Tutorial 7 — Editing and Launching a DSE in the App

Overview

This tutorial will show you how to:

1. Open a DSE configuration in the INTO-CPS app.
2. Edit the configuration in the app
3. Launch a DSE from the app
4. Read the results and find the details for each simulation

Requirements

This tutorial requires the following tools from the INTO-CPS tool chain to be installed:

- INTO-CPS app 3.10 or above
- Python 2.7 with numpy and matplotlib – you have to install it yourself
- DSE scripts 0.2.0 or above – accessible through the INTO-CPS App Download Manager

You may have been provided with tools and tutorials on a USB drive at your training session. Otherwise:

- Follow Tutorial 0 with the guidelines to install the INTO-CPS Application and COE.
- Ask your instructor for the tutorial materials. These are available for students and members of the INTO-CPS Association[1].

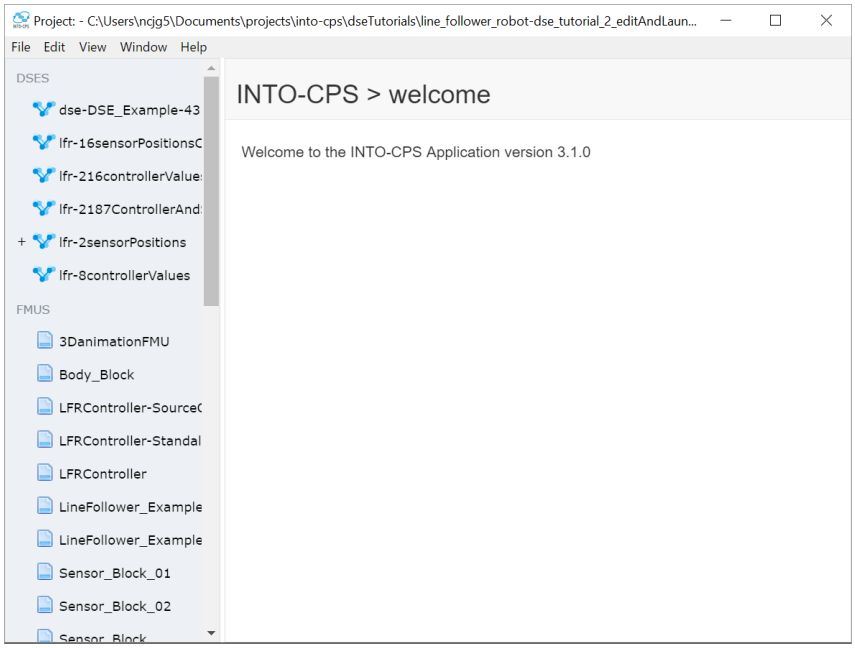
1 Preparing the Workspace

This tutorial requires the contents of Tutorial7.zip to be unzipped and placed in your desired location. Please note that the DSE scripts currently do not handle path names including spaces, so please ensure that the path to where Tutorial7.zip extract does not include spaces.

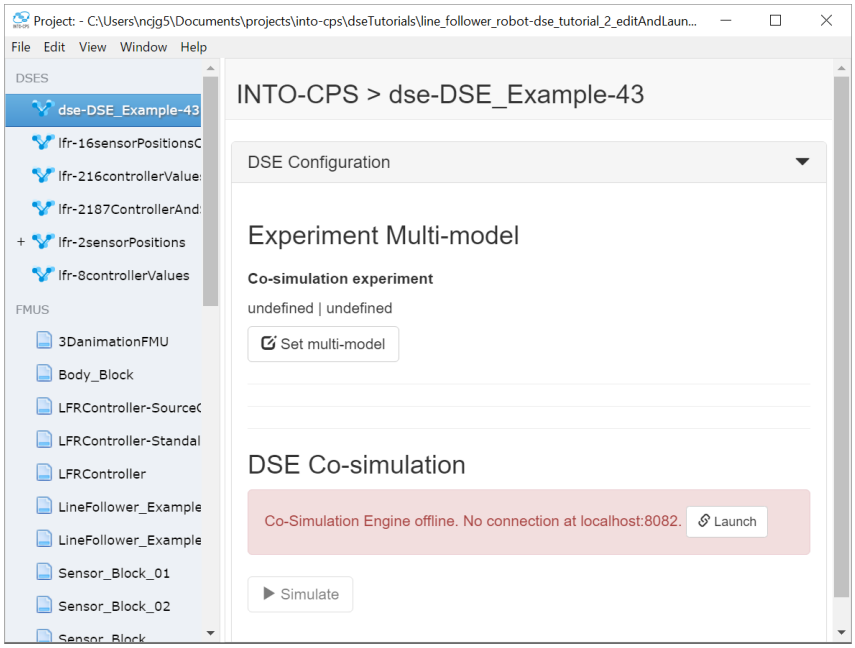
[1] `https://into-cps.org/`

2 Opening a DSE Configuration

After opening the app, you will need to move to the tutorial_7 workspace that you unzipped in the previous step. In the app select *File* > *Open Project* and navigate to the your *tutorial_7* folder. With the folder open you should be presented with the app welcome screen as below.

