Replication / Computational Neuroscience

# [Re] Modeling habits as self-sustained patterns of sensorimotor behavior

Tristan Gillard[1,2], Jérémy Fix[1,3, ID], and Alain Dutech[1,4]

[1]Loria, UMR 7503, Nancy, France – [2]Université de Lorraine„ Nancy, France – [3]Centrale Supelec„ Metz, France – [4]INRIA Nancy Grand-Est, Nancy, France

## 1 Introduction

Habits, or habitual behaviors, are experiencing a resurgence of interest in cognitive science. After cognitivism put them in the background as not needing internal representations, a more recent trend is more aware of their fundamental role in explaining the cognition. With the rise of the notions of embodiment, of the importance of sensorimotor interactions and of self-organization processes in the brain, new simulation models of habits are needed.

We are particularly interested in such a model, proposed by Egbert and Barandiaran [1] which puts forward self-organization as mesoscopic processes that leads to macroscopic self-organization of behavioral habits through repetition in a continuous sensorimotor space. This model does not claim any biological plausibility and is meant more as a "logical" modelization of habits. The idea is to identify, and question, key characteristics of habits about their development, their interference, their adaptation. The model is based on a "Iterant Deformable Sensorimotor Medium" (IDSM) where past sensorimotor trajectories are more likely to attract the current trajectory, ending up in trajectories similar to the most frequent past ones. Habits, in short. In another work, Egbert and Cañamero, show how this model [2] can lead to reinforcement of habits that keeps an artificial homeostatic agent in its viability zone without the need of an explicit reinforcement signal.

Our interest lies in exploring the limits of the IDSM framework, especially regarding the bootstrapping of behaviors and how different behaviors can coexist, influence themselves and be adapted to the needs of the agent. For that, a first step has been to try to reproduce the work of [1] where basic elements and properties of the IDSM are tested. As explained in section 2, we have implemented the IDSM-based model in python. The next sections show that we indeed succeeded in replicating most of the previous results: basic influence of several nodes on the behavior (section 2.6 and 2.7); recreating a previously forced behavior (section 3); training functional habits (section 4). But it is difficult to reproduce the last part of the original paper. Section 5 shows that the peculiar behaviors produced in the original paper seem to be quite rare in our own experiments. We discuss the reasons that can explain this discrepancy before concluding our work.

## 2  Method

### 2.1  Generalities about IDSM

Egbert and Barandiaran introduced the "Iterant Deformable Sensorimotor Medium" (IDSM) as a abstract model to study the formation of habits. This model has no biological plausibility and is only a conceptual tool for discussing matters about habits, where habits are thought as a tendency to reproduce previously seen sensorimotor trajectories.
The core elements of the IDSM are sensorimotor nodes that act as a kind of memory of what the agent experienced before. A node is essentially situated in the sensorimotor space (component $N_{\boldsymbol{p}}$ of the model) but also encodes what is the "preferred" evolution of the sensorimotor trajectories in its vicinity using a velocity component $N_{\boldsymbol{v}}$. As depicted on Figure 1, at a given instant in time, the agent being in sensorimotor state $\boldsymbol{p}$, an existing node exerts two influences on this current sensorimotor state:

1. a node "pushes" the agent with a force proportionnal to the component $N_{\boldsymbol{v}}$ (force V),

2. a node "attracts" the agent to the node's position $N_{\boldsymbol{p}}$ (force A).

Thus, every existing node influences the current sensorimotor trajectory of the agent. The main idea behind this is to get the agent to try to reproduce memorized sensorimotor trajectories. This capacity is further induced as a node will be more influent the more frequently the agent "travels" near this node (by increasing the weight $N_w$ of the node). On the contrary, the weight of a node which is never "visited" will decay.
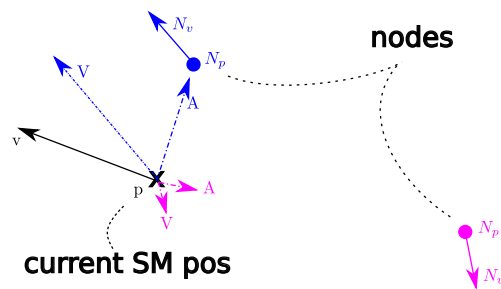


**Figure 1.** **The basics of IDSM**. Each node (blue and pink) exerts a double influence on the current sensorimotor position : an attraction $A$ toward its $N_{\boldsymbol{p}}$ component and a push $V$ along its $N_{\boldsymbol{v}}$ component. Node pink is far away, so its influence is lower.

During the initialization phase of the IDSM, the agent creates nodes along sensorimotor trajectories it experiences. There is no relation between the number of nodes and the dimensions of the sensorimotor space. But, bigger (in term of the number of dimensions) sensorimotor spaces would require a larger number of node to be paved, keeping in mind that all parts of the normalized sensorimotor space are maybe not reachable by the agent.
One of the open problems with the IDSM, outside of the scope of this paper, is thus to know what are innate behaviors (or reflexes) and the conditions that will produce these initial trajectories.

### 2.2  Model

The original model of [1] is described with text and equations in the article. We suspect that this model has been implemented and tested using the python language, but the code was not released with the publication.

We have chosen to reproduce this model and the experiments by using python (3.6.9), numpy (1.17.4) and matplotlib (3.1.1). We provide a virtual environment to make it easier to run the python code with all required dependencies (see Appendix A).

All symbols and parameters used in the various equations describing the IDSM model are listed in table 1.

The core of the IDSM model is based on "nodes" that help maintain a history of sensorimotor dynamics. More formally, a node $N$ is a tuple made of two vectors and two scalars.

$$N = \langle \boldsymbol{p}, \boldsymbol{v}, w, a \rangle \tag{1}$$

where

- $\boldsymbol{p}$ indicates the position of the node $N$, in sensorimotor space SM

- $\boldsymbol{v}$ indicates the velocity of the change in sensorimotor space SM

- $w$ is a scalar which indicates the influence of a node, as described later.

- $a$ is the age of the node.

The age $a$ of a node is not an attribute of a node in the original work, we have added it as it simplifies some explanation later in the paper (in particular in section 2.5).

It must also be noted that $\boldsymbol{p}$ and $\boldsymbol{v}$ are computed in *normalized* sensorimotor space where the range of all sensors and motors values are linearly scaled to lie in $[0, 1]$. Example of such normalization (and denormalization) can be found later in the paper, as code snippets attached to the experiments. The components of a node will be referred to using subscript notation, so the SM-position, SM-velocity, weight and age of the node $N$ are written $N_{\boldsymbol{p}}$, $N_{\boldsymbol{v}}$, $N_w$ and $N_a$.

The dynamics of the model involves first order differential equations. The original work does not specify how these are numerically integrated. In our simulations, we use forward Euler integration with a time step $dt = 0.01$.

## 2.3 Creation of nodes

As an agent controlled by the IDSM moves through sensorimotor states, new nodes are created if the current density $\phi_{\mathcal{N}}(\boldsymbol{x})$ of nodes at the normalized position $\boldsymbol{x}$ of the agent is lower than a scalar threshold $k_t$.

$$\phi_{\mathcal{N}}(\boldsymbol{x}) = \sum_{\mathcal{N}} \omega(N_w).d(N_{\boldsymbol{p}}, \boldsymbol{x}) \tag{2}$$

$$\omega(N_w) = \frac{2}{1 + \exp(-k_{\omega} N_w)} \tag{3}$$

$$d(N_{\boldsymbol{p}}, \boldsymbol{x}) = \frac{2}{1 + \exp(k_d \|N_{\boldsymbol{p}} - \boldsymbol{x}\|^2)} \tag{4}$$

In equation (2), $\mathcal{N}$ is the set of all nodes. The distance factor (equation 4) and weight factor (equation 3) are depicted on figure 2.

Concretely, in our python code, the creation of a node uses the function `create_node( self, pos, vel)` in the `class IDSM` of `IDSM.py`. The function is called in the `step()` function of the same class.

- the velocity $N_{\boldsymbol{v}}$ is computed as the difference between the previous sensorimotor position (`self.sm_pos_1`) and the current sensorimotor position (`sm_pos` argument of `step()`, see line 185 of listing 1) divided by the time step `dt` of the Euler integration. The first time `step()` is called, no node is added as `self.sm_pos_1` is initialized as `None` (lines 187-189 of the same listing).

- for stability numerical reason, the numerator and denominator in the computation of the distance $d$ (see equation 4) are multiplied by $\exp(-k_d\|N_{\boldsymbol{p}} - \boldsymbol{x}\|^2)$ to prevent an explosion of the exponential when a node is far away from the current sensorimotor position, as $k_d\|N_{\boldsymbol{p}} - \boldsymbol{x}\|^2$ can become quite large. See lines 39 and 40 of listing 2 where `RBF(self, sm_pos, pos=None)` computes $d(N_{\boldsymbol{p}}, x)$.

