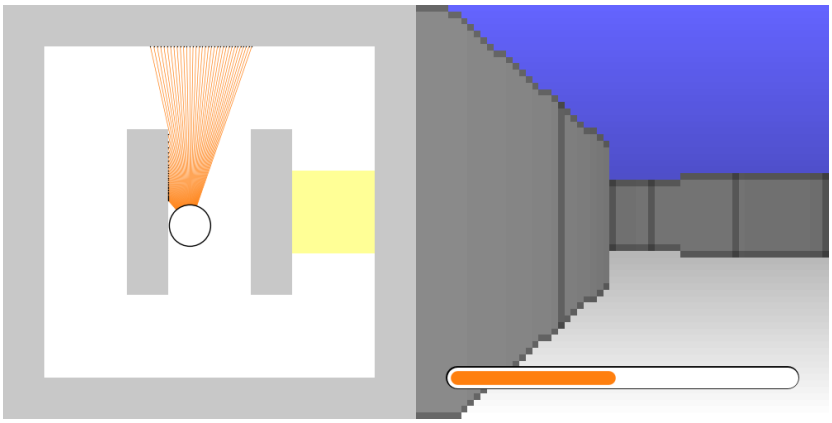
1 Programmatoid braincraft challenge solution

Let us consider the following digital experimental setup, as described in braincraft challenge presentation[1].

1.1 Simplified problem statement



The braincraft challenge[2] bot moves at a constant with sensor inputs and one orientation output, it uses some energy and refill this energy on a given yellow location. The 2D space size is $[0, 1] \times [0, 1]$. The bot starts in the middle and oriented at 90, i.e., upward.

Input variables	
$p_l \in [0_{\text{wall-hit}}, 1_{\text{no-wall}}[$	Leftward proximity, max value of the [0, +30°] range 32 left sensors.
$p_r \in [0_{\text{wall-hit}}, 1_{\text{no-wall}}[$	Rightward proximity, max value of the [-30°, 0] range 32 right sensors.
$c_{l\bullet} \in \{0, 1\}, \bullet \in \{b_{\text{blue}}, r_{\text{red}}\}$	Leftward binary red and blue color detectors.
$c_{r\bullet} \in \{0, 1\}, \bullet \in \{b_{\text{blue}}, r_{\text{red}}\}$	Rightward binary red and blue color detectors.
$g_e \in [0_{\text{death}}, 1_{\text{full}}]$	Energy gauge value.
Output variable	
$d_o \in [-5, 5]$	Orientation difference, saturated at ± 5°.

With respect to the original braincraft challenge:
- we consider two leftward and rightward "average" sensors only,
- color is input as binary variables, and always available,
- the wall hit indicator is not used.

[1] `https://github.com/rougier/braincraft/blob/master/README.md#introduction`
[2] `https://github.com/rougier/braincraft`

1.2 A putative controller

1.2.1 Input preprocessing

1.2.2 Proximity sensors

Motivation Simplifies the left-right navigation by compacting the leftward and rightward sensors as a simple pair of input.

Implementation A simple sum or average could be used, thus using directly a linearcombination of the input in afferent units.
Then, if apropriate, the maximal and the average operators can be combined, e.g.[3]:

$$\begin{array}{rcl} p_\dagger & \leftarrow & \log\left(\sum_{k\in K} \exp(\mu\, p[k])\right)/\mu \\ & = & \frac{1}{K}\sum_k p_k + \log(K)/\mu + O(\mu) \\ & = & \max_k(p[k]) + o\left(\frac{1}{\mu}\right) \end{array}$$

where K stands for the left or right sensor related indexes, $p[k]$ stands for the sensor proximity value, and $\mu > 0$ parameterizes the balance between the max (for large μ) and the average (for small μ) operators, as shown in Fig. 1.

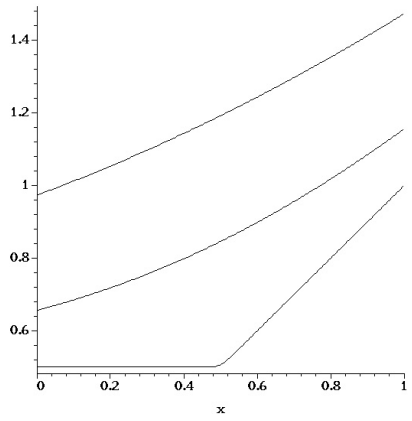


Figure 1: Representation of the exp-log function for $K = 2$, $p[2] = 1/2$, with $p[1] \in [0, 1]$ and $\mu \in \{1, 2, 100\}$, from top to bottom. With small μ it is closed to linear average, whereas for $\mu = 100$ it is close to $\max(x, 1/2)$.

