Danmarks Tekniske Universitet

Technical University of Denmark, Kongens Lyngby, Danmark

run.py - a HAWC supplementation

code/documentation: Jonas Lohmann
email: s223685@student.dtu.dk
date: 2023-11-15
version: 0.1.1

Table of Contents

1 What it can help you with

The script *run.py* and its classes are designed to automate the manipulation of .htc files, run HAWC2/HAWC2S simulations, and collect all results in an orderly manner. With a few user definitions, the script fully automatically

- changes any parameters in an .htc file and then runs it
- runs simulations in parallel
- collects the results and saves them to a user-defined directory
- accepts multiple values for the same parameter(s) and runs them one after the other
- incorporates HAWC2S control tuning results into HAWC2 .htc files and runs the HAWC2 simulation
- creates and runs simulations for slices of existing .opt files
- incorporates settings from a HAWC2S .htc into a HAWC2 .htc file

The updated code and the updated (if there are updates) documentation can be found on GitHub. Unexpected behaviour and errors can be posted on run.py - issues or send to s223685@student.dtu.dk. Example use cases are given in Section 4.

2 Prerequisites

To balance user-friendliness (few user inputs) against the generality of the script (many user inputs), some assumptions are hard-coded into the script. These are now briefly explained.

2.1 Always: directory structure

The directory that the file path definitions in your .htc file relate to (here called *rel*) must contain an *htc* and *opt* directory. The *htc* directory holds all (both HAWC2 and HAWC2S) .htc files and the *opt* directory all controller dll's. Jk, all .opt files have to go there. The minimum required directory structure can be seen in Fig. 1.

2.2 For HAWC2 runs: results file name

The *filename* of the *output* block of .htc files needs to be a path including at least one directory. The directory *rel* does not suffice. Result files named *./tsr_10_test* thus do not work. An acceptable *filename* is *./results/tsr_10_test* (here, the needed directory is *results*).

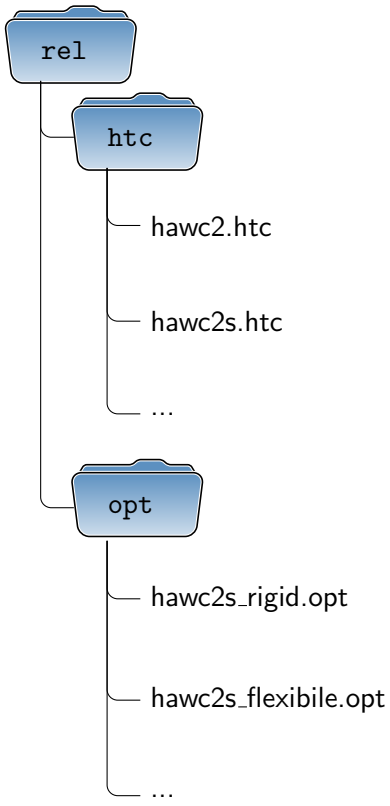

Figure 1: Minimum directory structure

2.3 For incorporating tuned control parameters: parameter file contents

The file containing the tuned control parameters that is created with a HAWC2S run needs to contain parameters for regions 1, 2, and 3, and be for quadratic gain scheduling. This depends on the wind speeds for which the control tuning parameters are calculated and the *gain_scheduling* parameter in the *controller_tuning* block of the HAWC2S .htc.

3 How to use run.py and its parameters

All run.py does is modify .htc files, run it, and collect the outputs. That inherently means that all parameters in the .htc file that run.py doesn't change need to be set up properly beforehand. No parameter in an .htc file that influences the results is changed silently (meaning without the user's action). If the constraints of the prerequisites of Section 2 are satisfied, only the parameters inside the run.py script need to be set and the script run. These parameters are explained in the following subsections.

The parameters `change_params`, `change_opt_file`, and `incorporate_ctrl_tuning` should be used mutually exclusive. Only one of them may not be empty every time run.py is used. See the respective parameter descriptions for when the parameters are called empty.

3.1 hawc_type

type	str
description	tell run.py whether to run HAWC2 or HAWC2S
takes values	"hawc2mb" or "hawc2s"

3.2 dir_htc_relates_to

type	str
description	path from the location of the run.py file to the directory the .htc file relates to
takes values	string representing the path

Note: All file paths specified in the .htc file (to the different structural, aerodynamic, etc. data files) are in relation to a certain directory. This directory is here called "the directory the .htc file relates to". See Fig. 1.

3.3 htc_file_name

type	str
description	file name (i.e. without the .htc extension) of the .htc file to be run
takes values	string representing the file name

Note: This .htc file has to exist in the *htc* directory.

3.4 root_results

type	str
description	path to the root directory of the results in relation to *rel* (see Fig. 1)
takes values	string representing the path

Note: This is not the directory the results are going to be saved to.