- as in the code provided to us by M. Egbert, a node is created only if the squared norm of the velocity component is less than 10 ($\|N_{\boldsymbol{v}}\|^2 < 10$).

```
      def reset_dsm(self):
186       self.sm_pos_1 = None

188   def step(self, dt, sm_pos):

190       if self.sm_pos_1 is None:
              self.sm_pos_1 = sm_pos.copy()
192           return

194       # Increment the age of all the nodes
          self.nodes['age'] += dt
196
          # Computes the density at sm_pos
198       phi, dists = self.phi(sm_pos)

200       # Update the weights
          self.nodes['weight'] += dt * (-1.0 + 10.0 * dists)
202
          # Do we need to create new nodes ?
204       if phi < self.Kt:
              dsm_pos = (sm_pos - self.sm_pos_1)/dt
```

**Listing 1.** A `step` creates nodes in line 203, updates the weights $w$ according to equation 5 in line 198 using $d$ computed in line 195, in file `IDSM.py`

```
      def RBF(self, sm_pos, nodes_positions=None):
36        if nodes_positions is None:
              nodes_positions = self.nodes['position']
38        l2 = np.sum((nodes_positions - sm_pos)**2, axis=1)
          coeff = np.exp(-self.Kd * l2)
40        dists = 2.0 * coeff / (coeff + 1.0)
          return dists
```

**Listing 2.** Compute distance $d$ according to equation 4, in file `IDSM.py`. The position of the nodes is optionally provided as the `nodes_position` argument and is used to provide either the subset of the active nodes or all the nodes which is the default if none is provided.

## 2.4 Update of nodes

After it has been created, the weight of a node will change as described by the differential equation (5). Its age is also increased at each time step by `dt`. Two factors influence this update rule. There is a steady decrease of the weight (the $-1$ coefficient) thus a node positioned where the agent never comes back will slowly have null influence on the agent decision. But, every time the agent comes near the position $N_{\boldsymbol{p}}$ of the node, its weight will be increased thanks to the $r(N, \boldsymbol{x})$ factor.

$$\frac{dN_w}{dt} = -1 + r(N, \boldsymbol{x}) \tag{5}$$

$$r(N, \boldsymbol{x}) = 10.d(N_{\boldsymbol{p}}, \boldsymbol{x}) \tag{6}$$
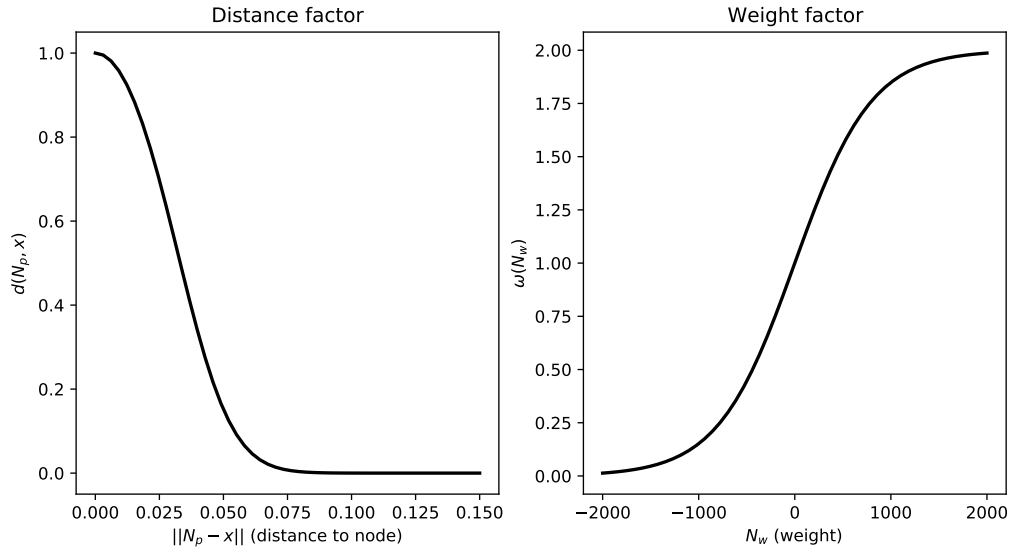
**Figure 2**. **Non-linear functions** used to calculate the node-density of a SM-state, and to scale the influence of nodes by their proximity to the current SM-state (left), as given by equation 4, and by their weight (right), as given by equation 3. See `intro-figs.py`. This is reproduction of figure 1 of the original work. The parameters are $k_d = 1000$ and $k_\omega = 0.0025$.

In our simulation, we use the forward Euler method to numerically integrate that equation, with `dt`= 0.01, as seen in the `step()` function of listing 1, line 198.

## 2.5 Influence of Nodes on the motor

Only nodes which have been created for some period of time $\tau$ will influence the motor state of the agent. This set of active nodes at time $t$ is denoted $\mathcal{A}(t)$, often written $\mathcal{A}$ when $t$ is implicit.

$$\mathcal{A}(t) = \{N \in \mathcal{N} | N_a > \tau\} \tag{7}$$

and thus we can define a density function $\phi_{\mathcal{A}}$ using only active nodes by:

$$\phi_{\mathcal{A}}(\boldsymbol{x}) = \sum_{\mathcal{A}} \omega(N_w).d(N_{\boldsymbol{p}}, \boldsymbol{x}) \tag{8}$$

This density is computed over the active nodes, compared with the density $\phi_{\mathcal{N}}$ (equation 2) used for deciding if a node is to be created and computed over all the nodes. The influence of an active node $N$ on the agent is twofold. The first, a "velocity factor", is simply the motor component of the SM-velocity $N_{\boldsymbol{v}}$ of the node. The second is an "attraction factor" that is akin to a force drawing the agent to the SM-position $N_{\boldsymbol{p}}$ of the node. The idea is to attract the agent to part of the SM-space with a higher density of nodes.

For a given node $N$, the attraction factor creates a force which is perpendicular to the $N_{\boldsymbol{v}}$ of this node by subtracting from the raw attraction the component which is parallel to $N_{\boldsymbol{v}}$. See equations (10, 11).

$$V(\boldsymbol{x}) = \sum_{\mathcal{A}} \omega(N_w).d(N_{\boldsymbol{p}}, \boldsymbol{x}).[N_{\boldsymbol{v}}]^{\mu} \tag{9}$$

**Table 1**. Brief description of symbols and parameters

| Symbol | Short description |
|---|---|
| $\boldsymbol{x}$ | current SM-position of the agent |
| $N_{\boldsymbol{p}}$ | SM-position associated with node $N$ |
| $N_{\boldsymbol{v}}$ | SM-velocity indicated by node $N$ |
| $N_w$ | scalar, weight of Node $N$ |
| $N_a$ | scalar, age of a Node $N$ |
| $\phi_{\mathcal{N}}(\boldsymbol{x})$ | density of Nodes at current agent SM-position |
| $\omega(N_w)$ | function describing how the weight of a node scales its influence |
| $d(\boldsymbol{x}, \boldsymbol{y})$ | distance between two SM-positions |
| $r(N, \boldsymbol{x})$ | function describing how the weight of Node $N$ is increased |
| $\phi_{\mathcal{A}}(\boldsymbol{x})$ | density of *active* Nodes at current agent SM-position |
| $V(\boldsymbol{x})$ | velocity factor (i.e. how influent nodes would like the current SM-velocity to change) |
| $A(\boldsymbol{x})$ | attraction factor (i.e. the attraction factor of influent nodes on the current SM-position) |
| $\boldsymbol{\mu}$ | motor command of the agent, influenced by $V(\boldsymbol{x})$ and $A(\boldsymbol{x})$ |
| dt | euler integration step (usually 0.01 time unit) |
| $k_\omega$ | weight parameter (usually 0.0025) |
| $k_d$ | distance parameter (usually 1000) |
| $\tau$ | age of node to be active (usually 10.0) |

where $[N_{\boldsymbol{v}}]^\mu$ is the projection to the motor part of the SM-space.

$$A(\boldsymbol{x}) = \sum_{\mathcal{A}} \omega(N_w).d(N_{\boldsymbol{p}}, \boldsymbol{x}).[\Gamma(N_{\boldsymbol{p}} - \boldsymbol{x}, N_{\boldsymbol{v}})]^\mu, \qquad (10)$$

$$\Gamma(\boldsymbol{a}, N_{\boldsymbol{v}}) = \begin{cases} \boldsymbol{a} - \left\langle \boldsymbol{a}, \frac{N_{\boldsymbol{v}}}{\|N_{\boldsymbol{v}}\|} \right\rangle \frac{N_{\boldsymbol{v}}}{\|N_{\boldsymbol{v}}\|} & \text{if } N_{\boldsymbol{v}} \neq 0 \\ \boldsymbol{a} & \text{otherwise} \end{cases}, \qquad (11)$$

where $\langle x, y \rangle$ is the scalar product of the vectors. We introduced the case $N_{\boldsymbol{v}} = 0$ in equation (11) which was not present in the original paper.
Then the motor command $\boldsymbol{\mu}$ of the agent is updated according to the differential equation (12)

$$\frac{d\boldsymbol{\mu}}{dt} = \begin{cases} \frac{V(\boldsymbol{x}) + A(\boldsymbol{x})}{\phi_{\mathcal{A}}(\boldsymbol{x})} & \text{if } \phi_{\mathcal{A}}(\boldsymbol{x}) > \epsilon \\ 0 & \text{otherwise} \end{cases} \qquad (12)$$