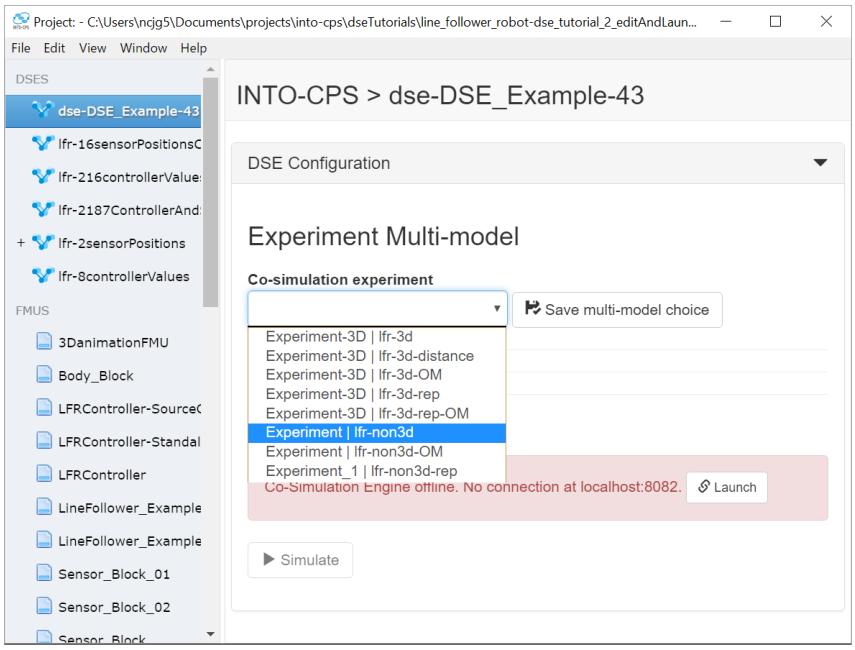


The name of the DSE configuration we will use in the tutorial starts with dse-DSE_Example, which was the name given to it when the app imports the DSE configuration that was exported from Modelio in Tutorial 6. To open the configuration, double click on its name in the DSEs section.

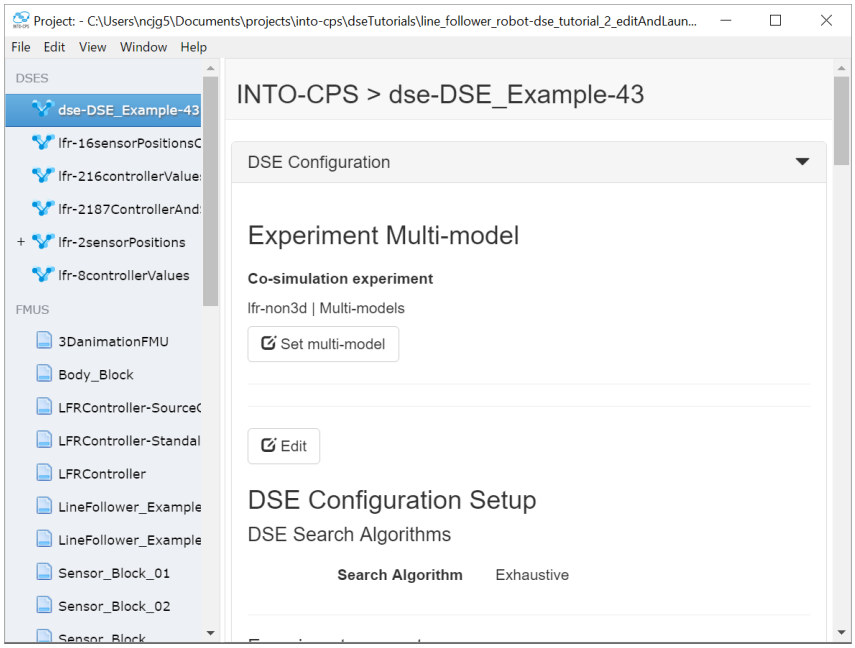


A DSE configuration can only work in concert with an existing multi-model since it only describes the changes that should be made to the model, it does not describe the connections or FMUs. So the first thing

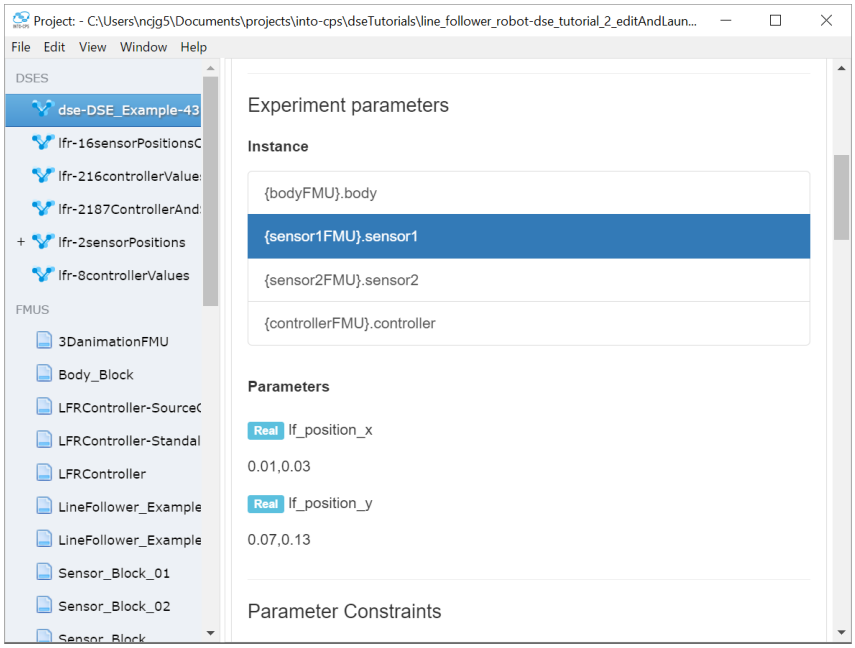
we need to do is select the multi-model the DSE scripts will use. As below, click on 'Set multimodal' and then scroll down to select *Experiment/lfr-non3d*. Then click on *Save multi-model choice*.