[3] Let us derive the formula:
- The series at $\mu \to 0^+$ is easily obtained from any symbolic calculator writing, e.g.:

```
series(log(sum(exp(mu * p[k]), k = 1..K)) / mu, mu = 0, 2);
```

- There is no obvious series development at $\mu \to +\infty$ but, considering $p[1] \leq p[2] \leq \cdots \leq p[K]$, without loss of generality, i.e., that -for the notation- index's order correspond to decreasing values, we obtain from straightforward algebra:

$$\begin{array}{c} \log\left(\sum_{k\in\{1\cdots K\}} \exp(\mu\, p[k])\right)/\mu = p[1] + \rho/\mu \\ \rho \stackrel{\text{def}}{=} \log\left(1 + \sum_{k\in\{2\cdots K\}} \exp(-\mu\,(p[1] - p[k]))\right) \\ \text{with } 0 \leq \rho \leq \log\left(1 + (K-1)\exp(-\mu\,(p[1] - p[2]))\right) \leq \log(K) \end{array}$$

so that $\lim_{\mu\to+\infty} \rho = 0$ and $\rho = o(\mu)$, yielding the expected result.

1.2.3 Color sensors

Motivations Again, color blob detection is to perform either on the left or on the right, allowing the color input to be compacted. For each color, a channel is specified, simplifying the programmatoid implementation and providing an input closer to biological colored vision. Since the setup color input is a discrete color index, the channel value is binary, accounting for the presence, or not, of a least one related color index.

Implementation For the distributed implementation, each camera color index value is mapped on each color channel input with the 0 value if the color is different and the 1 value if equal, e.g., using step unit:

$$\begin{aligned} c_{\dagger\bullet} &\leftarrow H(\textstyle\sum_{k\in K}(1-D(i_\bullet - c[k]))),\\ &\quad \dagger \in \{l_{\text{left}}, r_{\text{right}}\}, \bullet \in \{b_{\text{blue}}, r_{\text{red}}\}\\ D(x) &\stackrel{\text{def}}{=} H(x-\epsilon)+H(-x-\epsilon)) = \text{ if } |x|<\epsilon \text{ then 0 else } 1, \epsilon \ll 1 \end{aligned}$$

where K stands for the left or right sensor related indexes, $i_\bullet$ stands for the color index, and $c[k]$ stands for the sensor color index value.

1.2.4 Energy measurement

Motivation The energy increase accumulation is to be pre-processed, since used in some task.

Processed Variables

$g_{c\flat} \in [0,1], \flat \in \{1,2\}$ $\quad g_{c\flat}|_{t=0} = 0$ Last and last-before-last cumulative energy increases.
$g_{e\flat} \in [0,1], \flat \in \{1,2\}$ $\quad g_{e\flat}|_{t=0} = 0$ Last and last-before-last energy value.

Here instantaneous energy increase is $(g_e - g_{e1})$, and we assume that the energy always changes so that $g_e \neq g_{e1}$.

Implementation

Cumulating energy increase starts, saving last increase in g_{c2}.

if $\underbrace{g_e > g_{e1} \text{ and } g_{e1} < g_{e2}}$ then $g_{c2} \leftarrow g_{c1},\ g_{c1} \leftarrow (g_e - g_{e1})$

increase after a decrease

Cumulating energy increase continues.

if $\underbrace{g_e > g_{e1} \text{ and } g_{e1} > g_{e2}}$ then $g_{c1} \leftarrow g_{c1} + (g_e - g_{e1})$

increase after an increase

Otherwise $g_e < g_{e1}$, thus cumulating energy increase stops, and g_{c1} is memorized.

Pseudo programmatoid solution

$$\begin{aligned} g_{c1} &= g_{c1} - H(g_{e2} - g_{e1})\, g_{c1} + H(g_e - g_{e1})\,(g_e - g_{e1})\\ g_{c2} &= g_{c2} + H(H(g_e - g_{e1}) + H(g_{e2} - g_{e1}) - 3/2)\,(g_{c1} - g_{c2}) \end{aligned}$$

which is not a pure programmatoid solution because, because of term of the form $H(u)\,v$, thus with a product, but is implementable in the neuronoid framework discussed in the sequel. In brief:

$$H(x)\,y \simeq h(y/\upsilon + \upsilon\, h(\upsilon\, x) - \upsilon)$$

where $h(\cdot)$ is the sigmoid function, and υ a sufficiently large number.