We introduced the case $\phi_{\mathcal{A}}(\boldsymbol{x}) < \epsilon$ in equation (12). In our code, we set $\epsilon$ to the smallest floating point value as returned by numpy (see IDSM.py, line 158). Using only the active nodes for computing the influence is justified in the original work, in section 2.2, where it is written: "*A short period of time after creation (10 simulated time-units), nodes are activated, meaning that they are added to the pool of nodes that influence the motor state*".
Thus, the motor command is computed as the current motor part of the sensorimotor space to which $d\boldsymbol{\mu}$ is added. In short:

$$\boldsymbol{\mu}_t = [x]^\mu \qquad (13)$$

$$\boldsymbol{\mu}_{t+dt} = \boldsymbol{\mu}_t + \frac{d\boldsymbol{\mu}}{dt} \times dt \qquad (14)$$

where, in addition, we make sure (by clamping) that $\boldsymbol{\mu}_{t+dt}$ lies in $[0, 1]$. This clamping is also present in the code provided to us by M. Egbert.
The right plot of figure 3 depicts the influence $\frac{d\boldsymbol{\mu}}{dt}$ (attraction plus velocity) of a single node, with weight $N_w = 0$, positioned at $N_{\boldsymbol{p}} = (0.5, 0.5)$ in a 2-motor, 0-sensor IDSM.
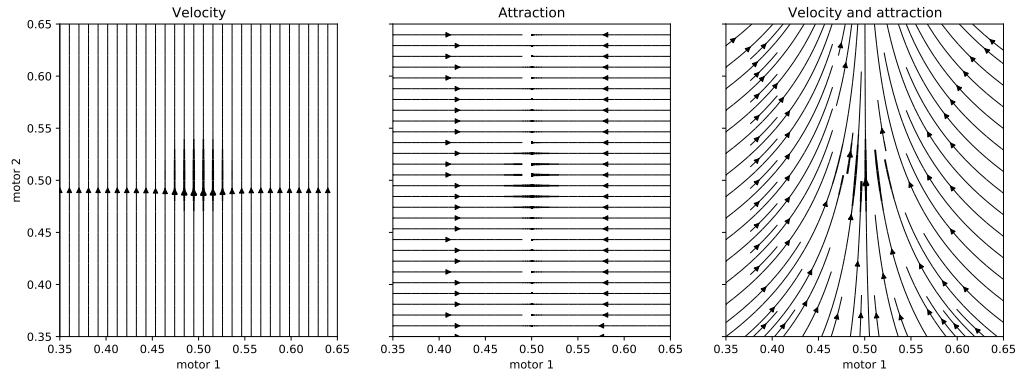
**Figure 3. The influence of a single node**. Left plot shows the velocity part (equation 9), middle plot shows the attraction part (equation 10), right plot shows the resulting influence for a single node situated at $N_p = (0.5, 0.5)$. We use $k_d = 1000$ and $k_\omega = 0.0025$. Note that since isolines are represented on the steamplots, $k_\omega$ does not influence the plots. The rightmost plot is a replication of figure 2 of the original paper, (see `intro-figs.py`).

For this node, the velocity vector is $N_v = (0, 0.1)$, so it is exactly vertical and thus, all horizontal motions are only due to the attraction factor $A$ of the IDSM (as seen on the middle plot of the figure) while the vertical motion is due to the velocity factor $V$ of equation (12), shown in the left plot of the figure. Other parameters are $k_\omega = 0.0025$, $k_d = 1000$ and the node is active.

## 2.6 Influence of four nodes, role of the weights

The plots of figure 4 provide a visualization of how the weights modulate the influence of nodes. In this example that we tried to replicate from the original work (see figure 3 of [1]), we plot the influence of four nodes (depicted as bold arrows) on a 2-motor, 0-sensor IDSM by displaying the motor field $\frac{d\mu}{dt}$ created by these nodes. The weights of all nodes are set to 0 except for the rightmost node (node0). The weight of this node (node0, in a circle on the plots) varies from $-500$ to $100$. The other values of the nodes, shown in table 2, stay constant. The four nodes are active. As explained below, in order to reproduce these results, we had to depart from the parameters indicated in the original paper. We use $k_\omega = 0.025$ (not $0.0025$) and $k_d = 1000$.

**Table 2.** Values of the four nodes of figure 4

| Node | $N_p$ | $N_v$ | $N_w$ |
|------|-------|-------|-------|
| node0 | $(0.55, 0.5)$ | $(0, -1)$ | varies |
| node1 | $(0.5, 0.45)$ | $(-1, 0)$ | 0 |
| node2 | $(0.45, 0.5)$ | $(0, 1)$ | 0 |
| node3 | $(0.5, 0.55)$ | $(1, 0)$ | 0 |

In fact, if we use the original weight and weight parameters of the paper ($k_\omega = 0.0025$), we get the plot of Figure 5 where the varying weights of node0 do not have as big an influence. Our first thought was to play with the velocity component of the nodes as this value is not given in the original paper.

But, as we are plotting isolines, it is only the relative influence of the Velocity vs Attraction part of $d\mu$ that is relevant. If the velocity is very low, attraction takes precedence and we get "squared" trajectories (see Figure 6, with $N_v = 0.01$). Note also that in this case, even if the velocity is low, since the attraction term is kept orthogonal to the veloc-
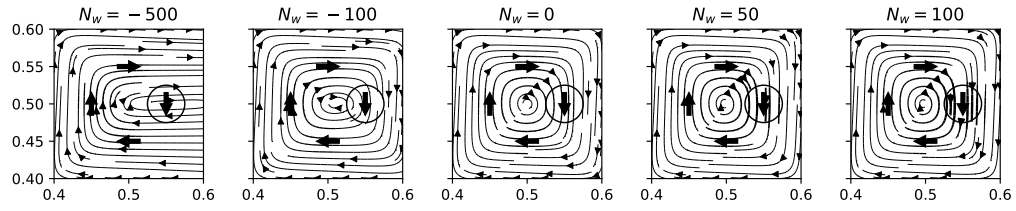
**Figure 4**. Nodes with lower weights have less influence on system-dynamics. The weight of the rightmost node (in a circle) is indicated at the top of each plot. Replicated from the figure 3 of the original work but with a different $k_\omega$ parameter (see text and file `intro-figs.py`).
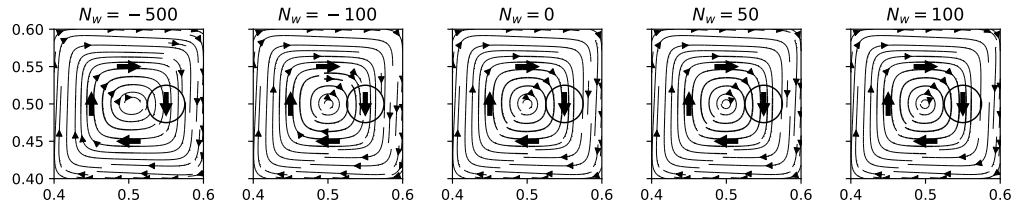


**Figure 5**. Weight influence, original paper values. We use $k_\omega = 0.0025$, $N_v = 1.0$ and the weights indicated at the top.

ity, we do not get isoline pointing straight to the nodes. If the velocity is high, attraction is ignored and we get "rounded" trajectories (see Figure 7, with $N_v = 1000$).
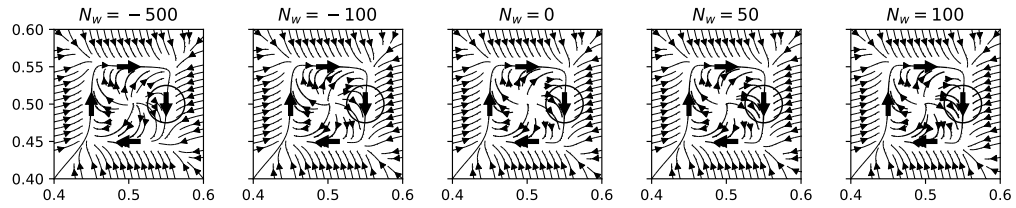


**Figure 6**. Weight influence, original paper values but with low velocity. We use $k_\omega = 0.0025$, $N_v = 0.01$ and the weights indicated at the top.