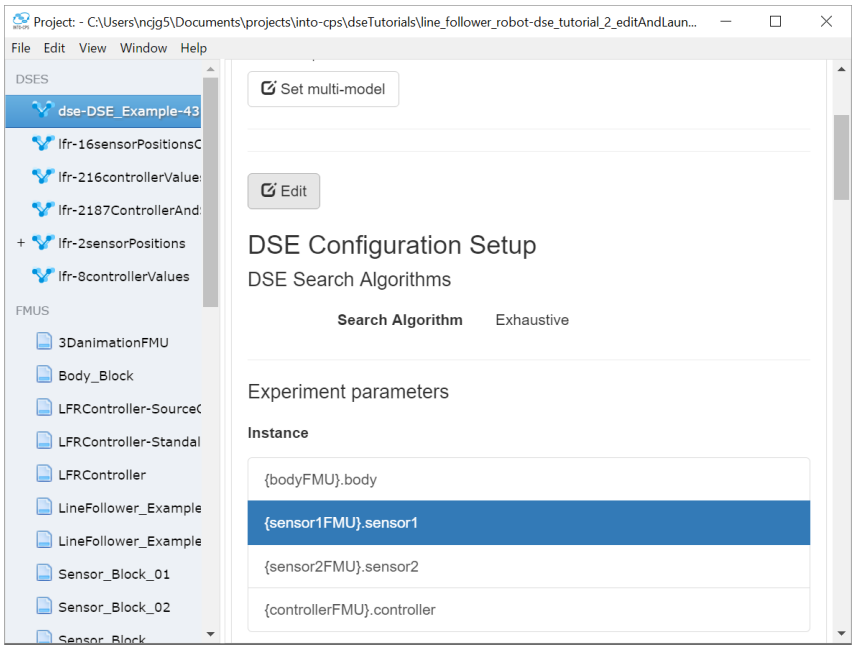
With the multi-model set, the app will now load and parse the DSE configuration and you should be able to see the top of the new panel below, with the algorithm choice *Exhaustive* showing.



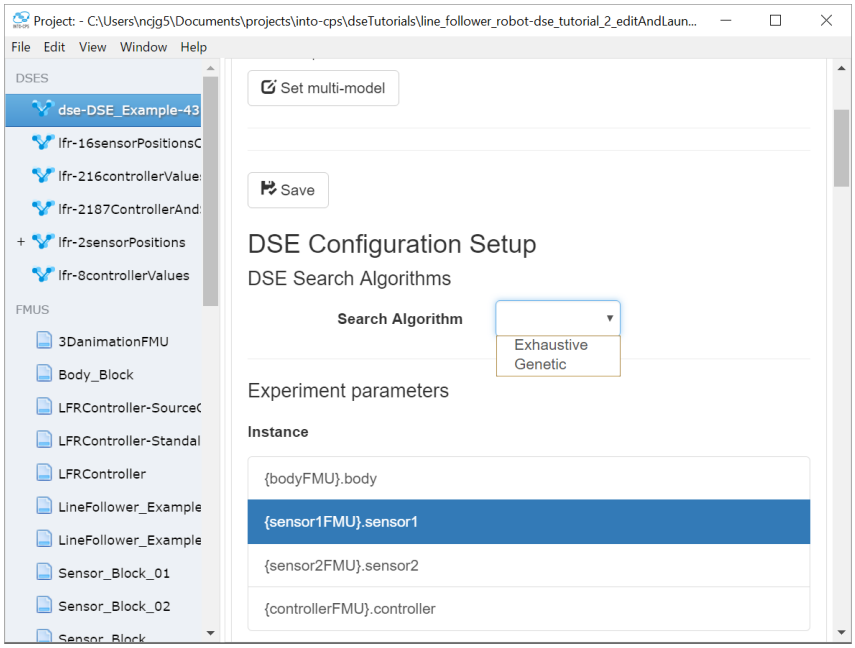
At this point, the app is just showing the contents of the configuration, so you may scroll down and view the contents. You may for example scroll down to the parameters section and see the values each parameter of each FMU will take by clicking on the name of the FMU instance. In the example below, the '$sensor1$' FMU is selected and the values for its '$lf\_position\_x$' and '$lf\_position\_y$' parameters are shown.



