# Stroop-effect in Python ACT-R

Sanne Eeckhout

*Leiden University, Course Name: Cognitive Modelling | Studentnumber s2259796*

January 2, 2022

**Abstract.** This document contains a simulation of the Stroop-effect by J.R. Stroop. It starts with an introduction on the Stroop-effect and the design principles of the task. It contains a general overview of the working principles of ACT-R and its architecture is explained. The paper includes an implementation of a simulated Stroop-task using the ACT-R-based library ccmsuite in Python, containing an detailed description of the Python code. The data is analyzed by creating several figures using matplotlib and it's results are compared to the original findings of Stroop.

## 1 Introduction

The Stroop-effect, reported by John Ridley Stroop, is one of the most known phenomena in the cognitive sciences [**stroop1935studies**]. It refers to a psychological task, the Stroop-task, which consists of a word naming a color printed in a certain ink color, where the letter string and the ink color either referred to the same color, a different color, or the ink color was set to black. These conditions were named as congruent, incongruent and neutral (or control) respectively. The phenomena presented that naming an incongruent color takes significantly longer than the congruent or neutral trials. Since the stroop-effect is an interesting phenomena presenting the human brain, it can be interesting to simulate this in cognitive architectures, such as ACT-R.

ACT-R, which stands for Adaptive Control of Thought—Rational, is a hybrid cognitive architecture and can be seen as modelling higher-order functions of the brain [**lovett1999modeling**]. ACT-R is a production system, an environment that uses if-then rules (or statements) to define what needs to happen when certain pre-conditions are met [**sanne**]. The architecture of ACT-R can be explained by its representation, modules and buffers. The representation in ACT-R are of the form slot-and-value pairs. An example of this representation can be seen below.

```
    inputWord:red inputColor:green
```

One of the main components of ACT-R is modules. There are two types of modules, which run in parallel. There are perceptual-motor modules, such as visual and motor modules that interact with the environment and memory modules, which can be divided in declarative memory and procedural memory [**sanne**]. The buffers in ACT-R try to model the working memory of the brain, where data is stored in chunks. An overview of the mechanism of the ACT-R architecture can be found in figure 1 below [**ritter2019act**].

The most important reason ACT-R is an interesting architecture for modelling, is due to it being a programming language as well as a theory of how the brain works [**sanne**]. The architecture reflects aspects of certain brain regions in human cognition [**ritter2019act**]. In later versions of ACT-R, such as ACT-R 5 and ACT-R 6, Anderson et al. [**anderson2004integrated**] states that he believes certain modules in ACT-R can be directly linked to regions of the brain where higher order brain functions such as learning and reasoning takes place [**sanne**]. For example, the declarative module represents the functions of long term memory in the Hippocampus, whereas the buffers represent the working memory (in the Parietal or Motor cortices) and the productions represent the functions of motor control happening in the Basal Ganglia. ACT-R can be and has been used to create models in problem solving, perception, attention, learning, communication and many more [**budiuwebsite**] [**sanne**].

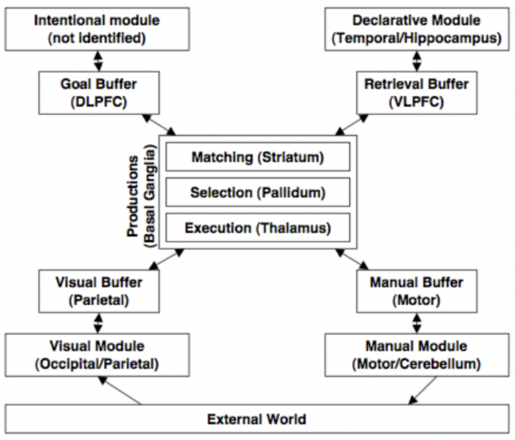
Intentional module (not identified)
Declarative Module (Temporal/Hippocampus)
Goal Buffer (DLPFC)
Retrieval Buffer (VLPFC)
Productions (Basal Ganglia)
Matching (Striatum)
Selection (Pallidum)
Execution (Thalamus)
Visual Buffer (Parietal)
Manual Buffer (Motor)
Visual Module (Occipital/Parietal)
Manual Module (Motor/Cerebellum)
External World

Figure 1: Overview of the ACT-R architecture [**taatgen2005act**]

## 2 Methods

The code simulating the stroop-effect in ACT-R is written in Python 2.7 using the library CCM-Suite and can be accessed via the link in the Appendix. The code includes some packages that need to be installed, these can be found in the `README.md` file. The code can be run using

```
python2 myModel.py
```

The program contains two classes, an environment and an agent and a main function. The environment is a CCM Model, whereas the agent is an ACT-R. It contains a few globally used variables, which are shown in table 1. The pseudocode is given in algorithm 1.