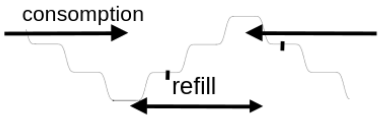


Figure 2: Representation of energy profile when cumulating energy increase starts (first tick), with $g_e > g_{e1} < g_{e2}$, and stops (last tick), with $g_e < g_{e1}$, while during refill $g_e > g_{e1} > g_{e2}$.

1.2.5 Navigation

Heuristic : The bot runs at constant velocity,

(i) ahead by default, thanks to a linear correction, high enough to correct the direction, small enough to avoid oscillations, and

(ii) attempts to perform a quarter-turn in the preferred orientation as soon as passing is detected.

With this navigation mechanism the bot performs either leftward or rightward half-loops, traversing the central corridor.

Internal variable

$q_p \in \{0_{\text{leftward}}, 1_{\text{rightward}}\}, \quad q_p|_{t=0} = 0$ Preferred quarter-turn direction.

Programmatoid solution :

$$d_o \leftarrow \underbrace{\gamma\,(p_l - p_r)}_{\substack{\text{linear correction to}\\ \text{maintain direction ahead}}} + \underbrace{\alpha\,(t_l - t_r)}_{\substack{\text{quarter- turn}\\ \text{left or right}}}$$

$$\begin{aligned} t_l &\leftarrow (1-q_p)\,H(\beta - p_l) &&= H((\beta - p_l) - \upsilon\, q_p) \\ t_r &\leftarrow q_p\,H(\beta - p_r) &&= H((\beta - p_r) - \upsilon\,(1-q_p)) \end{aligned}$$

where:

$w \simeq 1/4$ Rough estimation of the path-width.

$\gamma = \frac{5}{w/2}$ Saturates the correction at 5° if the depth difference is half of the path-width.

$\alpha = 90$ Saturates the correction at ± 90° to make the quarter-turn, since the linear $|\gamma\,(p_l - p_r)| < 40$ is lower than the quarter-turn term, the latter submut the former.

$\beta = w$ Triggers the quarter-turn if the depth is higher than the path-width.

$\upsilon = 10$ Transform a boolean product to a step-function threshold.

while:

$$\begin{aligned} t_l = 1 \quad &\text{iff} \quad q_p = 0 \quad \text{and while} \quad \beta < p_l \\ t_r = 1 \quad &\text{iff} \quad q_p = 1 \quad \text{and while} \quad \beta < p_r \end{aligned}$$

in words: we execute the quater-turn until another wall is detected.

Regarding transforming a boolean product to a step-function threshold, we easily verify[4] that:

$$b \in \{0,1\}, x \in]-M, M[\Rightarrow b\, H(x) = H(x - (1-b)\, M),$$

for both values of b, while $\max(|\beta - p_l|, |\beta - p_r) \leq 1$; this is developed and generalized in the sequel.

We thus have a linear output unit for d_o and two step-unit for t_l and t_r.

1.2.6 Direction choices

1.2.7 Task 1: Simple decision

Strategy Restrict navigation to the half-loop that contains the energy source, while the other does not.

Heuristic If the energy is too low, thus looping in the wrong direction, the direction is changed once.

At start $q_p = 0$. Then, if the energy is too low it changes once to $q_p = 1$.

$$q_p = \text{if } q_p = 1 \text{ or } \eta > g_e \text{then1 else 0}$$

Programmatoid solution

$$q_p \quad \leftarrow \quad H(\upsilon\, q_p + (\eta - g_e))$$

where:

c	$=$	$1/1000$	Energy consumption at each step.
s	$=$	$1/100$	Speed: location increment at each steps.
b	$=$	$3/2$	Distance bound between the starting point and the putative energy sources.
η	$\simeq$	$b\, c/s = 3/20$	Energy consumption threshold if no source on the path.

1.2.8 Task 1b: Simple decision but varying environment

Strategy Restrict navigation to the half-loop that contains the energy source, while the other does not, this may change with time.

Heuristic

- If the energy is too low, the direction is inverted.
-This is registered, avoiding multiple changes at low energy.
-When the energy is high enough, change registration is reset.

Internal variable

$g_c \in \{0,1\}$ $\quad g_c|_{t=0} = 0$ Registers if the low energy has been detected.