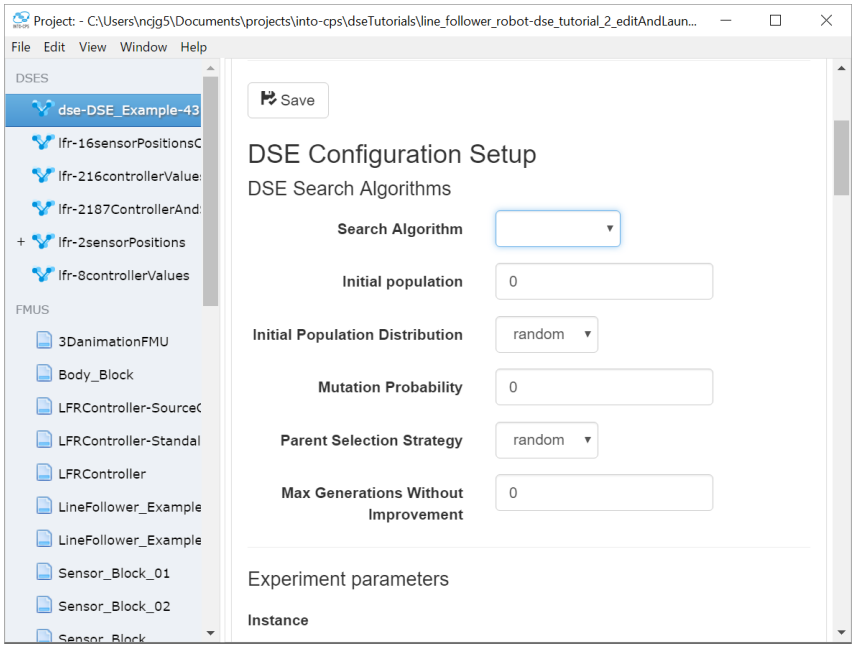
To switch the app into editing mode, scroll back to the top of the DSE configuration section and click on the *Edit* button. This will update all the fields in the DSE configuration to be editable and the button will change to say *Save*, ready for when you are finished editing.



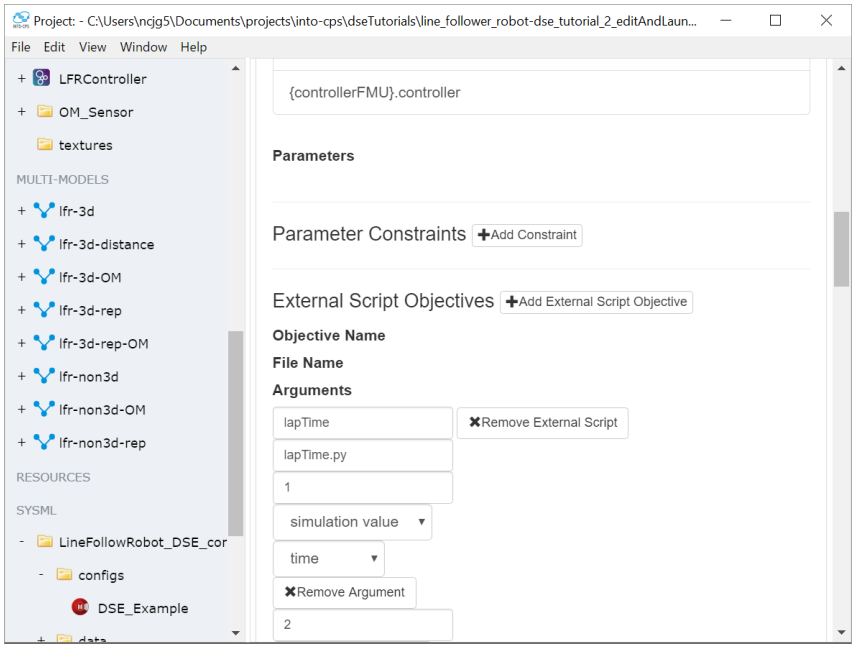
The view changes to give editable options. Starting with the algorithm section, clicking on the *Search algorithm* dropdown will reveal the two options which are *Genetic* and *Exhaustive*.



If you click on *Genetic*, this will reveal the options available for a genetic search. The initial population parameter accepts an integer value > 1, which defines the number of designs that will be simulated at the start of the algorithm. The population distribution option currently only has one option random, meaning the initial population will be randomly distributed throughout the design space. Other options are currently in development and will be added in the future. The mutation probability accepts integer values ($0 < value < 100$) which describes the probability that a parameter will spontaneously adopt another valid value when new designs are generated by breeding two parents. The parent selection strategy affects how parents are selected during the breeding phase of the genetic algorithm. Currently the only option is random, meaning that the only factor affecting parent selection is their rank, where the designs on rank 1 (best) have a higher probability of being selected than those on rank 2, and designs on rank 2 have a higher chance than those on rank 3 and so on. Other options will be added here in the future. A more detailed description of how the genetic algorithm works and its parameters may be found in the INTO-CPS deliverable D5.2D 'DSE in the INTO-CPS Platform', which may be found on the INTO-CPS website. We will not be using the genetic algorithm in this tutorial, so select the *Exhaustive* algorithm in the *Search algorithm* dropdown. This will remove the genetic options and put the configuration back the way we need it for the tutorial.