So, the only solution to replicate Egbert results was to play with the weight part in the equations (9, 10,12). To keep the same weights, we have to change $k_\omega$ and a "lucky" guess did it with setting $k_\omega = 0.025$ (just omitting one "0").
We did some manual computation of the influence of the 4 nodes and found that, at position $(0.55, 0.55)$ with weight of $-500$ and $k_\omega = 0.0025$, the influence $\frac{d\boldsymbol{\mu}}{dt}$ is $(0.692, -0.308)$. It does not seem compatible with horizontal trajectories at this point as in the plot of the original paper. If we set the weight to $-5000$ or $k_\omega$ to $0.025$, we have $\frac{d\boldsymbol{\mu}}{dt} = (1.0, 0)$ at position $(0.55, 0.55)$.

## 2.7 Influence of many nodes

We replicate here the figure 4 of the original work [1]. The IDSM is again a 2-motor, 0-sensor sensorimotor space. For the first 20 time units, the motor state $(m_1, m_2)$ of the agent is externally forced according to the following equations:
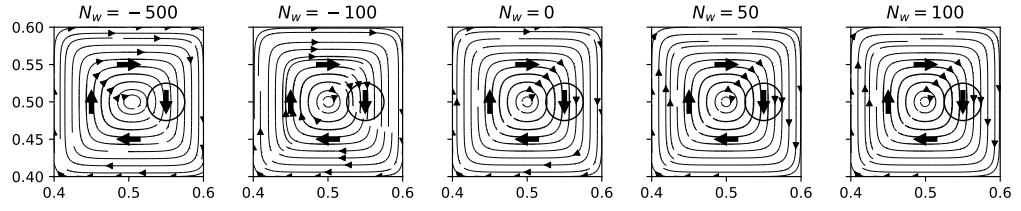
**Figure 7**. **Weight influence, original paper values but with high velocity.** We use $k_\omega = 0.0025$, $N_v = 1000$ and the weights indicated at the top.

$$m_1 = 0.75.\cos(\frac{2\pi}{10}t) \qquad m_2 = 0.75.\sin(\frac{2\pi}{10}t). \tag{15}$$

Then (after $t = 20$), the weights of the IDSM are frozen and the agent is controlled by the IDSM. At time $t = 30$, the two motor values are changed to $(m_1, m_2) = (0.5, 0.6)$ (these particular motor values are taken by the location of the star symbol from figure 4 of the original work) and driven by the IDSM.

**Experiments**    The figure 8, replicated from the original work, shows three plots. On the left, we plot the velocity influence $V$ of $d\boldsymbol{\mu}$, equation (12), of the IDSM at time $t = 20$. The center plot shows the attraction influence $A$ of $d\boldsymbol{\mu}$, equation (12). The right plot shows the resulting influence $\frac{d\boldsymbol{\mu}}{dt}$ when combining velocity and attraction, and we plot in blue the trajectory of the agent from time $t = 30$ to time $t = 50$ starting from the star position. As in the original work, the perturbed agent is "attracted" by the IDSM to the forced behavior imposed on it during the first $20$ time units.

For the experiments, the value of the hyper-parameters were $k_\omega = 0.0025$, $k_d = 1000$, $k_t = 1$ and $dt = 0.01$. In total, 99 nodes are created during the first 20 time units.
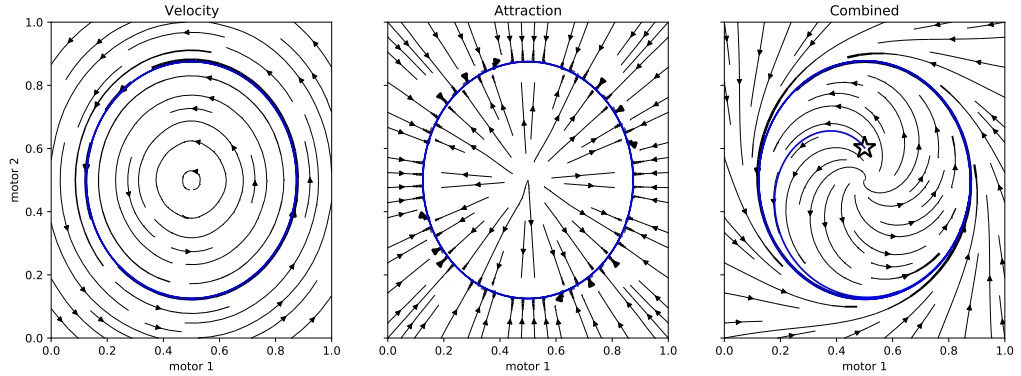


**Figure 8.** **Three snapshots of the 2-Motor IDSM as a fixed dynamical system**. In the left and center plot, the blue curve shows the trajectory imposed on the agent during the first 20 time units. In the left plot, the blue curve shows the sensorimotor trajectory of the agent driven by its IDSM after a perturbation. This figure is a replication of figure 4 of the original paper, see text (and exp-001.py). A video showing the trajectory of the agent is available at https://homepages.loria.fr/ADutech/Video/exp-001.mp4.

```
     def norm_sm_state(m):
70       return (1.0 + m)/2.0

72   def denorm_motor(m):
         return 2.0 * m - 1.0
74
     def get_norm_sm_state(robot):
76       m = robot.get_motors()
         return norm_sm_state(m)
```

**Listing 3.** Normalizing and denormalizing "real" values to sensorimotor space, in file `exp-001.py`

## 3 [Re] Recreating previous sensorimotor behavior

This experiment tries to replicate the experiment detailed in section 3.1 of [1].
The robot driven by the IDSM is a point $\boldsymbol{x}$ which evolves in a one-dimensional environment. A single light source, also represented by a point, is located at the origin of the environment. The SM-state $\boldsymbol{p}$ of the IDSM is two-dimensional and made of a motor-state $m$ and a sensor-state $s$, given by the following equations:

$$\boldsymbol{p} = \begin{pmatrix} m \\ s \end{pmatrix} = normalization \begin{pmatrix} \dot{\boldsymbol{x}} \\ \frac{1}{1+\boldsymbol{x}^2} \end{pmatrix}, \tag{16}$$

where $\dot{\boldsymbol{x}}$ is the velocity of the robot.
The goal is to see to what extend the IDSM is able to memorize a sensorimotor pattern it is first forced to comply so as to keep showing the same behavior once the forcing pattern is removed. Thus, the experiment is made of three phases.

**Training Phase**  At $t = 0$, the agent is positioned at $x = -2.5$. During the first 20 time units of the simulation, its velocity is not influenced by the IDSM but imposed by a controller with $m = \cos(t/2)$. Although the original paper said that the motor control was forced to $\cos(t/2)/2$, our analyze of the original figure 5 shows that the motor amplitude is in fact 1, whereas the original formulation would have give an amplitude of 0.5. Indeed, integrating the motor command ($\cos(t/2)$) leads to a position trajectory with an amplitude of 2, which is coherent with the figure 5 (top) of the original paper.
As shown by the middle plot of figure 9, the controlled velocity of the training phase keeps the robot on the negative side of the light, moving back and forth. During this training phase, nodes are added to the IDSM and updated as explained in sections 2.3 and 2.4. In our experiment, 291 nodes are created during this phase, displayed in the bottom plots of figure 9 as gray dots in the sensorimotor space, behind the colored trajectories. The darker the dot, the lower the weights. In fact, the decreasing influence of equation (5) is stronger than the reinforcement part, leading "old" nodes to see their weights drop to negative values ($\approx -17$). "Newer" nodes are in lighter gray. The arrows depict the velocity component $N_{\boldsymbol{v}}$ of one of every 25 nodes, as in the original paper[1].

**Testing Phase**  At time $t = 20$, the training phase ends and the only influence on the motor of the robot is given by the IDSM, as explained in section 2.5. At the end of this phase, there are still 291 nodes in the IDSM as no new node has been created.

---

[1]Section **3.1**,below figure **5** of the original paper: *"The motor component of activated nodes are shown as gray arrows in the SM-plots of Figure 5, with only every 25th node plotted for clarity"*