[4] If $b = 0$ then the equality writes $0 = H(x - M)$, but $x < M$ so that $H(x - M) = 0$, thus verified, while if $b = 1$ then the equality writes $H(x) = H(x)$, again verified.

Programmatoid solution

Inverts the direction if to be changed.

$q_p \leftarrow$ if $\eta > g_e$ and $g_c = 0$ then $1 - q_p$ else q_p

Registers the inversion until the energy is high enough.

$g_c \leftarrow$ if $g_c = 0$ and $\eta > g_e$ then 1 elif $g_c = 1$ and $2\,\eta < g_e$ then 0 else g_c

In the sequel, we are going to derive a generic way to compile such an expression with binary values as a programmatoid .

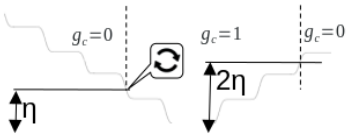


Figure 3: Representation of g_c value at different level of energy with the indication of the direction change.

1.2.9 Task 2: Cued environment decision

Strategy Restrict navigation to the half-loop without a closed path, as indicated by a color that has already been seen once.

Heuristic Detect and store the first color blob, and choose to turn in the direction it appears again.
The detection is reset when the energy decreases.

Internal variable

$c_{c\bullet} \in \{0,1\}, \bullet \in \{b_{\text{blue}}, r_{\text{red}}\}$ $\quad c_{p\bullet}i|_{t=0} = 0$ Detected the color cue code, if any.

Programmatoid solution

Detect the cue, if not yet done, and reset below an energy threshold

$c_{cb} \leftarrow$ if $c_{cb} = 0$ and $c_{cr} = 0$ and ($c_{lb} = 1$ or $c_{rb} = 1$) then1 elif $g_e < \eta/2$ then 0 else c_{cb}

$c_{cr} \leftarrow$ if $c_{cb} = 0$ and $c_{cr} = 0$ and ($c_{lr} = 1$ or $c_{lr} = 1$) then1 elif $g_e < \eta/2$ then 0 else c_{cr}

Set the direction according to the cue

$q_c \leftarrow$ if $c_{lb} = c_{cb}$ or $c_{lr} = c_{cr}$ then 0 elif $c_{rb} = c_{cb}$ or $c_{rr} = c_{cr}$ then 1 else q_c

1.2.10 Task 3: Cued environment decision

Strategy and Heuristic Test energy sources and change direction if the latter yields less increse than the former.

Internal variable

Programmatoid solution

Programmatoid solution :

$$\begin{aligned}
\lambda &= \lambda + \Delta\lambda_1 + \Delta\lambda_2 + \Delta\lambda_3 + \cdots \\
\Delta\lambda_1 &= (\lambda - 1)\, H(\alpha - \varepsilon) \\
\Delta\lambda_2 &= (\lambda - 1)\, H(\iota - I_{\text{left blue color sensor}}) \\
\Delta\lambda_3 &= (\lambda - 1)\, (\Upsilon_{\text{again red color}} + \Upsilon_{\text{again yellow color}} + \cdots) \\
&\cdots \\
\Upsilon_{\text{again this color}} &= \Upsilon_{\text{seen this color}}\, H(\iota - I_{\text{left this color sensor}}) \\
\Upsilon_{\text{seen this color}} &= \Upsilon_{\text{seen this color}} + (1 - \Upsilon_{\text{seen this color}})\, H(\iota - I_{\text{this color sensor}}) \\
&\cdots
\end{aligned}$$

where:

α is a the energy threshold, corresponding to the energy consumption during one turn.

ι is some color detection threshold.

$\Delta\lambda_1$ raises to one if the energy decreases below a threshold.

$\Delta\lambda_2$ raises to one if the blue color is seen on the left.

$\Delta\lambda_3$ raises to one if some color is seen again.

$\Upsilon_{\text{again this color}}$ raises to one a previously seen color is seen again on the left.

$\Upsilon_{\text{seen this color}}$ raises to one a color is seen for the first time.

$I_{\text{left this color sensor}}$ combines color sensor input.

$I_{\text{this color sensor}}$ combines left and right color sensor input.