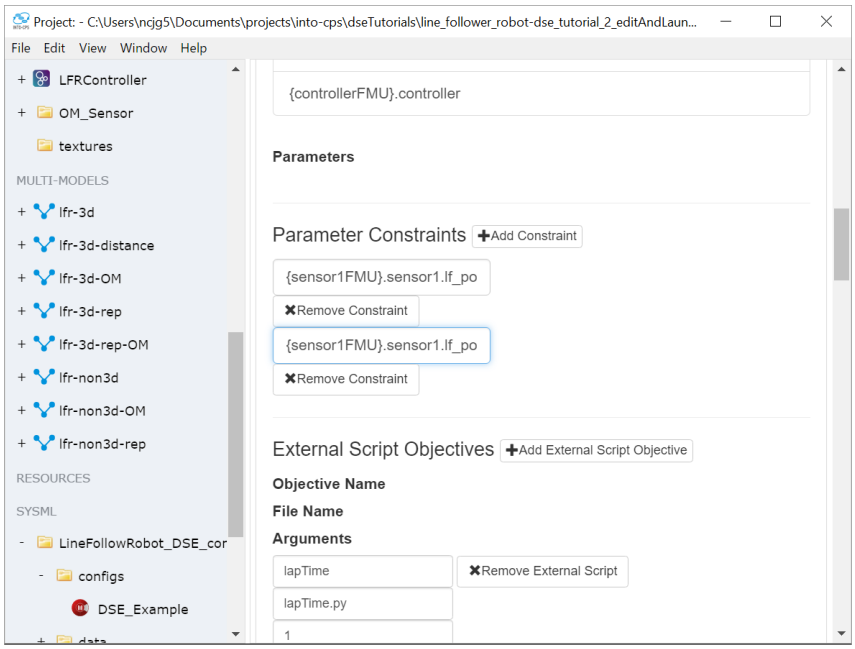


Scrolling down a little further we arrive at the *Parameters* section, and specifically the *Parameter Constraints* section, which is currently empty. Without adding constraints, the exhaustive algorithm will explore all combinations of the parameter values we have defined. Now since the x and y coordinate parameters of the left and right line follower sensors are independent of each other, if we do not add constraints we leave ourselves open to trying the robot with asymmetrical sensor placement, such as the left sensor close to the robot with the right hand sensor way out in front. In this case we don't want this so we are going add constraints to make the design symmetrical. Constraints are written as boolean expressions using Python syntax and referencing the full names of the parameters the DSE configuration knows about. In this case, to make the designs symmetrical, we want to make the y coordinates of the left and right sensors the same while making the x coordinate of one sensor the negation of the other sensor.

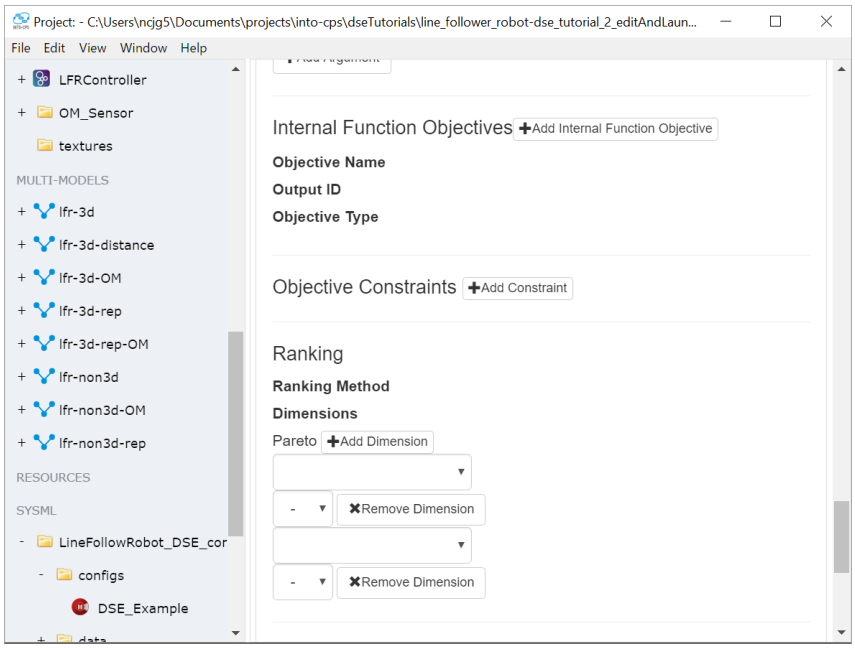


To add the constraints, click on the 'add constraint' button, this will add a text box and enter a single constraint. Repeat this for each new constraint you wish to add. For the purpose of the example you will need to add the following two constraints, the first ensures the y coordinates of the two sensors are the same and the second mirrors the x coordinate.