**Relocation**  At time $t = 35$, the robot is relocalized at position $\boldsymbol{x} = -2.5$, with its "real" velocity changed to $0$ (and a normed motor value of $0.5$). As in the original paper, the IDSM quickly brings the agent back on the trained behavior. At the end of this phase, there are $460$ nodes it the IDSM, $441$ are active, their weights ranging from $-70$ to $0.3$.
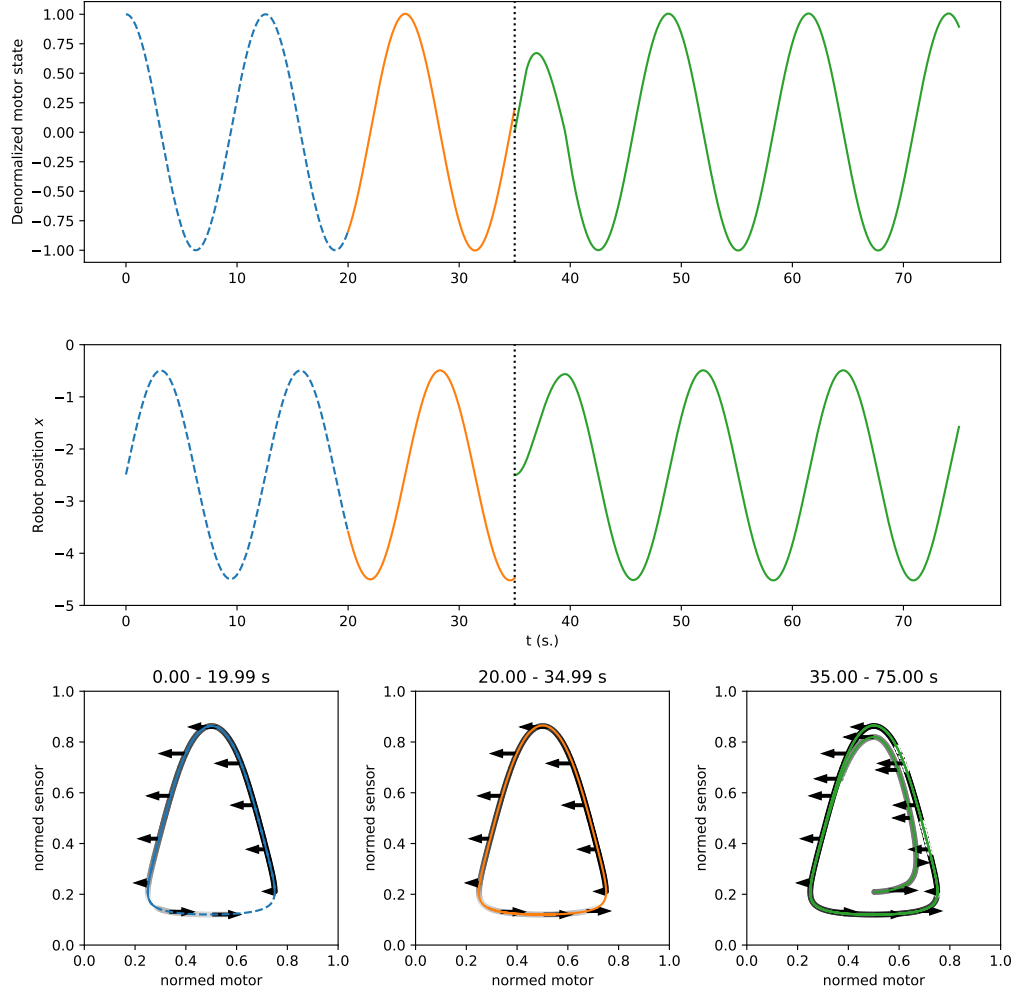


**Figure 9. Training and performance of an oscillatory behavior**. The top plot shows the value of the "real" motor as time progresses. The middle plot shows the trajectory of the agent as its position $\boldsymbol{x}$ against time. The three bottom plots show the trajectory of the agent in its **sensorimotor**-space for three different periods of time: $0 - 20$ (dotted blue), $20 - 35$ (orange) and $35 - 75$ (green). Every 4th node, an arrow represents the node velocity component $N_{\boldsymbol{v}}$ starting from the node positions $N_{\boldsymbol{p}}$. This is a replication of figure 5 of [1] (using exp-002.py). A video showing the trajectory of the agent is available at https://homepages.loria.fr/ADutech/Video/exp-002.mp4.

**Experiments**  For this experiment, we have to depart from the hyper-parameters given in the original paper. With the "original" hyper-parameters[2], our simulation create a total of $87$ nodes (at the end of the relocation phase). If we count the number of nodes in the bottom plots of the Figure 5 of the original work, at least $25$ times the number of arrow, it appears that more than $15 \times 25 = 275$ nodes are created. Thus, we have to "play" with the density function $\phi_{\mathcal{A}}$ that drives the creation of nodes: if we set $k_{\omega} = 0.025$ (as in the experiment of section 2.6), we must set $k_d = 12500$ to get a plot similar to the

---

[2]In the original paper we can read $k_{\omega} = 0.0025$, $k_d = 1000$, $k_t = 1$, $\tau = 10$ and $dt = 0.1$

original plot without touching to $k_t$. So, we used to following hyper-parameters in our experiment: $k_\omega = 0.025$, $k_d = 12500$, $k_t = 1$, $\tau = 10$ and $dt = 0.01$.

In the code repository, file `exp-002.py` can be used to replicate the experiment and draw the plots. Lines 83-94 of listing 4 show how "real" sensor and motor are normalized (and denormalized) to a $[0, 1]$ sensorimotor space. The coefficients of the normalization and denormalization functions have been estimated using the Figure 5 of the original work using the following procedure.

From the variation of the position $\boldsymbol{x}$ of the agent and the sensor function $\frac{1}{1+\boldsymbol{x}^2}$ we can compute the maximum and minimum values of the sensor readings, lets call them `sval_min` and `sval_max`, respectively computed at $0.047$ and $0.8$. Using a ruler on Figure 5, the normalized minimum and maximum values (`snor_min` and `snor_max`) were manually estimated as $0.12$ and $0.86$. For the motor, it is a simple projection from $[-2, 2]$ to $[0, 1]$. These values are then used in the normalization/denormalization procedure, as shown on listing 4.

```
88   sval_min = 0.047
     sval_max = 0.8
     snor_min = 0.12
90   snor_max = 0.86

92   def norm_sensor(s):
         return (s-sval_min)/(sval_max-sval_min)*(snor_max-snor_min) +
             snor_min
94
     def denorm_sensor(s):
96       return (s-snor_min)/(snor_max-snor_min)*(sval_max-sval_min) +
             sval_min

98   def norm_motor(m):
         return 0.5 + 0.25 * m
100
     def denorm_motor(m):
102      return (m - 0.5)/0.25
```

**Listing 4.** Part of `exp-002.py` file that link "real" world to the agent sensorimotor space.

## 4  [Re] Training functional habits

For this experiment where we try to replicate the results of section 3.2 of the original work, the agent is modeled as a two-wheel robot evolving in a two-dimensional toric world where a single light lies at the origin of the environment. Figure 10 shows the agent with its two wheels and two light sensors. The agent is a circle with a radius $r = 0.25$, and the light sensors are situated on the circle, at an angle $\pm\beta$ from the heading of the agent. If $(x, y, \alpha)$ is the position ($x$ and $y$) and orientation ($\alpha$) of the agent and $\boldsymbol{m} = (m_l, m_r)$ is the speed of its wheels (left and right), the dynamic of the agent is governed by these differential equations:

$$\dot{x} = \cos(\alpha)(m_l + m_r), \tag{17}$$

$$\dot{y} = \sin(\alpha)(m_l + m_r), \tag{18}$$

$$\dot{\alpha} = \frac{m_r - m_l}{2r} = 2(m_r - m_l). \tag{19}$$

The activity of the sensors depends on the distance $D$ between the sensor and the light situated at $(x = 0, y = 0)$ and also on the angle between the direction the sensor is facing

(vector $\boldsymbol{b}$) and the direction of the light seen from the sensor (vector $\boldsymbol{c}$), see figure 10. The sensor activity is computed according to:

$$s = \left\lfloor \frac{\langle \boldsymbol{b}, \boldsymbol{c} \rangle}{\|\boldsymbol{b}\|\|\boldsymbol{c}\|} \right\rfloor^+ \frac{1}{1+D^2}, \tag{20}$$

where $\lfloor . \rfloor^+$ is either $0$ or the positive value of its argument and $\boldsymbol{b} = (\cos(\alpha+\beta), \sin(\alpha+\beta))$ This equation, where the scalar product between vectors $\boldsymbol{b}$ and $\boldsymbol{c}$ is correctly written, and where $\boldsymbol{c}$ is normalized, replaces the equation (12) of the original paper $s = \frac{\lfloor \boldsymbol{b}.\|\boldsymbol{c}\| \rfloor^+}{1+D^2}$.
It must also be noted that, in order to ensure a proper simulation of light sensing in a toric world of size 4 centered in $(0,0)$, we add 8 "virtual" lights that lie outside the center section (at positions $(3,0), (3,3), (0,3), (-3,3), ...$) and compute the value of a sensor as the maximum value over all possible virtual lights. See `exp-003.py`, lines 132-144.
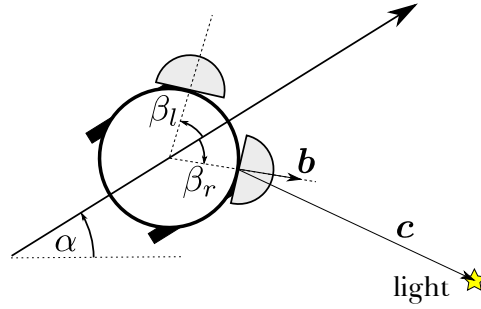


**Figure 10. Robot with two motors and two directional light sensors**. On the depicted setting, only the right sensor will "see" the light.