3.5 case_name

type	str
description	directory path relative to `root_results` into which the results are saved
takes values	string representing the path

3.6 overwrite_existing_case

type	bool
description	Whether to overwrite the `case_name` directory if it already exists
takes values	True or False

3.7 opt_file_to_be_saved

type	str
description	path of an .opt file relative to *rel* that should be saved into the `case_name` directory
takes values	string representing the path

Note: Using run.py to calculate optimal .opt files with HAWC2S automatically saves the optimal .opt files to `case_name`.

3.8 change_param_type

type	str
description	controls how to change the parameters in the .htc file for a given `change_params`
takes values	"consecutively" or "simultaneously"

Note: This only has an influence on cases in which multiple parameters are to be changed. "consecutively" means each parameter is changed one after the other. "simultaneously" means each i^{th} value of all parameters is set simultaneously.

Assume you want to change `opt_lambda=[7, 8]` and `minpitch=[0, 10]`. Using `change_param_type="consecutively"` will cause *four* runs setting `opt_lambda=7` first, then `opt_lambda=8`, then `minpitch=0`, and finally `minpitch=10`. Using `change_param_type="simultaneously"` will cause *two* runs setting `opt_lambda=7` and `minpitch=0` first, and then `opt_lambda=8` and `minpitch=10`.

3.9 n_parallel_processes

type	int
description	how many simulations to do in parallel
takes values	integers > 0

Note: **DANGEROUS IF MISUSED.** Check yourself before you shrek yourself (or more accurately your PC). Start with `n_parallel_processes=1` and check your RAM and CPU. Then increase it to 2 and see the influence on your RAM and CPU. From that, judge how many processes your PC can handle.

3.10 change_params

type	dict[dict[...[list]]]
description	nested dictionary defining which parameters the change to which values
takes values	deeply nested dictionary; the wanted values for each parameter have to be specified as a list

Note: The nested dictionary is structured according to the blocks in the .htc file. The definition of the new parameter-value combination is based on a key-value (dictionary) pair, too. `change_params` is considered empty when the deepest layers of keys have empty dictionaries as values.

Example setting the optimal tip-speed-ratio: The respective HAWC2S parameter is called `opt_lambda` and it exists in the block `hawcstab2` → `operational_data`. To have 3 runs with different tip-speed-ratios, one would set

```
change_param={
  "hawcstab2":  {
    "operational_data":  {
      "opt_lambda":  [7, 8, 9]
```

```
    }
  }
}
```

To set the controller constant 7 for a HAWC2 simulation, one could set

```
change_param={
  "dll":  {
    "type2_dll":  {
      "innit":  {
        "constant__7":  [100, 200, 300]
      }
    }
  }
}
```

A block in `change_param` is skipped if it ends in an empty dictionary. E.g.,

```
change_param={
  "hawcstab2":  {
    "operational_data":  {
      #"opt_lambda":  [7, 8, 9]
    }
  }
  "dll":  {
    "type2_dll":  {
      "innit":  {
        "constant__7":  [100, 200, 300]
      }
    }
  }
}
```

only changes `constant__7` because the key `operational_data`'s value is an empty dictionary (note the comment indication #).

3.11 change_opt_file

type	list[tuple]
description	run with different .opt files for certain ranges of wind speeds
takes values	list of tuples; first tuple value is the file path relative to *rel* and the second value is a tuple specifying the wind speed range

Note: `change_opt_file` is considered empty when it is an empty list. The file paths have to be specified relative to *rel*.

3.12 incorporate_ctrl_tuning

type	list[str]
description	incorporate the parameters from a HAWC2S control tuning computation into a HAWC2 .htc and runs it
takes values	list of strings representing paths

Note: `incorporate_ctrl_tuning` is considered empty when it is an empty list. The file paths have to be specified relative to *rel*.

3.13 hawc2_use_from_hawc2s

type	list[tuple[str, str]]
description	specify all parameters and values a HAWC2 .htc file should use from a HAWC2S .htc file
takes values	list of tuples with the first tuple entry denoting the HAWC2S parameter and the second entry the HAWC2 parameter

Note: Currently, this only works if `incorporate_ctrl_tuning` is used and the directories that contain the HAWC2S control tuning parameter file contain the HAWC2S .htc file that was used to create the control parameter file.

The parameters have to be specified as a string in the format of `"block1.block2. ... .parameter"`, i.e. to use the constant power setting from a HAWC2S .htc in a HAWC2 .htc file use

```
incorporate_ctrl_tuning =
  [("hawcstab2.controller_tuning.constant_power",
  "dll.type2_dll.init.constant__15")]
```

4 Example use cases

run.py can always be used if just a single simulation should be performed or when multiple parameters need to be changed. The three main use cases the author has used are quickly shown.