Neuronoid implementation : The mollification of this system uses 3 neuronoids and writes:

$$\begin{aligned}
d\theta &= h\left(\gamma\,(\theta_l - \theta_r) + \tfrac{\pi}{2}\,(\Delta\theta_r - \Delta\theta_l)\right) \\
\Delta\theta_r &= h(W_\infty\,(\beta - \theta_r) + 2\,W_\infty\,(\lambda - 1)) \\
\Delta\theta_l &= h(W_\infty\,(\beta - \theta_l) - 2\,W_\infty\,\lambda)
\end{aligned}$$

as easily verified considering the four cases $\beta \lessgtr \theta_*$ versus $\lambda \in \{0, 1\}$.

With respect to the programmatoid solution, the output value is "saturated" by the $h(\cdot)$ function while the $\Delta\theta_*$ almost binary values are approximated by the mollification of the step function. This part of the system is feed-forward thus without convergence or stability issue.

1.2.11 Computation unit: neuronoid

By "neuronoid" we name the not very biologically plausible[5] simplest biological neuron or neuron small ensemble inspired by mean-field modelisation[6] of the Hodgkin–Huxley neuronal axon model[7].

$$\tau \frac{\partial v_i}{\partial t}(t) + v_i(t) = z_i(t), \quad z_i(t) \stackrel{\text{def}}{=} h\left(\sum_j w_{ij}\, v_j(t) + w_i\right),$$
$$h(v) \stackrel{\text{def}}{=} \frac{1}{1+\exp(-4\,v)}, \qquad H(v) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if} \quad v > 0 \\ 0 & \text{if} \quad v < 0 \end{cases}$$

where v_i is the membrane potential, so that:

$$\begin{aligned} v(t) &= 1/\tau \int_0^t z(s)\, exp(-(t-s)/\tau)\, ds + v(0)\, exp(-t/\tau) \\ &= z(0) + (v(0) - z(0))\, e^{-t/\tau}\big|_{z(t)=z(0)}\,. \\ &= z(t)|_{\tau=0} \\ &\simeq (1-\gamma)\, v(t-\Delta t) + \gamma\, z(t-\Delta t)) \\ &= \sum_{s=0}^{t-1} z(s)\, \gamma\, (1-\gamma)^{t-s+1}\, z(s) + v(0)\, (1-\gamma)^t \\ &= z(0) + (v(0) - z(0))\, (1-\gamma)^t|_{z(t)=z(0)} \\ &= z(t)|_{\gamma=1}\,. \end{aligned}$$

writing also the corresponding discrete approximation using an Euler schema with $0 < \gamma < 1, 0 < \tau$:

$$\gamma \stackrel{\text{def}}{=} 1 - \exp(-1/\tau) \Leftrightarrow \tau = 1/\log(1/(1-\gamma))$$
$$\lim_{\gamma \to 0} \tau = +\infty, \lim_{\gamma \to 1} \tau = 0.$$

Here, $h(\cdot)$ is the normalized sigmoid with $h(-\infty) = 0, h(0) = 1/2, h'(0) = 1, h(+\infty) = 1$, linearly related to the hyperbolic function:

$$h(v) = \frac{1+\tanh(2\,v)}{2}.$$

It is the mollification of the step function, called also Heaviside function, $H(\cdot)$, as detailed below.

It is thus a very common 1st order "neuronoid" model, but with an adjustable bias (or offset) w_i.

1.3 Programmatoid and neuronoid computation

1.3.1 Programmatoid computation

We name "programmatoid" computation the conception of an input-output straight-line program[8] implementing test operator on numerical value expressions using the step function, considering 1 as the true value and 0 as the false value. The $H(v)$ implements the test of the positive sign of v, while for $v_i \in \{0, 1\}$:

[5] https://en.wikipedia.org/wiki/Biological_neuron_model#Relation_between_artificial_and_biological_neuron_models

[6] https://inria.hal.science/cel-01095603v1

[7] https://en.wikipedia.org/wiki/Hodgkin-Huxley_model

[8] https://en.wikipedia.org/wiki/Straight-line_program

programmatoid conjunction The $v_0 = H(v_1 + v_2 + \cdots)$ formula performs a *or* operation.

programmatoid disjunction The $v_0 = v_1\, v_2 \cdots$ formula performs a *and* operation.

programmatoid negation The $v_0 = (1 - v_1)$ formula performs a negation.

so that we can combine any boolean expression on any test of value sign, thus value comparison, and value interval inclusion, switches between two expressions, etc. This defines real semi-algebraic[9] sets of degree 1.

Using local feedback we can also design several functions detailed in the Appendix of this section, while we also can combine neuronoid and programmatoid functions to design temporal functions.