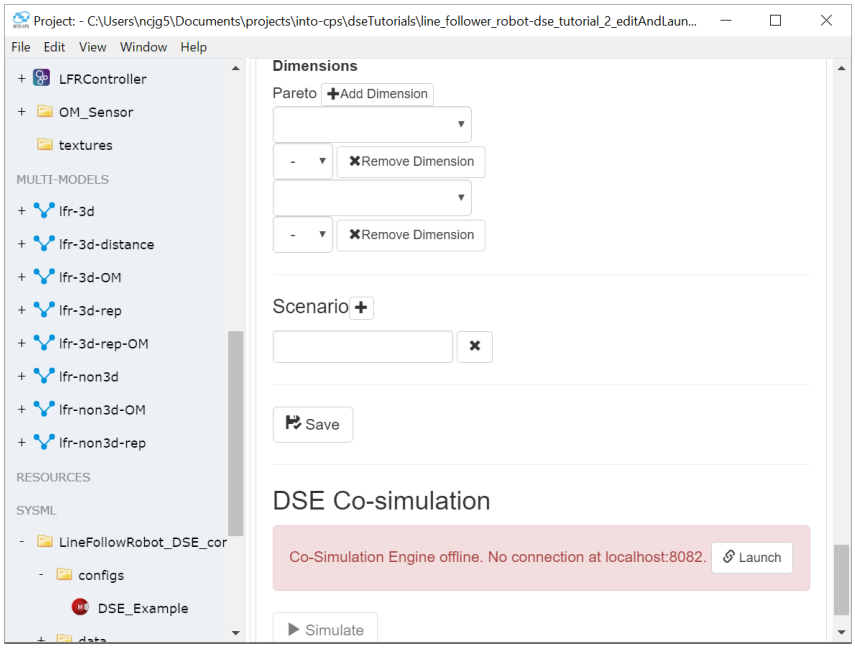
```
{sensor1}.sensor1.lf_position_y == {sensor2FMU}.sensor2.lf_position_y
{sensor1}.sensor1.lf_position_x == - {sensor2FMU}.sensor2.lf_position_x
```

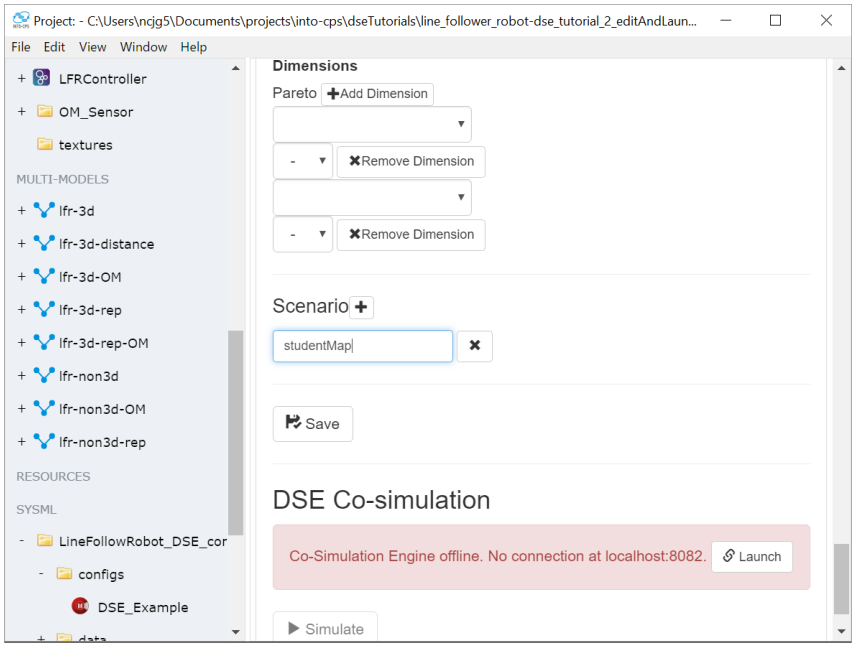


The next image shows a section named *Objective Constraints*. This is not yet implemented, but when it is you will be able to add boolean expressions referencing the Objective names. These constraints will control which designs are presented to the engineer in the final results, and, in the genetic algorithm, will only let acceptable designs enter the breeding phase of the algorithm.
Next is the Ranking section. Recall from the previous tutorial (Tutorial 6), that we set the Pareto ranking to use the mean cross track error and lap time objectives. This is still true here, though in editing mode the currently selected objectives are obscured. If you want change the objectives used for ranking, you could select their names from the two drop down boxes.
Hint: don't do this now or you will get different results later in the tutorial!.