Table 1: Description of global parameters.

| **Variable** | **Description** | **Options** | **Default** |
|---|---|---|---|
| `CONDITIONS` | Sets the condition of the experiment. "Control" means experiment where color of the word is black, "Congruent" means the color of the word corresponds with the word and "Incongruent" means the color of the word is a different color than the word itself | Control, Congruent or Incongruent. | Control |
| `TASK` | Sets the task that is going to be performed. "ColorNaming" means the answer should be the color of the word and "WordReading" means the answer should be the word itself, regardless of the color | ColorNaming or WordReading | ColorNaming |
| `LATENCIES` | A list containing the latencies for the declarative memory, these correspond to the TASK and MODE | - | - |
| `t` | The time the agent runs the simulation in miliseconds. These correspond to an actual person in seconds | An integer | 5 |

***Agent*** In the agent class are five buffers defined: the input, output, goal, word count and a buffer for the declarative memory. The input buffer consists of two slots, the word that is shown

and the color that its shown in. The output buffer consits of one slot, the decision of the agent. The goal is a slot containing the stage of the process. It uses a string naming what the current reached goal is and defines which function should be used next. The word count consists of two slots, the total word count, counting how many words the simulation has answered and a process slot, which is used to count how far along the agent was in the process of answering to the stimuli. These slots are used in the main function.
The DM Buffer is used to set a certain latency, which represents the process of the brain and how long it takes to make a decision and is used to simulate the behaviour for the different tasks. The latency is set in the `main()` function by `LATENCIES`, which will be explained in the next paragraph.

**Algorithm 1** Stroop-effect pseudocode

```
Require: TASK, MODE, Buffer()
  for each combination of TASK and MODE do
      while $time < t$ do
          Start word: increment process
          Look at color: increment process
          Read word: increment process
          Decide: increment process
          if task is word reading then
              $Decision \leftarrow word$
          else if task is color naming then
              $Decision \leftarrow color$
          end if
          Output: $wordcount \leftarrow wordcount + 1$
      end while
  end for
```

Let us examine this pseudocode in more detail. The subcode inside the while-loop is declared by the agent class. Each function is a step in the progress of processing the stimuli and choosing an answer. This is, in order, starting the word, looking at the color, reading the word, deciding and answering.
*Starting the word* is done by choosing a random word of a color and corresponding ink color. These are then set in the input buffer to the word and color that were chosen.
*Looking at the color and reading the word* are both simply steps in the algorithm to increment the process
*Deciding* is done by either naming the input color or the input word, depending on the task. The *output* stage is the final stage of the process of one word and consists of checking the answer, as well as incrementing the total word count by one. This stage also defines the next step to be a new starting word, making the algorithm run infinitely, until stopped.

***The Main Function*** In the main function, the agent and environment are defined and run for a certain amount of time, defined by `t`. The agent is run and after the previously defined time has passed, the reaction time is calculated. This computation can be found in the following equation (1).

$$RT = \left( \frac{t}{word + process} \cdot 1000 \right) + \mathcal{N}(RT, 30) \tag{1}$$

The reaction time is obtained by adding the process to the word count of the wordcount buffer. The result is the amount of stimuli the agent was able to answer in the aforementioned time. Using this, we can calculate the amount of time it took to answer one stimulus in milliseconds. To create some variation in the reaction time, a random number is added to the reaction time, taking a normal distribution with a standard deviation of -30 milliseconds to 30 milliseconds. These values are chosen based on an experiment with running multiple latencies on the agent and pinpointing which latency gives the closest reaction time as stated by Cohen et al [**cohen1990control**]. Each reaction time is saved into an numpy array corresponding its condition and saved to a file.

To visualize the data, the file `datavisualisation.py` is used and can be run using

```
python datavisualistation.py
```

## 3 Results

The file containing the reaction times was used and their mean and standard deviation were computed using `numpy` from the extracted data. The results can be seen in table 2. These results can also be visualised using `matplotlib` and can be seen in figure 2. A density plot was also created, and is shown in figure 3.