1.3.2 Mollification of the step function

The step function is related to a conditional expression by a simple relation:

$$H(v) = \textit{conditional value} > 0\ ?\ 1 : \textit{conditional value} < 0\ ?\ 0 :\ H(0),$$

where $\textit{condition}\ ?\ \textit{value if true} : \textit{value if false}$ is a conditional expression.

The step function approximates sigmoid with huge slope at zero, i.e.:

$$\forall v \neq 0, H(v) = \lim_{W_\infty \to +\infty} h(W_\infty\, v),\ h'(W_\infty\, v)|_{v=0} = W_\infty$$

while the convergence is also obtained for $v = 0$ in the distribution sense with $H(0) = h(0) = 1/2$. More precisely:

$$|H(\cdot) - h(\cdot)|_{\mathcal{L}_1} = \frac{\log(2)}{2}\, \frac{1}{W_\infty}$$

and, for $0 < \epsilon_\infty \ll 1 < v_\infty$:

$$h(W_\infty\, v_\infty) = 1 - \epsilon_\infty \Leftrightarrow v_\infty = \frac{\log(1-\epsilon_\infty) - \log(\epsilon_\infty)}{4\, W_\infty} = \frac{-\log(\epsilon_\infty)}{4\, W_\infty} + O\left(\epsilon_\infty\right).$$

Numerically, the convergence is very fast, e.g. $W_\infty = 2$, for $\epsilon_\infty = 0.1\%$:

ϵ_∞	10^{-1}	10^{-2}	10^{-3}	10^{-6}
W_∞	0.55	1.15	1.73	3.46

A step further, the local variation of sigmoid writes:

$$\begin{aligned} h(v) &= h(v_0) + \left[4\, \exp(-8\, v_0) + O\left(\exp(-12\, v_0)\right)\right]\, (v - v_0) + O\left((v - v_0)^2\right) \\ &\simeq h(v_0) + 4\, \exp(-8\, v_0)\, (v - v_0). \end{aligned}$$

Numerically, values decrease very rapidly:

v_0	1	2	5	10
$4\, \exp(-8\, v_0) \simeq$	10^{-1}	10^{-3}	10^{-8}	10^{-17}

1.3.3 Neuronoid implementation of programmatoid computation

We call "neuronoid" computation the conception of an input-output transform based on feed-forward and recurrent combination of neuronoids, as defined previously.

[9] `https://en.wikipedia.org/wiki/Real_algebraic_geometry`

By successive combination, any programmatoid computation involving the $H(\cdot)$ function can be approximated by a neuronoid, with $\tau \simeq 0$.

A step ahead, we considering neuronoid with $\tau > 0$ it seems obvious that we can designed temporizing mechanisms, oscillators and sequence generator, sleep sort mechanism, etc. (not detailed here because not used at this stage, see appendix).

Neuronoid implementation : The mollification uses 3 neuronoids for cases 1 and 2 plus 3 neuronoids by color for case 3. The derivation of the neuronoid equations is straightforward after the previous one.

1.4 Appendix: a few programmatoid and neuronoid components

1.4.1 Conditional expression

For two inputs $v_{i1}(t) \in \{0,1\}, v_{i0}(t) \in \{0,1\}$, an output $v_o(t) \in \{0,1\}$ and a control $v_l \in \{0,1\}$, the condition expression equation, for $0 < W_\delta \ll 1 < W_\sigma$:

$$
\begin{aligned}
v_o(t+1) &= v_l(t) == 1 \ ? \ v_{i1}(t) : v_{i0}(t) && (1)\\
&= v_l(t)\, v_{i1}(t) + (1 - v_l(t))\, v_{i0}(t) && (2)\\
&= H(v_{i1}(t) - W_\sigma\,(1 - v_l(t)) - W_\delta) + H(v_{i0}(t) - W_\sigma\, v_l(t) - W_\delta) && (3)\\
&\simeq h(W'_\infty\,(W_\omega\, v_{i1}(t) - W_\sigma\,(1 - v_l(t)) - W_\delta))\\
&+ h(W'_\infty\,(W_\omega\, v_{i0}(t) - W_\sigma\, v_l(t) - W_\delta)). && (4)
\end{aligned}
$$

To explain these design choices, let us notice that:

- The W_σ value is used at the programmatoid level to ensure that given the $v_l(t)$ binary switch value, it constraints the step function output to correspond to the desired value. Here, an expression including a product by a binary function, is replaces by a sum. The rationale is that there is no explicit multiplication between two variables at the neuronoid level. This trick allows one a straightforward neuronoid approximation.