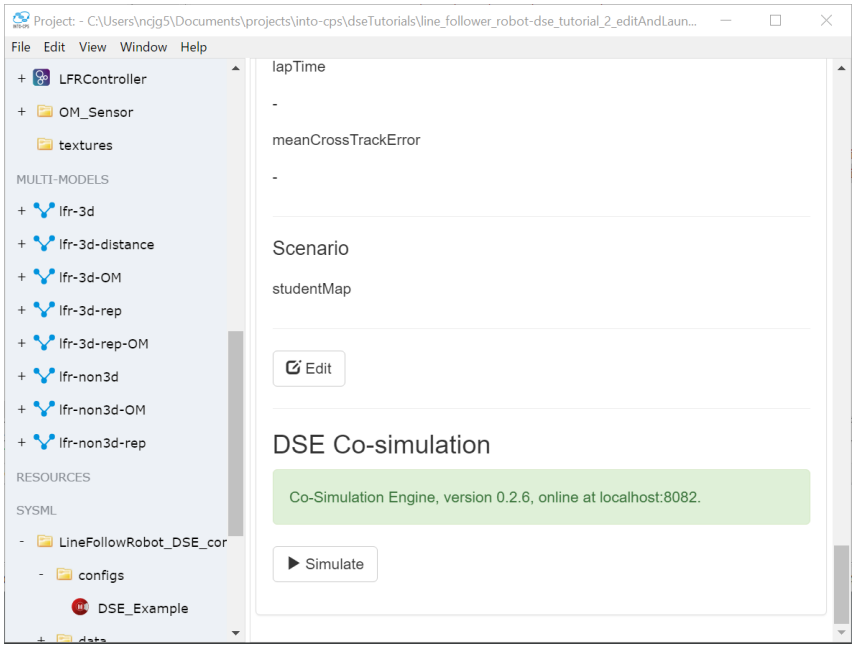


In the final part of the configuration, we need to edit the Scenario name. Scenarios are a way of letting the objective scripts know what test environment the CPS was faced with, which in the case of the line follower robot, is the name of the map it is using. This details of what the objective scripts do with this name are discussed in Tutorial 7 on objective scripts. For now we simply need to enter the name of the map which is `studentMap` in the scenarios text box.

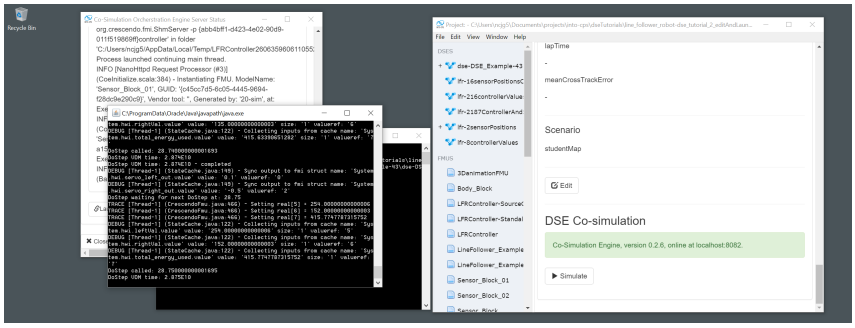




With the scenario name entered, you may now save the configuration and launch the COE ready to run the DSE. So click on the *Save* button and then on the *Launch* button for the COE.



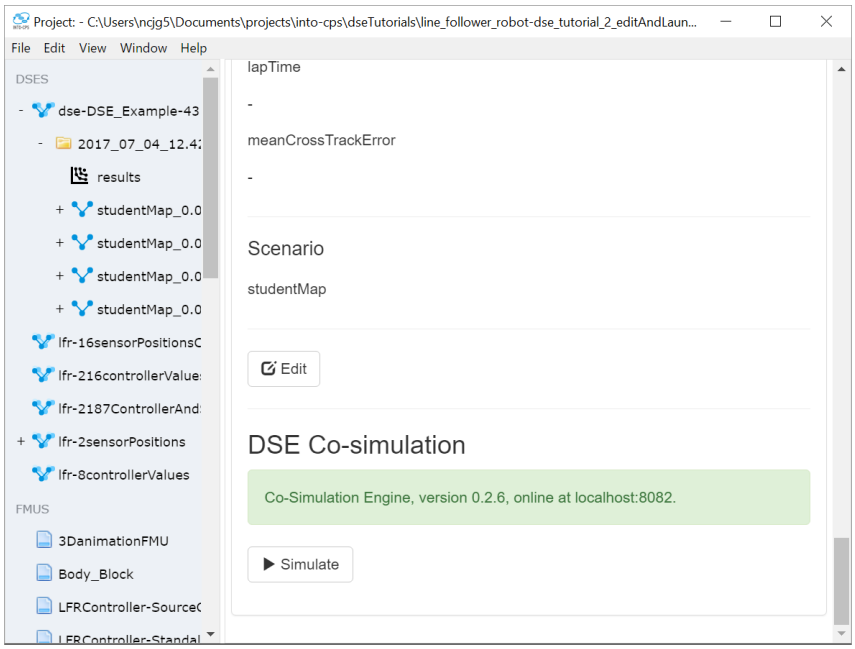
When the COE status panel is green, you may click on the *Simulate* button to start the DSE.



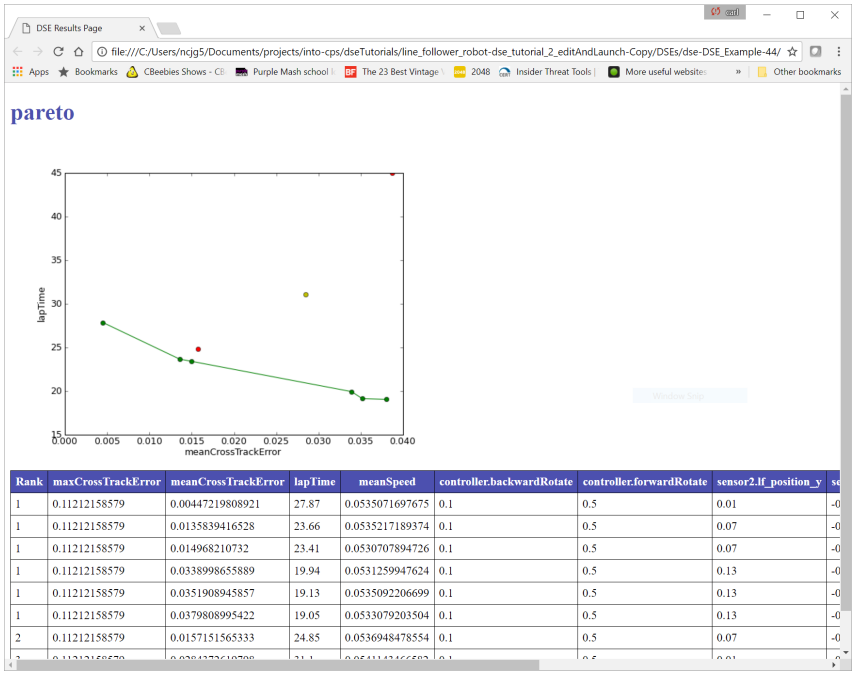
This DSE will run 9 simulations and takes a little under 9 minutes on a 3Ghz laptop; It may take more or less time on different hardware. If you are on Windows you will see terminal windows appearing as the app launches the DSE scripts. This confirms that the DSE is running, but there may be less obvious signs of activity on other operating systems. Currently, the main indicator that a DSE is complete may be found in the app itself. In the DSEs section, expanding the structure under the name of the DSE configuration reveals a folder named for the date and time that the DSE commenced. Expanding this directory reveals subdirectories containing results from each individual simulation. When the DSE is complete there will be another file in this directory named '*Results*', as below.

