interface object allowing to control this object in Matlab. In addition, it modifies the model of the building to add these new inputs.

```
ssM = b.buildingModel.thermalModel.getStateSpaceModel();
```

Finally, this command creates the continuous-time state-space model of the building. This model takes the form

$$\begin{aligned} \dot{x} &= Ax + B_u u + B_d d \\ y &= Cx \end{aligned} \tag{1}$$

where x is the state of the building which represents temperatures at the different nodes of the building. u is the control input to the building. In this case it is the desired gain in each room in watts. d is the vector of disturbance including outside temperature, sky temperature, solar gains on inside and outside surfaces, and internal gains due to occupants, lights, equipment. The matrices can be found under the fields `ssM.A`, `ssM.Bu`,`ssM.Bd`,`ssM.C`.

Next, the following part of the tutorial looks at simulating the building in cosimulation with EnergyPlus. The process of setting up a simulation and running up is dealt with in the `simulationEngine` object. It simply takes as argument the `building` object of the building to simulate.

```
sE = simulationEngine(b);
```

Inside the `simulationEngine`, each of the component of the simulation loop will be simulated and communication between these components will be handled. Each component is encapsulated in a `simulator` object and respects the same scheme: it has two main attributes: one is a structure `IO` recapitulating the different group of inputs and outputs that the simulator takes and gives out. The second is a structure `data` which contains any data required during the simulation for this component. In addition, it takes 3 handles of functions as attributes: the first is the initialization of the simulator, which will create the IO and `data` fields; the second iterates the simulator and will be called at each timestep to produce the next outputs. The third one is optional and is called at the end of the simulation to complete task such as communication closing,... The central piece of the simulation is of course the building itself.

```
sE.addSimulatorEnergyPlus;
```

This function sets up the communication between MATLAB and EnergyPlus based on scripts derived from the MLE+ toolbox. it uses directly all the data stored in the `building` object to start the simulator.

Then, a second simulator object takes care of the controller.

```
sE.addControllerPI;
```

In this simple case, it sets up a separate PI controller in each room. The parameters of the controller are manually provided in the initialization function. The iteration function reads the temperature in each room and compares it with the setpoint to form the error and compute the next control action.

```
sE.runSimulation;
```

This last function finally takes care of running the simulation and the communication between the different components. finally, it also saves the data at the end of the simulation.

Important Notice: In the current state, the toolbox does not recognize the presence of existing HVAC system in the `.idf` file, which means that if there exist any, they will be run in parallel to external control, without the model being able to predict their behaviour. Therefore, to have predictions conform with the actual building behaviour, the heating system needs to be removed from the `.idf` file. This has to be done manually and has been done in all files in the examples. Future developments will allow to perform this task automatically by conserving only EnergyPlus objects supported by OpenBuild. Nevertheless, it is not required to do this if OpenBuild is only used to get a model of the building.

At this point, the simplest possible simulation has been performed. Let us now go through a more sophisticated simulation with an MPC controller, which actually uses the model of the building. The following formulation is used for the MPC. Still assuming that we can directly control the heat fluxes to each room our aim is to minimize the total energy use. Therefore, the cost function of our controller will be

$$J(\mathbf{x}, \mathbf{u}) = \sum_{t=0}^{N-1} \|u_t\|_1 \tag{2}$$

with N the horizon length of our controller and $\mathbf{x}$ and $\mathbf{u}$ the predicted state and input trajectories.

The MPC problem comes out as:

$$\begin{aligned} &\text{minimize } J(\mathbf{x}, \mathbf{u}) \\ &\text{subject to} \\ &\forall t \in [1, N] \\ &\qquad x_t = Ax_{t-1} + B_u u_{t-1} + B_d d_{t-1} \\ &\qquad y_t = Cx_t \\ &\qquad u_t \in \mathcal{U} \\ &\qquad y_t \in \mathcal{X}_t \end{aligned} \tag{3}$$

where $\mathcal{U}$ the input constraints (in this case box constraints), $\mathcal{X}_t$ the time-varying constraints on the temperature in the buildings (we constraint the temperature more tightly during the day). To create this second simulation, the procedure is very similar.

```
sE = simulationEngine(b);
sE.addSimulatorEnergyPlus;
```

Next we need to prepare the state-space model we want to feed the controller. We already have a continuous-time state-space model `ssM`. We have to discretize it. For simplicity, we will run the controller at the same sampling rate as the simulation is run.

```
%discretize the state-space model
ssMd = discretizeModel(ssM,b.buildingData.simulationData.timestep);
```

This command discretizes the system to the appropriate timestep.
An MPC controller requires full-state information, which in practice is often not available. This is the case in EnergyPlus. Indeed, EnergyPlus can output the temperature of the zones, and the temperatures of the surfaces, but not the internal nodes temperatures. Moerover, due to modeling differences, the surfaces temperatures in EnergyPlus and in the state-space model can be slightly different. Therefore, most of the time, we only consider that the zone temperature are available, which also correspond to the most realistic case. The state-space model extracted from the building does just assume full-state measurement (*i.e.*`ssM.C` is the identity matrix by default). Therefore, we have to specify which measurements are available to our controller to set up the estimator accordingly.

```
measurement =
    regexpcmp(ssMd.outputLabels.getLabel(1:ssMd.outputLabels.length),'Zone
    Mean Air Temperature','ignorecase');
ssMd.C = ssMd.C(find(measurement),:);
ssMd.outputLabels.removeLabels(find(~measurement));
```

These command manually specify we want to consider only the room temperature as available measurements.

