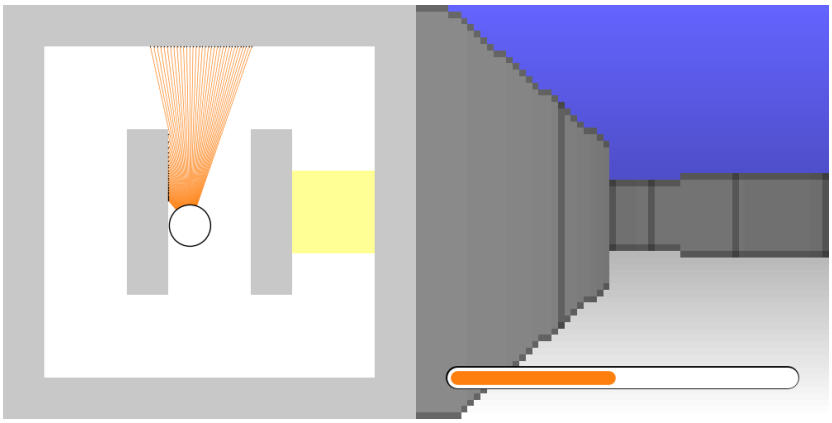
1 Programmatoid braincraft challenge solution

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

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.

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:

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

[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 $p[1] \le p[2] \le \cdots \le p[K]$, without loss of generality, i.e., that -for the notation- index's order correspond to decreasing values, we obtain from straightforward algebra:

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

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

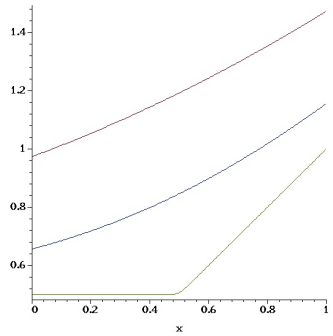


Figure 1: Representation of the exp-log function for $K = 2$, $p[2] = 1/2$, with $p[1] \in [0, 1]$ and $\mu \in \{1, 2, 100\}$. With small μ it is closed to linear average, whereas for $\mu = 100$ it is close to $\max(x, 1/2)$.

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.

Energy measurement

Motivation The energy increase accumulation has 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 $(g_e - g_{e1})$,

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 $g_e \leq g_{e1}$, 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:

$$H(u)\,v \simeq h(v/\upsilon + \upsilon\,h(\upsilon\,u) - \upsilon)$$

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

1.2.2 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 :

$$\begin{array}{rcl} d_o & \leftarrow & \underbrace{\gamma\,(p_l - p_r)}_{\substack{\text{linear correction to}\\ \text{maintain direction ahead}}} + \underbrace{\alpha\,(t_l - t_r)}_{\substack{\text{quarter- turn}\\ \text{left or right}}} \\ t_l & \leftarrow & (1-q_p)\,H(\beta - p_l) = H((\beta - p_l) - \upsilon\,q_p) \\ t_r & \leftarrow & q_p\,H(\beta - p_r) = H((\beta - p_r) - \upsilon\,(1-q_p)) \end{array}$$

where:

w	$\simeq$	$1/4$	Rough estimation of the path-width.
γ	$=$	$\frac{5}{w/2}$	Saturates the correction at 5° if the depth difference is half of the path-width.
α	$=$	90	Saturates the correction at ± 90° to make the quarter-turn, since the linear $\lvert\gamma\,(p_l - p_r)\rvert < 40$ is lower than the quarter-turn term, the latter submut the former.
β	$=$	w	Triggers the quarter-turn if the depth is higher than the path-width.
υ	$=$	10	Transform a boolean product to a step-function threshold.

while:

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

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.

[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.

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

1.2.3 Direction choices

1.2.4 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 $q_p = 1$.

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

Programmatoid solution

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

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.
ν	$\simeq$	$b\,c/s = 3/20$	Energy consumption threshold if no source on the path.

1.2.5 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

$g_i \in \{0,1\}$	$g_i\vert_{t=0} = 0$	Detects the low energy, if not yet done.
$g_c \in \{0,1\}$	$g_c\vert_{t=0} = 0$	Registers if the low energy has been already detected.

Programmatoid solution

Detects if the direction is to be changed.

$$\begin{aligned} g_i \quad \leftarrow \quad & \text{if } g_c = 0 \text{ and } \nu > g_e \text{ then 1 else 0} && = \quad H(H(g_c - 1/2) \\ & && + \quad H(g_e - \nu) - 3/2) \end{aligned}$$

Inverts the direction if to be changed.

$$\begin{aligned} q_p \quad \leftarrow \quad & \text{if } g_i = 0 \text{ then } q_p \text{ else } 1 - q_p && = \quad H(q_p - 1/2 - \upsilon\,(1 - g_i)) \\ & && + \quad H(1/2 - q_p - \upsilon\, g_i) \end{aligned}$$

Registers the inversion until the energy is high enough.

$$\begin{aligned} g_c \quad \leftarrow \quad & \text{if } 2\,\nu < g_e \text{ then 0} \\ & \text{elif } g_c = 1 \text{ then 1 else } g_i && = \quad H(g_i - 1/2 + \upsilon\, g_c - 2\,\upsilon\, H(g_e - 2\,\nu)) \end{aligned}$$

1.2.6 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.

Internal variable

$$c_{p\bullet} \in \{0, 1\} \quad c_{p\bullet} i|_{t=0} = 0 \quad \text{Color}$$

Programmatoid solution

Detect the cue, if not yet done

$$\begin{aligned} &\text{if} \quad c_p = 0 \text{ and } c_l \neq 0 \quad \text{then} \quad c_p \leftarrow c_l \\ &\text{if} \quad c_p = 0 \text{ and } c_r \neq 0 \quad \text{then} \quad c_p \leftarrow c_r \end{aligned}$$

Set the direction according to the cue

$$\begin{aligned} &\text{if} \quad c_p \neq 0 \text{ and } c_l \neq 0 \text{ and } c_p = c_l \quad \text{then} \quad q_c \leftarrow 0 \\ &\text{if} \quad c_p \neq 0 \text{ and } c_r \neq 0 \text{ and } c_p = c_r \quad \text{then} \quad q_c \leftarrow 1 \end{aligned}$$

Reset the cue at a certain energy decrease

$$\text{if} \quad g_e < \kappa \quad \text{then} \quad c_p \leftarrow 0$$

blablab $c_{l\bullet} \in \{0, 1\}, \bullet \in \{b_{\text{blue}}, r_{\text{red}}\}$