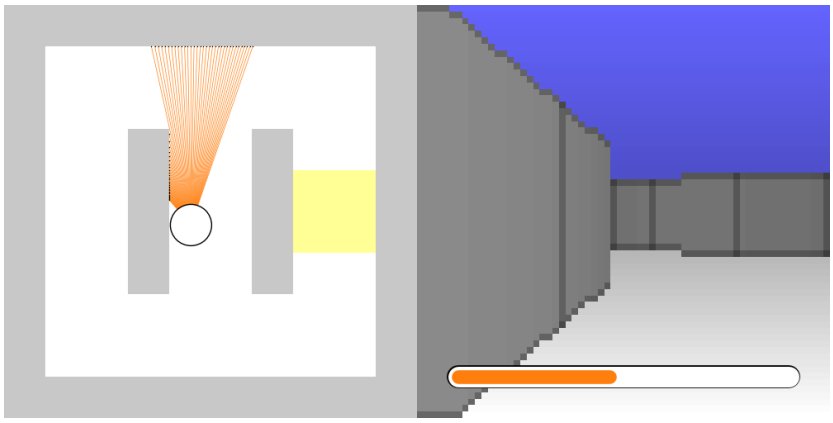
Simplified problem statement



Reference: The braincraft challenge[1].

Variables

Input θ_l Average of the *left* sensors input, between 0 and 1 when very close, 30 sensor field.

θ_r Average of the *right* sensors input, idem.

ε Energy indicator, between 1 when full and 0 when dead.

Output $d\theta$ Orientation relative increment, between -1 for ??? leftward and $-$ for ??? rightward.

Internal λ Orientation preference 0 if left, 1 if right, $\lambda = 0$ at start.

Computation unit: neuronoid

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

where v_i is the membrane potential, and $h(\cdot)$ is the normalized sigmoid with $h(-infty) = 0, h(0) = 1/2, h'(0) = 1, h(+\infty) = 1$, which is the mollification of the Heaviside function $H(\cdot)$, so that

$$v(t) = 1/\tau \int_0^t z(s)\, exp(-(t-s)/\tau)\, ds + v(0)\, exp(-t/\tau) = z(t)|_{\tau=0}\,.$$

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

[1] `https://github.com/rougier/braincraft`

Programmatoid and neuronoid computation

Programmatoid computation

Here we call "programmatoid" computation the conception of an input-output straight-line program[2] implementing test operator on numerical value expressions using the Heaviside function, considering 1 as the true value and 0 as the false value. With the convention $H(v)$ implements the test of the positive sign of v, while

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

programmatoid disjuction The $v_0 = H(v_1)\, H(v_2) \cdots)$ formula perfoms a *and* operation.

programmatoid negation The $v_0 = H(1 - H(v_1))$ 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.

Using local feedback we can also design:

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).

Mollification of the Heaviside function

The Heaviside function is thus a functional implementation of the sign test of a value, and $H(v) \simeq h(W_\infty\, v), W_\infty \to +\infty$, i.e.:

$$|H(\cdot) - h(\cdot)|_{\mathcal{L}_1} = O\left(\tfrac{1}{W_\infty}\right), \;\; h(W_\infty\, v_\infty) = 1 - \epsilon_\infty \Leftrightarrow v_\infty = \tfrac{-\log(\epsilon_\infty)}{4\, W_\infty} + O\left(\epsilon_\infty\right).$$

This allows one to approximate a construct of the form:

$$\lambda\, H(v) \simeq h(v + W_\infty\, (\lambda - 1)), \text{ with } \lambda \in \{0, 1\} \text{ and } v \ll W_\infty,$$

in words: it approximates a binary switch $\lambda \in \{0, 1\}$ prefixing a sign test by a computation unit, since for $\lambda = 1$ it equals $H(v)$ on both sides and if $\lambda = 0$ it almost equals 0on both sides.

Neuronoid computation

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

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

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).

A putative controller

Navigation

Heuristic: The bot runs at constant velocity, (i) ahead by default and (ii) if passing in the preferred orientation is possible, makes a quarter turn.

Programmatoid solution:

$$\begin{aligned} d\theta &= \underbrace{\gamma\,(\theta_l - \theta_r)}_{\substack{\text{linear correction to} \\ \text{maintain direction ahead}}} + \underbrace{\frac{\pi}{2}\,(\Delta\theta_r - \Delta\theta_l)}_{\text{direction change}} \\ \Delta\theta_r &= \lambda\, H(\beta - \theta_r) \\ \Delta\theta_l &= (1-\lambda)\, H(\beta - \theta_l) \end{aligned}$$

where:

$\Delta\theta_*$ raises from 0 to 1 when the left sensor detects a passing, i.e., the fact tat the side wall is not close anymore.

γ is a feedback loop gain $0 < \gamma < 1$ to be adjusted high enough to correct the direction, small enough to avoid oscillations.

β is a threshold below which the side sensor input corresponds to no side wall but a passing.

in words: the bot navigates ahead thanks to the linear correction parameterized by γ and perform a quarter turn in the preferred direction as soon a passing is detected.

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

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

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 $\Delta\theta_*$ almost binary value is approximated by the mollification for the previous subsection.

Direction choice

Heuristic: The initial direction is right, but as soon as an input contradicts this assumption, it is turned left once, and this remains.

Case 1 If the energy is to low, it means we turn in the wrong direction, so it changes.

Case 2 If there is a blue color on the left it means we must change from right to left.

Case 3 If there is a red (or yellow, etc) color somewhere, then change the turn direction if i see it again on the left.

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 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.

Appendix: a few programmatoid and neuronoid components

To be done.