Thus, the sensorimotor space of the agent is made of 4 dimensions, 2 for the motors and 2 for the sensors.

$$\boldsymbol{p} = \begin{pmatrix} m_l \\ m_r \\ s_l \\ s_r \end{pmatrix} \tag{21}$$

As in the previous experiment, the agent has a training phase where its motors are controlled to impose upon it some desired behaviors. The controller issues a command $\boldsymbol{\chi}$ which is used to change the motor value as described by equation 22.

$$\frac{d\boldsymbol{m}}{dt} = (\boldsymbol{\chi} - \boldsymbol{m}). \tag{22}$$

Three different behaviors have been used: phototaxis (the agent is drawn to the light), sinusoidal-phototaxis (attracted to light but with a sinusoidal perturbation) and photophobia (turns the agent away from the light). In the original paper, the sensory input is denoted $\sigma$ which, we suppose, is a normalization of the sensor values (but, in this setting, the sensor value $s$ is always in $[0, 1]$). And, as in the original paper, the command $\boldsymbol{\chi}$ is clipped to $[-1, 1]$.

### Simple phototaxis

$$\chi_l = 1 - 1.5\sqrt{\sigma_l} \tag{23}$$
$$\chi_r = 1 - 1.5\sqrt{\sigma_r} \tag{24}$$

### Sinusoidal-phototaxis

$$\chi_l = 1 - 1.5\sqrt{\sigma_l} + \sin(2t)/2 \tag{25}$$
$$\chi_r = 1 - 1.5\sqrt{\sigma_r} + \sin(2t)/2 \tag{26}$$
$$\tag{27}$$

### Simple photophobia

$$\chi_l = 1 - 1.5\sqrt{\sigma_r} \tag{28}$$
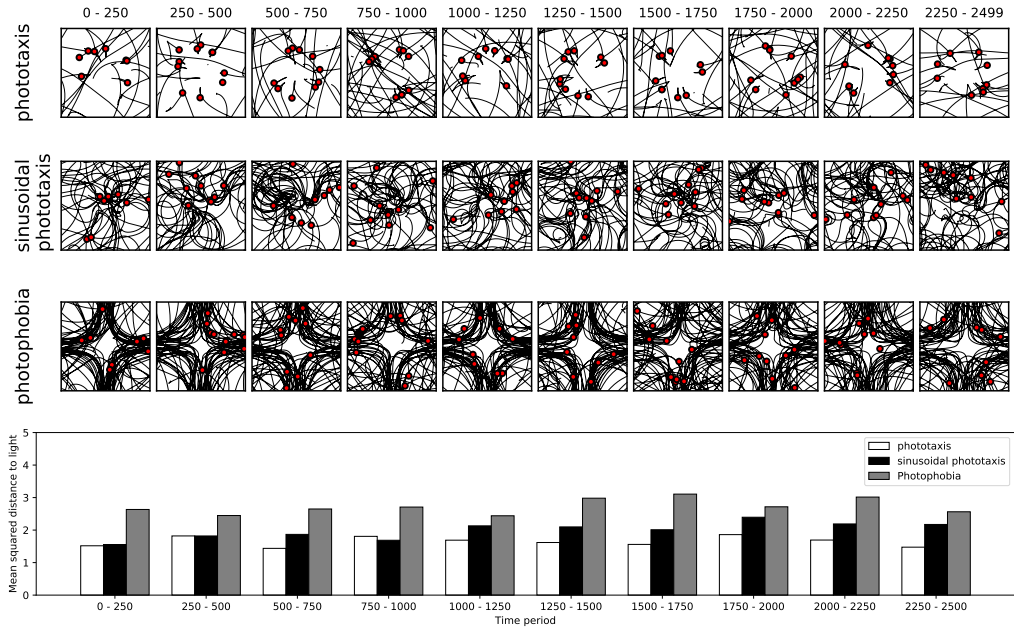$$\chi_r = 1 - 1.5\sqrt{\sigma_l} \tag{29}$$



**Figure 11.** Training of phototaxis and photophobia behaviors and the long term evolution of each of the trained behavior. The square frames show spatial trajectories of the agent for various imposed training behaviors (indicated on the left) at various periods of simulation time (shown on top). The agent is relocated every 25 time-units and the red dots show where the trajectory of an agent ended before its relocation. The bar chart shows the mean distance of the agent from the light (positioned at the center of the environment) for each type of behavior. This is a *quasi*-replication of Figure 7 of [1] (using exp-003.py for generating the experiments and plot_mean_sq.py for generating the histogram). Videos showing the trajectories of the agents are available at https://homepages.loria.fr/ADutech/Video/exp-003-SimplePhototaxis.mp4, https://homepages.loria.fr/ADutech/Video/exp-003-SinePhototaxis.mp4 and https://homepages.loria.fr/ADutech/Video/exp-003-Photophobia.mp4

**Experiments** For these experiments, the values of the hyper-parameters are the one written in the original paper ($k_\omega = 0.0025$, $k_d = 1000$, $k_t = 1$, $\tau = 10$ and $dt = 0.1$). We draw the attention on the fact that in the original paper the $\beta$ for the position of the sensors is said to be $\frac{\pi}{3}$. But using $\beta = \frac{\pi}{3}$ in our first simulations led to results inconsistent with the original paper. In particular, the agent was not able to stop in front of the light, contrary to what is suggested by the two top row plots of the original Figure 7 (phototaxis). Indeed, with $\beta = \frac{\pi}{3}$ and the phototaxis behavior of equations (23)-(24), there is no way for the agent to stay still in front of the light (see Figure 12). A value of $\beta = \frac{\pi}{6}$ (and thus an angle of $\frac{\pi}{3}$ *between* the sensors) seems to solves this issue by offering a

null velocity target attractor to the agent: in Figure 12 the leftmost intersection between the null velocity (red horizontal line) and the dotted $\chi$ velocity with $\beta = \frac{\pi}{6}$. At this point, the agent is at an approximate distance of $1.1$ from the light. Oddly, the plots of the original paper (Figure 7) seem more consistent with a stopping distance of $0.65$, which would suggest a bigger $\beta$ (approx. $\frac{\pi}{4.7} \approx 0.668$), leading to a fixed point very close to its unstable neighbor.

Although the original paper indicated that the agent was relocated every $50$ time-units, we relocated it every $25$ time-units because it is the only way to have $10$ trajectories in a $250$ time-units period as can be seen on the Figure 7 of the original paper.
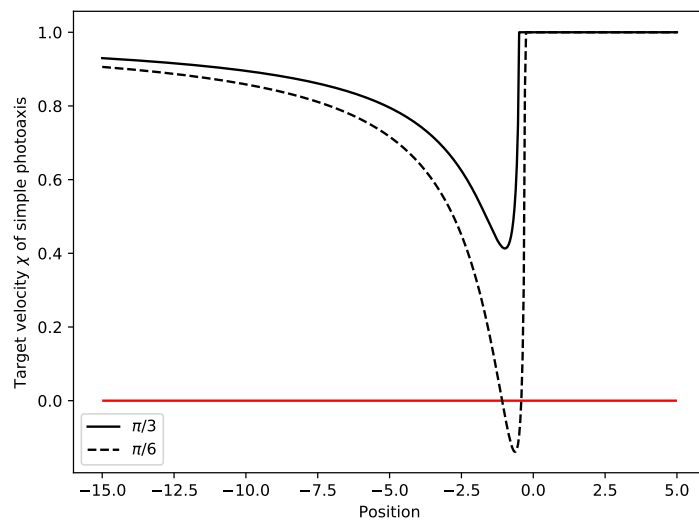


**Figure 12.** We plot the target velocity $\chi$ of an agent facing the light at various positions. With $\beta = \frac{\pi}{3}$ (solid line), this target velocity is never null (using `sensor_response.py`).

The results of our own experiments are summarized in the Figure 11 of this paper. The bar chart, showing the mean distance of the agent from the central light source, depicts the same trend than the original experiments. In general, the mean distance to the center of both phototaxis and sinusoidal phototaxis are approximately the same and lower than for photophobia. However, the exact values of the distances differ from the original paper. Photophobia keeps the agent at a mean (square) distance of approximately $2$ from the center, phototaxis keeps the agent closer to the light (app. distance of $1.5$), and sinusoidal phototaxis is close to taxis. Furthermore, the trajectories we get seem qualitatively different from the original paper. Our agent is generally animated with a larger velocity, which lead to trajectory that can "cross" the light at high speed, exiting the frame at one side to reenter at the opposite side (this is a world with "periodic boundary conditions", see Section 3.2 of [1]). Randomness plays its role here, and we cannot pretend that we should always have the same kind of trajectories. Indeed, changing the seed (we ran our experiments with a seed equals to $0$) does change the observed behaviors. We have the impression that, besides our comments about the value of $\beta$, the normalization and clamping of the motor, or velocity, of the agent may be part of the explanation. The IDSM model, at its core, does not guarantee that the velocity of the agent stays within $[0, 1]$ in the sensorimotor space. So, we suspect that some sort of clamping is used by the authors[3]. The original paper talks[4] about clamping $\chi$ in $[-1, 1]$ but by definition (equations (23)-(29)), $\chi$ always lies in $[-1, 1]$. So we think that clamping

---

[3]This has been confirmed by M. Egbert during our privates exchanges

[4]*"The equation below describe the target [...] motor values $(\chi_l, \chi_r)$ [...] which are limited to lie in range $[-1, 1]$ and then used to update [...] motors $(m_l, m_r)$ [...]"*, Section 3.2, a bit above equations (13), in [1].

is performed on the motors.