4.1 Setup

Given a directory structure like seen in Fig. 2 and assuming that `redesign.htc`'s file paths relate to the `input` directory, the following parameters in run.py have to be set:

```
dir_htc_relates_to =   "../data/HAWC/input"
htc_file_name =        "redesign.htc"
```

Then, the directory `output` is chosen as the root directory for the results:

```
dir_htc_relates_to =   "../output"
```

4.2 Simple run

Assume `redesign.htc` to be a HAWC2S .htc file. This implies

```
hawc_type =   "hawc2s"
```

Now, assume `redesign.htc` is already ready to use using a command prompt. The .htc file is set to calculate the optimal operational data, steady-state power values, induction values, and tune the controllers. These are a lot of files you'd like to collect in a directory `all_calculations`. This means

```
case_name =   "all_calculations"
```

Now additionally set

```
overwrite_existing_case =   True
skip_safety_warning =       False
opt_file_to_be_safed =      None
change_params =             {}
change_opt_file =           []
incorporate_ctrl_tuning =   []
```

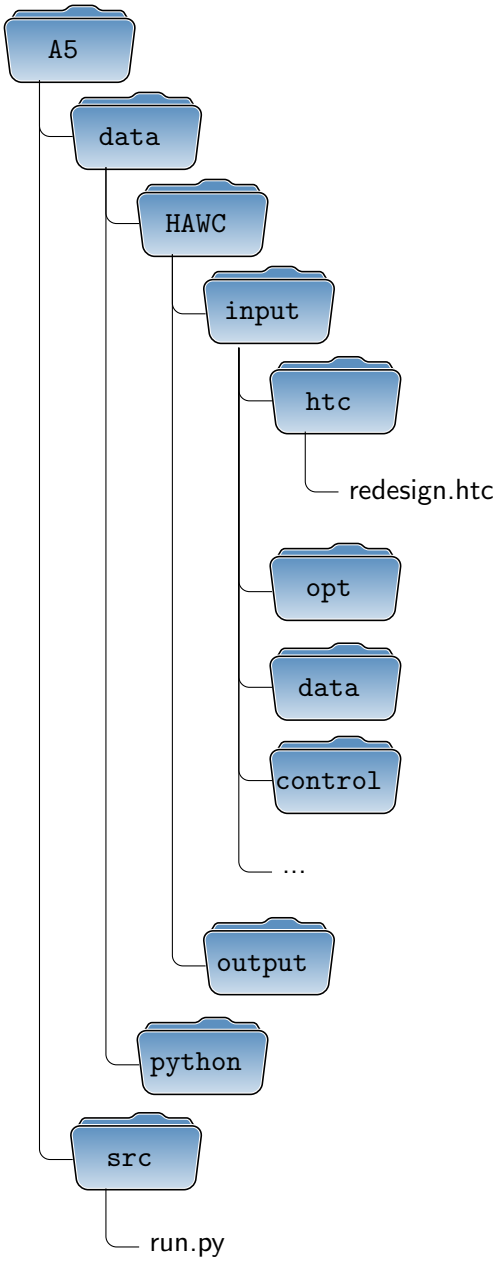

Figure 2: Exemplary directory structure

and run run.py. Parameters that aren't specifically set in this example don't have an influence in this case. Once the script is done, all results will be in `A5/data/HAWC/output/all_calculations`.

4.3 Finding the optimum Tip Speed Ratio

Change `redesign.htc` to calculate the optimal operational settings. Then, use the same settings as in Sections 4.1 and 4.2 except for

```
case_name =       "tsr_optimisation"
change_params =   {"hawcstab2":{"operational_data":{"opt_lambda":[5,6,7,8,9,10]}}}
```

And set `n_parallel_processes` to what your PC can handle (or what you like). Once the script is done, `A5/data/HAWC/output/tsr_optimisation` contains 6 directories called `opt_lambda_i`, with `i` the tip speed ratios 5 to 10. In each directory are the results for the respective tip speed ratio.

4.4 Controller optimisation

Assume there is a HAWC2 `redesign_hawc2.htc` file next to the `redesign.htc`. Set the latter to perform control parameter tuning. Create a list of tuples with different (ctrl_frequency, ctrl_damping) called `ctrller_values`. Then set

```
case_name =           "ctrl_optimisation"
change_param_type =   "simultaneously",
```

`n_parallel_processes` again to what you like and

```
change_param={
  "hawcstab2":  {
    "controller_tuning":  {
      "partial_load":  ctrller_values,
      "full_load":  ctrller_values
    }
  }
}
```

Now run run.py. In the meantime, set the HAWC2 `redesign_hawc2.htc` for your desired wind conditions and simulation time. Don't forget the note in Section 2.2. Once the script is done, the directory `A5/data/HAWC/output/ctrl_optimisation` will be filled with directories corresponding to the different controller values. In each of them, a file with the tuned parameters exists. Since all of them have the same name, one can `os.listdir()` through the `case_name` directory to quickly define all tuned control parameter files an save them as `tuned_files`. Then set

```
hawc_type =                 "hawc2mb"
htc_file_name =             "redesign_hawc2.htc"
case_name =                 "HAWC2_sim"
incorporate_ctrl_tuning =   tuned_files
```

and run again. After the script is done, each directory for a unique controller damping and frequency setting contains a directory `HAWC2_sim` containing the .hdf5, .log, and .htc file. The .htc file there is the original `redesign_hawc2.htc` file with the changed tuned controller values.