Next, in most MPC implementation, in order to correct for disturbance prediction errors, as well as modeling mismatches, we consider offset-free formulation, which virtually will add integrating behaviour to the estimator. The disturbance model has to be specified. We generally considered output disturbances:

```
% Augment the model with output disturbance model
nMeas = sum(measurement);
ssMd.A = blkdiag(ssMd.A,eye(nMeas));
ssMd.Bu = [ssMd.Bu; zeros(nMeas,size(ssMd.Bu,2))];
ssMd.Bd = [ssMd.Bd; zeros(nMeas,size(ssMd.Bd,2))];
ssMd.C = [ssMd.C eye(nMeas)];
outputs = strcat('Disturbance on
    ',ssMd.output.labels.getLabel(1:ssMd.output.labels.length));
```

```
ssMd.state.labels.addLabels(outputs);
ssMd.state.units.addLabels(ssMd.output.units.getLabel(1:ssMd.output.labels.length));
```

This few lines augment the model of the building with constant output disturbances. We can now set up the estimator:

```
sE.addEstimator(ssMd);
```

This sets up the estimator based on the augmented model: it will take as inputs the measurement, previous values of inputs and disturbances and output an estimate of the state of the building and the output disturbance.
Finally, we specify the MPC controller with this model

```
sE.addControllerMPC(ssMd);
```

The details of the MPC implementation can be found in the corresponding files (`MPC_setback_data.m` and `MPC_setback_update.m`). The augmentation of the model does not result in any modification in (3) except that now , we have that $y_t = Cx_t + f$ where f is the current estimate of the output error. It is implemented with soft constraints so that feasibility is always guaranteed. In the examples provided here, the MPC problem have been formulated with the toolbox YALMIP and the solver CPLEX has been used to solve the problems. Because we use the beta functionality decribed in `http://users.isy.liu.se/johanl/yalmip/pmwiki.php?n=Blog.Beta-version-of-a-more-general-optimizer`, the user needs to specify a solver in any case. Unfortunately, for this problem the quadprog solver from energyPlus fails to solve the problem...

```
sE.runSimulation;
```

This finally runs the simulation and saves the data.

Next,we display some of the result of the simulation. When a simulation is run, results are saved automatically. By default, data is saved in a subfolder of the project folder, named `simulation_saves`. The path of the saved data is temporarily in the simulationEngine object under the field `parameters.savePath`.

```
s = load(sE2.parameters.savePath);
saveData = s.saveData;
```

These lines load the data of the MPC simulation. The data is saved in a structure which has 2 fields. The field `labels` contains the list of all labels of the data saved. Since EnergyPlus mainly identifies the data through string labels, we use the same system and attach to each signal in the simulation a unique identifier. If the data comes from EnergyPlus, then the label is conserved in OpenBuild. The field `data` contains the data itself in an array.

```
% Find the temperatures in the rooms based on the labels
zoneTempId =
    regexpcmp(saveData.labels.getLabel(1:saveData.labels.length),'Zone Mean
```

```
    Air Temperature','ignorecase') &
    ~regexpcmp(saveData.labels.getLabel(1:saveData.labels.length),'Estimate','ignorecase');
zoneTempId = find(zoneTempId);
runTime = size(saveData.data,2);
zoneTemp = saveData.data(zoneTempId,:);
```

The previous commands gathers the temperature in the zones by looking into the label identifiers.

```
% constraints on temperature used in the MPC scheme
constraintsMin = sE2.controller.data.Tmin -
    sE2.controller.data.setback(1:runTime);
constraintsMax = sE2.controller.data.Tmax +
    sE2.controller.data.setback(1:runTime);
```

There, we look back into the data of the controller to retrieve the constraints to the zone temperatures used.

```
% Plot the result of the simulation
clf;
figure(1);
hold on;
xlabel('Timesteps'); ylabel('Temperatures');
plot(constraintMin,'r--');
plot(constraintMax,'r--');
plot(zoneTemp,'b');
```

This finally plots the temperature in the first zone and the constraints in the MPC problem, the outside temperature and the power input in the first zone in a second plot.
We can observe how the MPC follows the constraints. At the beginning of the simulation, since the weather is not so cold, only internal gains are enough to maintain temperature in the room. the system even has to resort to cooling for a short period to avoid overheating in the room because of the sun radiative gains. Then in the next few days, we see how the setbacks are exploited by the MPC controller.

To check and plot manually the data, a small GUI has been included in the toolbox and can be found under `Classes/visualisationTool`.

4 Some advice to modify existing examples

For the user to create its own simulation, we advice that the user starts from the existing examples and modifies them to his preference. We gather here some advice to do so.

4.1 Modifying characteristics of the simulation

In the current state of the software, the simulation period and timestep depend on the idf file used. To modify those, we therefore advice to directly modify the `RunPeriod` object in the .idf file. The fields of interest are `Begin Month, Begin Day of Month, End Month, End Day of Month, Day of Week for Start Day` .
The object `Timestep` is used to set up the number of timesteps per hour and can take values 1,2,3,4,6,12,60. If the HVAC system is slow enough, 2 time steps per hour was deemed generally sufficient.

4.2 Use own `.idf` files

OpenBuild should be able to support a most `.idf` provided they comply with some rules. In order to generate a model in MATLAB, the file has to be run as such with EnergyPlus, so it has to respect all requirements to do so. The modeling procedure detects objects in the simulation regardless of the rest of the file, so it can extract a thermodynamic model of the building independently of the HVAC system specified in the file. To run cosimulations, the requirements are however more stringent: in the current state of the software, for the model to fit the behavior of the building in simulation, the heating must be inexistent or inactive. Cosimulations do not support sizing periods so make sure that the obejct `RunControl` in the `.idf` file is set to No for the four first fields.