Table 2: Mean and standard deviation of the three conditions for color naming and word reading

| | Word Reading | | | Color Naming | | |
|---|---|---|---|---|---|---|
| | Congruent | Control | Incongruent | Congruent | Control | Incongruent |
| Mean | 493.3 | 512.47 | 520.13 | 626.59 | 647.263 | 840.04 |
| Std | 25.10 | 30.58 | 36.21 | 27.09 | 25.08 | 29.83 |

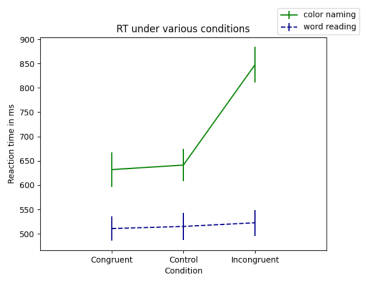

Figure 2: Simulation of the reaction times of the Stroop Task

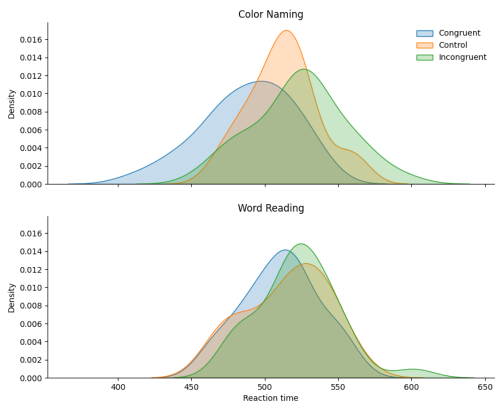

Figure 3: Probability density function of the simulation

## 4 Discussion

For analyzing the results, the original data by Dunbar et al. is used and compared to figure 2 [**dunbar1984horse**]. The figure containing the original data can be seen in figure 4.

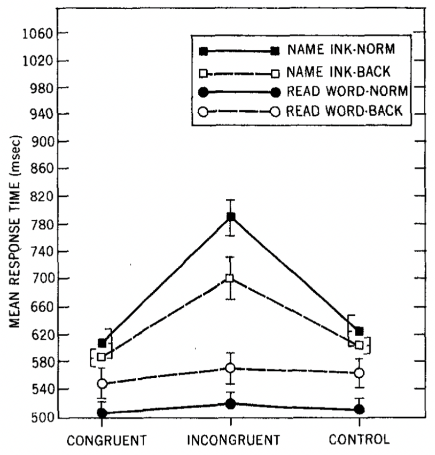

Figure 4: Simulation of the reaction times of the Stroop Task

When analyzing the results, there are a few main aspects that can be observed, which correspond to the original Stroop-effect findings by Stroop (1935) [**stroop1935studies**]. These aspects are a threefold.

***Color naming takes longer than word reading*** As Stroop observed and what can be seen from figure 2, is that the reaction times of color naming is significantly higher than that of word reading. Moreover, this can be concluded when examining 2. Research has shown that this observation can be explained by several theories, such as the speed of processing theory, [**mcmahon2013**] the selective attention theory, [**mcmahon2013**] the automaticity theory [**monahan2001coloring**] or by parallel distributed processing [**cohen1990control**].

***Incongruent vs congruent*** As also can be seen from the results, in particular figure 2, is that the naming or reading of incongruent stimuli takes longer than that of congruent stimuli. This is one of the most significant findings in cognitive psychology and it can also be seen in this simulation.

***The control group*** The control group shows the mean reaction times of either color naming or word reading with neither a congruent nor an incongruent aspect. What is interesting to see in figure 2 is that in both tasks, the congruent condition has a lower reaction time than the control group. This indicates that not only the incongruent conditions lower the reaction time, but the congruent conditions increase the reaction time.

As this research is considered a starting point for an implementation using the ccm-suite package and only hard codes the latencies, the limitations of the research are the aforementioned hard coded parameter. Additionally, the standard deviation was mostly hard coded from a random normal distribution, which can be derived from the function in figure 3.

## 5 Conclusion

The aim of this paper was to create a simulation of the Stroop-effect in ACT-R using the CCM-Suite package of Python. There were several Stroop simulations using the ACT-R architecture, however, ccm-suite has so far not been used widely for the stroop-task. An agent and environment was used to simulate the Stroop-task, using a declarative memory to account for differences in reaction times. The latencies, calculated using known research, created a variance in the different conditions, congruent, control and incongruent. For the two tasks color naming and word reading, the data was created and used to create the plots the original study also showed. These were compared and the main findings could also be found in these results, namely that color naming takes longer than word reading. Furthermore, incongruent conditions have a higher reaction time than congruent conditions and the control group has a lower reaction time than the congruent condition. These findings were also published in the literature, concluding this implementation to be a succesful simulation of the Stroop-effect.

Future work may suggest implementing an agent that could 'think' like the human brain does when performing the stroop task and may concentrate on overcoming the limitation presented in the discussion

## Appendix

Link to GitHub repository containing the code: `https://github.com/s-eeckhout/cm_final.git`