- The W_δ value is used at the programmatoid level to avoid the ambiguous 0 value and ensure that for $v \simeq 0$ we obtain $H(v - W_\delta) = 0$.

- The W'_∞ gain is used to approximate the step function by a sigmoid, as discussed previously.

- Informally, at the neuronoid level, we mimic the programmatoid mechanisms, assuming that suitable $W'_\infty, W_\omega, W_\sigma, W_\delta$ values will reproduced the programmatoid conditional expression. It works, although the parameter adjustment is not obvious, as derived now.

Let us now verify the equivalence between these equations:

- It is obvious to verify that line (1) and (2) are equivalent.

- The fact line (2) and (3) are equivalent is verified by this truth table:

$v_l(t)$	$v_{i1}(t)$	$v_{i0}(t)$	$v_{i1}(t) - W_\sigma\,(1 - v_l(t)) - W_\delta$	$v_{i0}(t) - W_\sigma\,v_l(t) - W_\delta$	$v_o(t+1)$
1	0	0	$-W_\delta < 0$	$-W_\sigma - W_\delta < 0$	$0 + 0 = 0 = v_{i1}(t)$
1	0	1	$-W_\delta < 0$	$1 - W_\sigma - W_\delta < 0$	$0 + 0 = 0 = v_{i1}(t)$
1	1	0	$1 - W_\delta > 0$	$-W_\sigma - W_\delta < 0$	$1 + 0 = 1 = v_{i1}(t)$
1	1	1	$1 - W_\delta > 0$	$1 - W_\sigma - W_\delta < 0$	$1 + 0 = 1 = v_{i1}(t)$
0	0	0	$-W_\sigma - W_\delta < 0$	$-W_\delta < 0$	$0 + 0 = 0 = v_{i0}(t)$
0	0	1	$-W_\sigma - W_\delta < 0$	$1 - W_\delta > 0$	$0 + 1 = 1 = v_{i0}(t)$
0	1	0	$1 - W_\sigma - W_\delta < 0$	$-W_\delta > 0$	$0 + 0 = 0 = v_{i0}(t)$
0	1	1	$1 - W_\sigma - W_\delta < 0$	$1 - W_\delta > 0$	$0 + 1 = 1 = v_{i0}(t)$

- The approximation of line (3) at line (4) by continuous quantities corresponds now to:

$$v_*(t) \in \{[0, \epsilon_\infty], [1 - \epsilon_\infty, 1]\} = \{\simeq 0, \simeq 1\}, \epsilon_\infty \ll 1/2,$$

in words: values to be either below or above a threshold close to either 0 or 1. The truth table now involves intervals and has been generated using the computer algebra piece of code associated to this document[10]. Considering the following intuitive design constraints:

$$0 < W_\delta < \{W_\omega, W_\sigma\}, 0 < W_\omega < W_\sigma, 0 < \epsilon_\infty < 1$$

the computer algebra derivations show that the approximation at line (4) is valid as soon as:

$$W_\omega\,\epsilon_\infty < W_\delta, W_\sigma\,\epsilon_\infty < W_\delta, W_\delta + (W_\omega + W_\sigma)\,\epsilon_\infty < W_\omega.$$

Furthermore, let us consider the margin:

$$0 < \mu = \min(W_\delta - \epsilon_\infty\,W_\sigma, W_\omega - (W_\omega + W_\sigma)\,\epsilon_\infty - W_\delta),$$

yielding:

$$|v_o(t+1) - 1/2| > h(W'_\infty\,\mu).$$

We thus can adjust $W'_\infty = W_\infty/\mu$ in order $v_o(t+1) \in \{[0, \epsilon_\infty], [1 - \epsilon_\infty, 1]\}$ for further calculation.

For instance $\{W_\delta = 1, W_\omega = 2, W_\sigma = 4, \epsilon_\infty = 1/8\}$ is a suitable solution with $\mu = 1/4$.

1.4.2 RS input/output gate

For an input $v_i(t) \in \{0, 1\}$, an output $v_o(t) \in \{0, 1\}$, and a control $v_l \in \{0, 1\}$, the equation:

$$\begin{aligned} v_o(t+1) &= v_l(t) == 1\ ?\ v_o(t) : v_i(t), \\ &= v_l(t)\,v_o(t) + (1 - v_l(t))\,v_i(t) \\ &= H(v_o(t) - W_\sigma\,(1 - v_l(t)) - W_\delta) + H(v_i(t) - W_\sigma\,v_l(t) - W_\delta) \\ &\simeq h(W'_\infty\,(W_\omega\,v_o(t) - W_\sigma\,(1 - v_l(t)) - W_\delta)) \\ &+ h(W'_\infty\,(W_\omega\,v_i(t) - W_\sigma\,v_l(t) - W_\delta)). \end{aligned}$$

implements a 1 bit memory, i.e., also called a RS gate, and reusing the previous conditional instruction parameters.