In the code repository, file `exp-003.py` can be used to replicate the experiment and draw the various plots of Figure 11. Listing 5 shows how "real" sensor and motor are normalized (and denormalized) to a $[0, 1]$ sensorimotor space.

```
104   def norm_sensor(s):
          return s

106   def denorm_sensor(s):
          return s

108
      def norm_motor(m):
110       return 0.5 + 0.5 * m

112   def denorm_motor(m):
          return (m - 0.5)/0.5
```

**Listing 5**. Part of `exp-003.py` file that links "real" world to the agent sensorimotor space.

## 5  [Re] Emergence of self-organized habits

These experiments, also taken from [1], were not very detailed in the original paper. The main idea is to let the agent behave randomly while creating nodes (the equivalent of a training phase) and then analyze the behaviors that are induced to detect some regularities or broad categorizations.
The environment and the agent are similar to the previous experiment (see section 4) with one light source at the origin, the agent has thus a four-dimensional sensorimotor space with 2 motors and 2 light sensors.

**Random initialization of IDSM**    In this phase, 5000 nodes are randomly created. To do this, 100 random walks in the SM-space of the agent are generated, each starting from a random location $l$ in the SM-space. From every starting random location, 50 more locations $l_i$ are generated using:

$$l_{i+1} = l_i + r \qquad\qquad for\ i \in 1, \ldots, 50, \qquad (30)$$

where $r$ is a vector randomly generated in $[-0.05, 0.05]^4$ using a uniform probability distribution. In the event where some components of $r$ would lead the agent outside the bounds of the normalized SM-space, these components are inverted.
A new node $N$ is created at every location $l_i$ (except the last one) of the trajectories with the following values:

$$N_{\boldsymbol{p}} = l_i$$
$$N_{\boldsymbol{v}} = \gamma(l_{i+1} - l_i) \qquad\qquad (31)$$
$$N_w = 0 \qquad\qquad (32)$$

. $\gamma$ is a scaling factor set to $1.0$ in the original publication. At the end of this initialization period, all the nodes are activated.

**Clarification needed for the experimental setup**    After the initialization phase, the authors generate agent trajectories that are identified as belonging to five categories (see Figure 8 of the original paper). For discussion, we can name these behaviors as:

- `large circle` (cyan)

- `square` (red)

- `small circle` (green)

- `point` (purple)

- `flower` (blue)

We were unable to reproduce all these behaviors. While the initialization phase is clearly described in the paper, there remains several unanswered questions or concerns that may explain why we were unable to reproduce these results:

1. is the arena still periodic ? The trajectories depicted in Figure 8 of the original paper do not have discontinuities that would reflect the periodic boundary conditions.

2. are nodes created only in the initialization phase or also during the trajectory generation ?

3. are all the subsequent trajectories generated from the same initialized IDSM ?

4. while the sensorimotor space is $[0, 1]^4$, the PCA projection of the trajectories in Figure 8 of the original paper has coordinates in $[-3, 5] \times [-1, 6]$ which is surprising.

**Clarification by the authors about behavior generation**    Fruitful exchanges with Matthew Egbert gave us some answers to questions 1, 2 and 3.
Answering points 2) and 3) above, Egbert told us that *one IDSM is initialized* with 5000 nodes and then all subsequent trajectories (lasting 100 units of time) are generated in sequence, possibly *adding their own new nodes* to the IDSM. And the last 25 time units of every sequence are displayed and analyzed.
Answering point 1), the arena is periodic in the original work. We are still puzzled by the plots of the Figure 8 so, for this experiment, we opted for a different implementation of a periodic world. Instead of relocating the agent when it crossed the "borders" of the arena , as done in the previous experiment (see `exp-003.py`), we computed a "virtual" position of the agent (using the modulus operator) that is always in $[-2, 2] \times [-2, 2]$ physical space, regardless of the real position of the agent in physical space. The sensor values, and thus the IDSM motor command, are computed using this virtual position (see `exp004.py` with `-repeating_env` option). Trajectories are plotted using the real positions of the agent.
Matthew Egbert sent us his code base, which is comprised of several developments for serveral experiments and papers over the years. We had a detailed look at this code but we were unable to run it, as it relies on GPU computing and the use of some unavailable library. Nevertheless, it helped us a lot in understanding small details that we mentionned earlier (clamping $\mu$, conditions for the creation of a node, virtual lights, etc). M. Egbert was also very kind to correct many typos of the draft version of the paper we send him, and we are really thankful for the time he took in trying to understand with us why our results are different from what he obtained.
As to what could explain the discrepancies between their results and ours in the experiments of this section, he puts forward three possible elements:

- a possible difference in the sensor function used in our experiments. But this argument was based on a wrong interpretation of the description of the sensor function we used. As M. Egbert provided the code for their function, we were able to check that we use the *same* sensor function.

- the fact that, at the time of the conversation, we randomly initialized the IDSM before each trial. But we are now initializing only once, at the start.

- the relative rates of change of the IDSM and the robot's activity in space. What he mean is this (quoted): "*when moving through the environment, the sensory activity changes at some rate (determined by the velocity of the robot and light, and the function describing the sensor activity). The motor state of the robot also changes at some rate (determined by the distribution of nodes, and the magnitude of their velocity and attraction components). If the latter rate of change is very slow compared to the former, then the habits that tend to form tend to be quite uninteresting. Essentially, the nodes end up having very little influence upon the motor state before they are departed because the sensory state is changing so quickly that any given node only influences the system for a short period of time. In this case the self-organizing habits that might form in the 2D sensorimotor space depicted in Figure 5 [*Figure 9 bottom in the present paper*] would be (mostly) up and down, without much variation left and right...i.e. the motor state would be relatively constant. This would correspond to the primarily circular patterns you are seeing (motor state is changing very little). You might try increasing the magnitude of the random velocity vectors you are creating at the start to see if you observe more interesting patterns of behaviour.*" In fact, we tried with various values for the magnitude of the step $r$ in equation 30 and with scaling up or down the norm of the $N_v$ component of nodes (by playing with the $\gamma$ factor of equation 31) added during the initialization (these are the parameters `rnd_step` and `rnd_scale` in script `exp-004.py`). We also experimented with several values for $k_d$ and $k_\omega$, and we were not able to observe `flowers` or `square` trajectories.

Furthermore, we have not puzzled out the discrepancy we observe in the "simple" experiment with 4 nodes, detailed in section 2.6. As we were able to reproduce other experiments, this might have no link on us not being able to replicate the results from the last experiments.

**Analysis of our experiments**   As illustrated by Figure 13, we mainly obtain `circle` trajectories of various radius[5]. The center of these circles usually drifts over time. Some curved trajectories are no more circles but look more like loops (e.g. $5th$ or $42nd$ plot). In agreement with the simulations of the authors, we observe small ($2nd$ plot) and large circles ($1st$ plot).
We have seen no `square` nor `flower` trajectories. `point` patterns might in fact be 'thick-circles' or even `circles` with very small radius, like for example plots $39$ or $70$.
In Figure 14, we present the distribution of the diameters of circles observed in Figure 13. This figure must be taken with caution as the diameters measurements are very approximate. The python program `figs/exp004_seed0_final/circle_radius.py` lists these measurements and was used to create the plots of Figure 14. We clearly observe two prototypical circle radius (around $0.2$ and $1.0$).