Common Issues:

- If nothing happens after you click the simulate button, check that your python installation is the correct one, and make sure the right python version is accessible in the *PATH* variable.
- If the correct python version is in the *PATH*, then double check that the constraints correctly refer to the FMU instances.

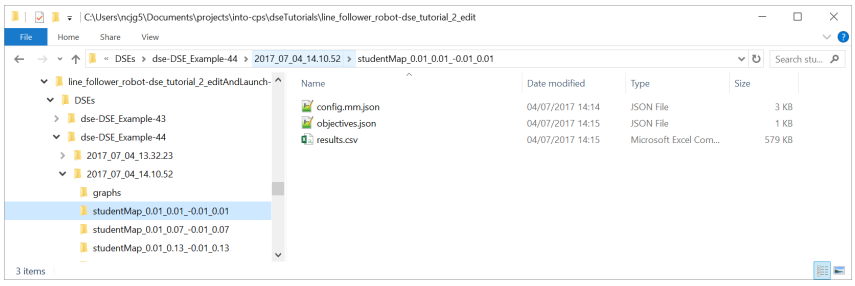


Double clicking on the Results icon will open the results HTML file in your default internet browser. The results consist of two parts: The first part shows a graph of results in the form of series of points representing each individual simulation. The non-dominated set (best results) are shown in green with a green line connecting them, with the remaining points shown in red and yellow. Below the graph are the details of the results, ordered by rank. Each row represents one design and includes details of the calculated objective values, along with the parameters that resulted in that particular outcome.
Results in rank 1 (green) are suitable for further investigation or testing on a physical prototype.



Rank	maxCrossTrackError	meanCrossTrackError	lapTime	meanSpeed	controller.backwardRotate	controller.forwardRotate	sensor2.lf_position_y	se
1	0.11212158579	0.00447219808921	27.87	0.0535071697675	0.1	0.5	0.01	-0
1	0.11212158579	0.0135839416528	23.66	0.0535217189374	0.1	0.5	0.07	-0
1	0.11212158579	0.014968210732	23.41	0.0530707894726	0.1	0.5	0.07	-0
1	0.11212158579	0.0338998655889	19.94	0.0531259947624	0.1	0.5	0.13	-0
1	0.11212158579	0.0351908945857	19.13	0.0535092206699	0.1	0.5	0.13	-0
1	0.11212158579	0.0379808995422	19.05	0.0533079203504	0.1	0.5	0.13	-0
2	0.11212158579	0.0157151565333	24.85	0.0536948478554	0.1	0.5	0.07	-0

The results may also be explored by opening a file browser and navigating to the *INTO-CPS* project folder, then opening the DSEs directory, then the subdirectory containing the DSE config. The results may be found in the directory labelled with the date and time the DSE was completed. In there you will find subdirectories named with the values of the parameters used in that simulation. In each of these directories you will find *results.csv* containing the raw simulation results and two other files.



The *objectives.json* file contains the objective values calculated from the raw simulation results.


```
{
    "lapTime": 23.410000000001048,
    "maxCrossTrackError": 0.11212158578971308,
    "meanCrossTrackError": 0.014968210731969004,
    "meanSpeed": 0.05307078947255971
}
```


The *config.mm.json* file contains the complete multi-model configuration sent to the coe when launching the simulation. In this you may find all parameters used to produce the simulation result, including those not altered by the DSE configurations. The only detail not included in this file is the length of the simulation. This file can provide the details required to replicate a simulation, perhaps with another multi-model including 3D output for visualisation.


```
"parameters": {
    "{controllerFMU}.controller.backwardRotate": 0.1,
    "{controllerFMU}.controller.forwardRotate": 0.5,
    "{controllerFMU}.controller.forwardSpeed": 0.4,
    "{sensor1FMU}.sensor1.lf_position_x": 0.01,
    "{sensor1FMU}.sensor1.lf_position_y": 0.07,
    "{sensor2FMU}.sensor2.lf_position_x": -0.01,
    "{sensor2FMU}.sensor2.lf_position_y": 0.07
}
```