**Lab Session 4: Learning to Control the Digital Twin**

**Objective**: The goal of this lab is for you to find, in an automated way, a sequence of actions to get the pendulum in upright position. In this lab you will use the digital twin to learn these controls. We assume that if you use the optimized model that you have obtained in lab 3, the action sequence should translate directly on the physical pendulum. You will attempt to find the sequence through the use of a genetic algorithm (GA).

Discuss the following with your group before you start:

*What are the specific challenges in using genetic algorithms   
for controlling a pendulum and achieving an upright position?*

For this lab session, you will need the following files: **Lab\_4\_control\_optimization.py**, **run\_actionlist.py** and **Lab\_1\_model\_simulation.py.** You will also need to place the lab\_3 python file in the same directory as the **Digital\_Twin.py** as the digital twin class will be imported.

First set the self.delta\_t in the Digital\_Twin.py to 0.005 as this is what we use during the optimization process.



Now test the functioning of the digital twin with Lab\_1\_model\_simulation.py, if that works you can continue.

The goal is to create an output sequence for our model, [3, 0, 7, 0, 3, 0, 7, 0, 3] e.g. You have seen this in lab 2 where these sequences map to moving to the right and left for a certain amount of time. Now with a genetic algorithm we will try to find a sequence that will result in reaching the upright position. How do you decide on the parameters for your genetic algorithm (e.g., population size, mutation rate, crossover rate)? What strategies will you use to fine-tune these parameters?

* *Decide the simulation length, in what amount of time do you think the process could be performed?*
* *What would your population size be?*
* *What would the parent size be (the individuals going to the next generation)?*
* *How does the mutation rate effect the result?*
* *How many generations does it on average take you to achieve the goal?*

Test the results in the run\_actionlist.py by copying the resulting sequence in the action list:



* Now select multiple solutions, are there repeating patterns? Visualization helps such as binning or creating graphs.
* Can you come-up with another method of evaluation that will improve the learning rate?
* What is fundamentally the problem with the current crossover function in the GA? Can you come up with a better version? **(Bonus question)**

Learning these controls can be computationally expensive. Discuss how you could further minimize the computation you would need for this optimization. Could you learn step by step (instead of x seconds at ones)? Could you find a better initialization of the population?