**Any reasons for such a discrepancies with the original work ?**   It appears that several hyper-parameters are crucial to the IDSM. In previous sections, we showed that setting $k_d$ and $k_\omega$ to proper values was the key to replicating the experiments of the original work. In this last experiment, we explored a large range of value for the $k_d$ and $k_\omega$ hyper-parameters but never succeeded in observing behaviors as diverse as the one of the original article.
One must also be careful to use a Euler integration step $dt$ that is small enough. Too big a step can prevent the creation of some nodes as only one node can be created at each time step. And also, a big time step can also explain behaviors where the trajectory of the agent changes very abruptly.
As illustrated in Figures 6 and 7 of Section 2.6, the right balance between the velocity component $N_v$ of node and the resulting attraction in equation (12) must be found in order to get square like trajectories with rounded corned. As a consequence, the random

---

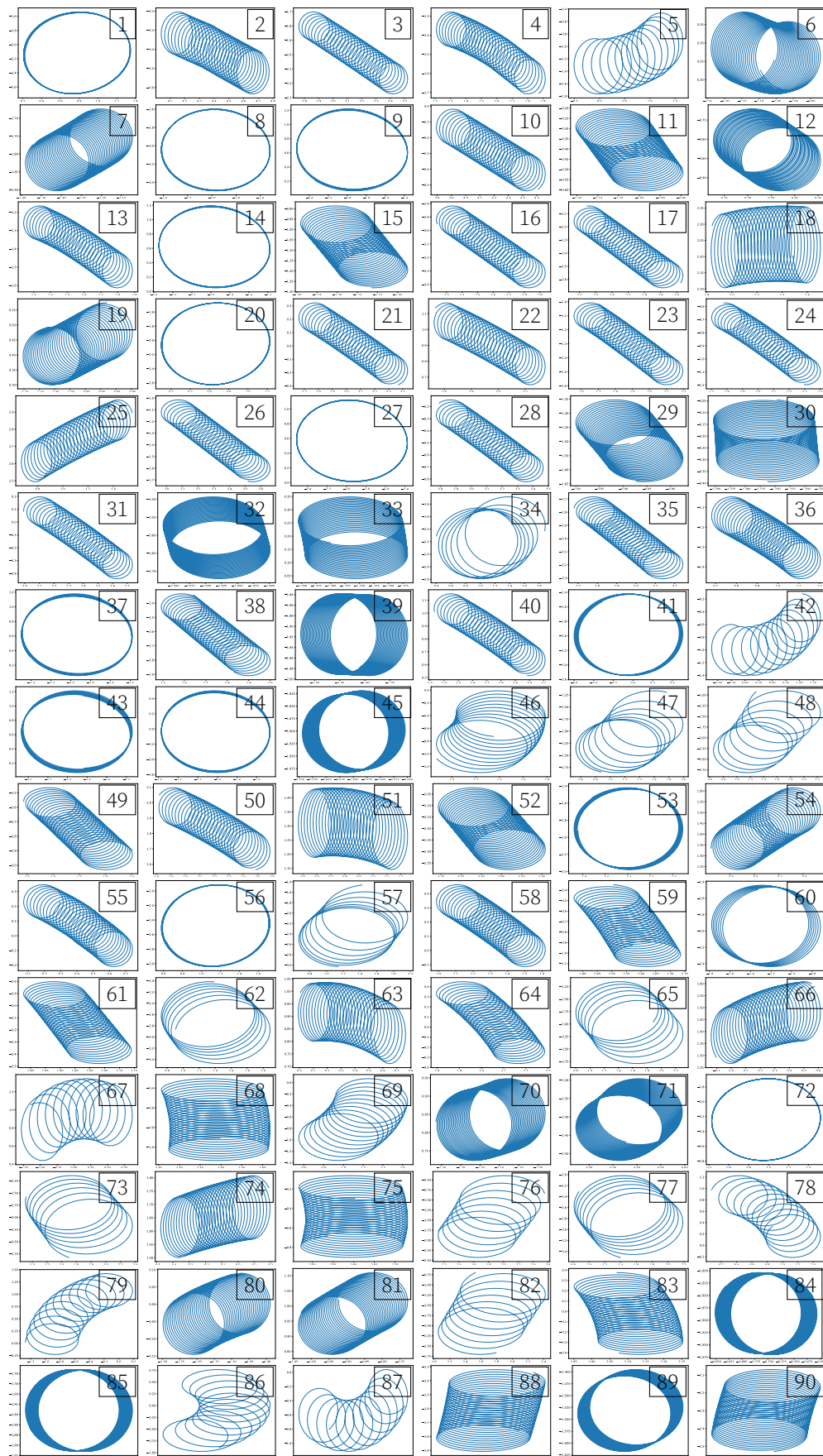[5]command used: `exp-004.py --repeating_env --seed 0 --num_trials 90`

**Figure 13**. The trajectory of the last 25 time-units of 90 agents pre-trained with a random walk generating 5000 nodes. These trajectories are generated in sequence and can add new nodes to the IDSM. $dt = 0.01$, the environment is periodic. (see exp-004.py).
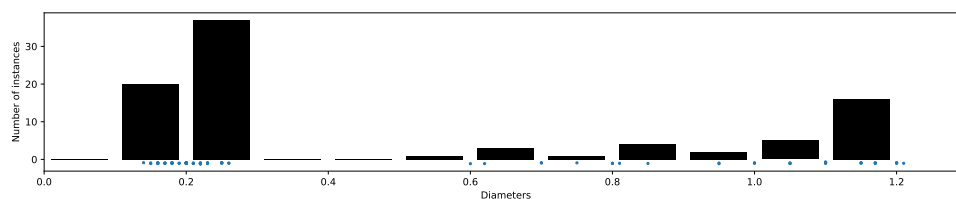
**Figure 14**. The distribution of the diameter values of `circle` trajectories from Figure 13. (see `figs/exp004_seed0_final/circle_radius.py`)

initialization of the IDSM and of the velocity of the agent when it is relocated could also play a role in what behaviors will subsequently be observed.

Then, as pointed out before in the exchanges we had with Matthew Egbert: "*the relative rates of change of the IDSM and the robot's activity in space*" is important. It means that the agent must move at a "correct" (or proper) speed so that the rate of change of its sensor is comparable to the rate of change induced by the IDSM nodes. This consideration also stresses the importance of the starting velocity of the agent, and of the initialization phase of the IDSM (i.e, the sensorimotor distribution of the created nodes). These points have been explored in some subsequent works of Egbert and colleagues, see [3, 4].

Besides, when the agent moves in the simulated environment, it can only roam in a small subpart of the sensor dimensions of the sensorimotor space. As the 5000 random nodes created during the initialization do not take this into consideration, many initial nodes will never have any influence on the agent: the exponential nature of the distance $d$ between the agent and these nodes will always be null. This is one of the reason we tried to use smaller $k_d$, but with no real impact on the results.

In fact, in the large majority of our runs, we observe that the nodes created by the agent during the initialization phase are quickly less influent than the new nodes created during the trajectories. Indeed, during the 25 last steps, the agent is mostly influenced by "newer" nodes and, very often, one or two different nodes only. This might also explain the prevalence of `circle` behavior as the velocity change thus induced lead to a kind or circling, sometime with a drift, for the agent.

## 6 Conclusion

This paper explores the "Iterant Deformable Sensorimotor Medium" (IDSM), a model to study the generation of habits in cognitive agents proposed by Egbert and Barandiaran [1]. We give a python code to implement this model and to replicate most of the experiments of the original paper. The only part we are not able to reproduce is the "Emergence of self-organized habits" (see our Section 5). As the original formulation of these experiments lacks some details, we highlight our choice and lay down some hypothesis explaining what we observe.

This replication work is very instructive for us. We have a better and detailed understanding of the IDSM and some interesting problems in the habits formation can be explicited in this formal model, and our current works aims at addressing several of theses questions.

- The initialization phase is crucial and cannot be left to a random process. What kind of "reflex" or "hard-wired" behavior can bootstrap the system. Egbert experimented with a new kind of random initialization in [3] but, again, some details are missing and the question is still largely open.

- The IDSM, in its current state, does not seem compatible with "learning" several habits and combining them. How could it be altered to do so ?

- Forgetting a habit is allowed through a decrease of the weights of the nodes. But re-acquiring this habit can take as long as engraving it in the first time. Can we add some mechanisms to improve re-acquisition of forgotten habits ?

## Acknowledgment

We would like to thank Matthew Egbert for the fruitfull exchanges which helped to clarify the content of the reproduced paper and to correct typos in our original submission. Althought we were unable to reproduce all the results, we thank the author for the efforts of sharing his code base with us.

## A  Python virtual environment

In order to make running the code easier, we provide a virtual environment for python. As such, installing requirements for the code can be easily done with the following steps.

```
   ## In a directory where the code has been cloned
2  ## create a virtual environment (named venv) for python3.6
   $> virtualenv --python=python3.6 venv
4
   ## activate this virtual environment
6  $> source venv/bin/activate
8  ## and install the requires dependencies (requirements is a file
       provided in the code repository)
   $> pip install -r requirements
10
   ## then run experiments, for example
12 $> cd python
   $> python exp-001.py
14
   ## when done with the virtual environment, unlog from it
16 $> deactivate
```

## References

1. M. D. Egbert and X. E. Barandiaran. "Modeling habits as self-sustaining patterns of sensorimotor behavior." In: **Frontiers in Human Neuroscience** 8 (2014).
2. M. Egbert and L. Cañamero. "Habit-Based Regulation of Essential Variables." In: **Artificial Life 14: Proceedings of the Fourteenth International Conference on the Synthesis and Simulation of Living Systems**. The MIT Press, July 2014, pp. 168–175.
3. M. Egbert. "Investigations of an Adaptive and Autonomous Sensorimotor Individual." In: **The 2019 Conference on Artificial Life** 30 (July 2018), pp. 343–350.
4. M. Zarco and M. Egbert. "Different Forms of Random Motor Activity Scaffold the Formation of Different Habits in a Simulated Robot." In: **Artificial Life Conference Proceedings**. MIT Press. 2019, pp. 582–589.