[10] `https://github.com/vthierry/braincraft/raw/master/data/programmatic-solution.mpl.txt.out`

The key point is that it is now a recurrent system, which stability is obvious at the programmatoid level, but not necessarily at the neuronoid level, since we have a continous equation. More precisely:

$$v_o(t+1) = h(W'_\infty\, W_\omega\, v_o(t) - W_\beta(t)) + h_\beta(t),$$
$$\begin{cases} W_\beta & \stackrel{\text{def}}{=} & W_\sigma\,(1 - v_l(t)) + W_\delta \\ h_\beta(t) & \stackrel{\text{def}}{=} & h(W'_\infty\,(W_\omega\, v_i(t) - W_\sigma\, v_l(t) - W_\delta)) \in [-1, 1] \end{cases}$$

with a non contracting recurrent function, since:

$$h'(W'_\infty\, W_\omega\, v) > 1 \text{ for } |v| \leq W'_\infty\, W_\omega.$$

monostable binary value The $v_0 = v_0 + (1 - v_0)\, H(v_1)$ formula sets v_0 to 1 for ever, as soon v_1 has raised once to 1.

and by combination RS gates, bistable mechanisms, etc. (not detailed here because not used at this stage, see appendix).

To be done.

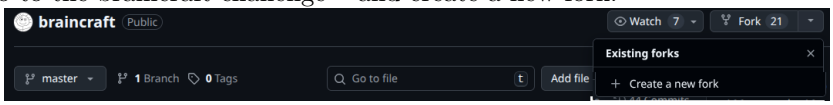
2 Using the braincraft challenge setup

Reference: The braincraft challenge[11].

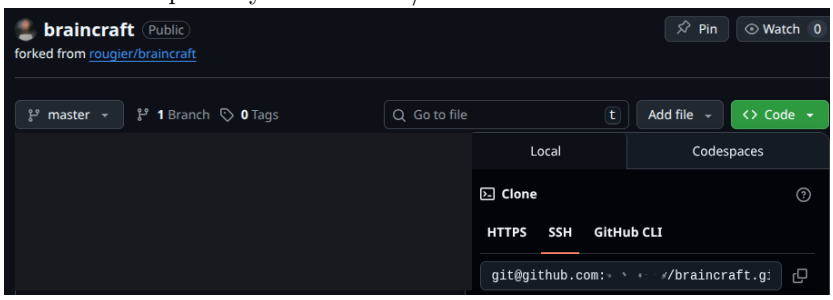
You must be familiar with basic `git` usage and basic `python` programmation.

2.1 Installation of the setup

- Connect to `https://github.com` with your login.
- Go to the braincraft challenge[12] and create a new fork:



- Download the repository in SSH read/write mode:



-In the braincraft local git directory, run `make test`

[11] `https://github.com/rougier/braincraft`
[12] `https://github.com/rougier/braincraft`

+ You may have to run `make install`, before.
+ You are advised to use a virtual environment, running `make venv`

2.2 Running at the programmatic level

API doc[13]

The 'challenge_callback_1.py' file contains the support routines

2.3 Running at the artificial neural network level

```
* Note : vthierry veut PAS gagner la compétition is veut just vérifier des hypothèse quant l

- Warningup : duration (bot don't move before warmup period is over) just 1 to allow a 1st i

- Quelles dimensions pour Win (quelles entrées où ? le truc de dimension P), W, et Wout(lign

- Avis sur calcul de depth et couleur ?
 + depth: prendre le min des capteurs de gauche/droite ou vaut mieux leur moyenne pondérée ?
 + couleur: les murs ont le vert comme couleur par defaut ? prendre la valeur de couleur la

- Pour expérimenter l'approche programmatique avant de passer à l'approche connexiviste :
 - comment ''débrancher´´ le réseau et just avoir (P inputs) -> output ? sans reimplementer

- Pour expérimenter l'approche connectiviste sans apprentissage ... juste implémenter def tr
```

[13] https://raw.githack.com/vthierry/braincraft/master/braincraft/doc/challenge_callback.html