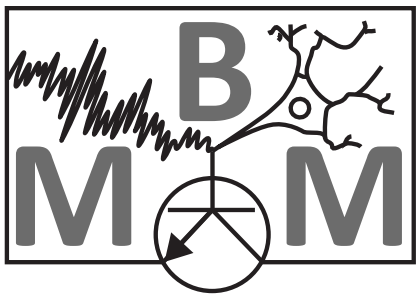
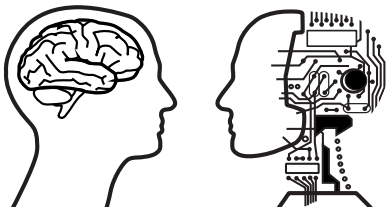


Mind, Brain, and Models 2023



CNCR

Synchronisation

In this lab, you will use a very simple object-oriented programming (OOP) approach (https://en.wikipedia.org/wiki/Object-oriented_programming). You will simulate agents that synchronise with a metronome or with each other to try producing isochronous behaviour (e.g. clapping). We will first look at the behaviour of a single agent and then of two agents. The object-oriented approach will allow the creation of multiple agents (called objects or instances), each independent from the others by having separate variable values (called fields, attributes or properties). [In a standard object-oriented approach, we would let multiple agents "know" about each other and "communicate" with each other directly (using appropriate functions often called methods or procedures), but here to keep things simple we will always be in charge of letting the agents know about the clapping of other agents.]

1) Download the two files `labsync` and `clappingAgent`. Save them in the same folder. Read and try to understand both files. You will not need to modify the `clappingAgent`, but only `labsync`.
2) Your first task is to execute `labsync` and understand what the clapping agent is doing out of the box. Set `initialClapTime=.1`. The agent will treat this initial clap time as an error because it is passed as `error` in the `playClap` function. The `error` passed is the difference between the actual clapping and the perfect metronome timing. Perform an experiment by changing the `selfCorrection` factor and observing what happens to the timing of the claps.
3) Plot the six error values on a graph to obtain a graph akin to slide 8 in the lecture. Include the graph in your report and describe the behaviour of the agent relating it to the concept of metronome synchronisation.
4) Increase the number of claps to 20 by entering them in a loop and plot the errors as before. Set `stdClappingError=.1` and `initialClapTime=0`. Observe how the behaviour of the agent changes as a function of `selfCorrection` (e.g., use values of `selfCorrection` like -1, 0, 1, 2). Execute the `labsync` script multiple times and plot graphs lines on the same axes to test if a pattern emerges in the graph across different executions.
5) Include the graph in your report. Describe the pattern found and write a comment on your findings about the change in agent behaviour depending on the parameter value.

6) Now we will look at how two agents sync to each other without an external metronome. Create two instances of `clappingAgent` that have correction values (`alpha`) with two variables (rather than one as in the previous tasks): `[selfCorrection alphaOther]`. By setting `selfCorrection=0`, the agent will ignore the metronome. The `error` passed to each agent will also be composed of two variables: the first is identical to the one with a single agent (but with `selfCorrection=0` it will not be used), and the other is the asynchrony between the two agents.
7) Using a loop, show that the agents drift apart with `alphaOther=0`, `and` maintain in sync with positive `alphaOther`. Using similar graphs as before, visualise one example of what happens with over-correction and one of what happens with under-correction and comment.

As your project of choice, you could implement the synchronisation of multiple agents, automatic communication between agents, or period correction in addition to phase correction.