An Attempt at Grid/Place-Cell-based Navigation

Inga Schöyen
(s1127295)

Friday 31st January, 2025

The field of cognitive robotics lies at the intersection of AI and Computational Neuroscience, bridging the gap between computational models and embodied agents. By integrating insights from both disciplines, researchers aim to develop intelligent systems that can interact with their environment in ways that mimic biological cognition. This work contributes to advancing computational models by implementing architectures for evaluating models of cognitive functions and provides practical intelligent agents that challenge traditional algorithmic boundaries.

While the rise of deep learning and big data has brought on major interest in both cognitive models and intelligent agents, many successful approaches rely on supervised learning, which often focuses on optimizing prediction accuracy or reward signals. Supervised learning alone, however, fails to capture the rich array of adaptive mechanisms that underpin animal nervous systems. Biological mechanisms enable learning through sensory input, motor interaction, and implicit reinforcement, processes that are inherently different from the explicit, labeled training data used in supervised settings.

A growing body of computational studies has explored various types of learning occurring in the brain, including unsupervised, reinforcement-based, and implicit learning. Some work has focussed on describing how these different subsystems integrate into one [1], [2], yet due to the sheer scale of the system, implementing models of macro-structures with multiple large subsystems remains a challenge.

Instead of trying to model the entire system, an alternative approach is to try to uncover the hidden formalism employed by neural systems to encode, internally represent, and decode information in a reliable manner.

One particularly intriguing concept within this approach is that of a ”cognitive map”. Cognitive maps have been proposed as a flexible framework for encoding space and its contents, but their computational implementation remains only partly explained. For example, whether the map is encoded in the activity of individually tuned ensembles, such as the neural engineering framework (NEF) proposes, or rather a distributed representation that emerges within the dynamics of the whole system, like dynamic field theory (DFT) aruges, remains unclear. The building blocks for such maps —neural representations, learning rules, and interaction with sensory data—are not strictly constrained by existing theories or implementations, but rather it seems that all variations of them are championed by someone as the representational vehicles for extracting and storing information.

This underdeterminedness raises important questions about the minimal requirements for effective cognitive models, the trade-offs between biological plausibility and computational efficiency, and the extent to which these models can practical serve as robot control systems.

In this report we present a minimalistic cognitive architecture, that was developed to allow an agent to learn a cognitive map of its environment in an unsupervised manner, based on an auto-encoder model approach.

This report is structured into two parts: (1) a detailed description of the implementation of the cognitive architecture that enables agents to build and use cognitive maps effectively, and (2) a reflection on the implications of such models for advancing our understanding of cognition in robotics.

1 Implementation of the Cognitive Architecture

1.1 Background

The ability to navigate complex environments and form internal representations of sensory experiences is a critical component underlying goal-directed behavior and other higher cognitive functions. Cognitive maps, that is, stable internal representations of the environment, that serve to store information and use them for internal simulation to predict the outcome of potential actions, are central to this process [3], [4]. One popular example of such a cognitive map is that implemented by the grid-cell/place-cell complex (comp. Figure 1): grid cells exhibit hexagonal lattice receptive fields, at varying scales, that sparsely encode a space. The grid cells project to the place cells, which in turn compute a unique encoding of the current location, which shows up as a gaussian bump in the population that smoothly shifts with changing location.

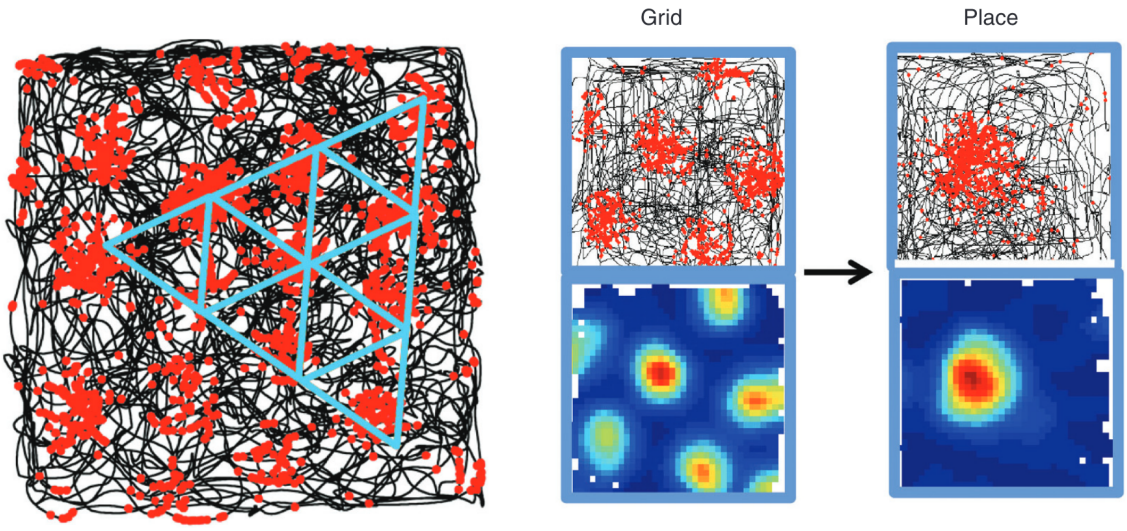

Figure 1: Activities of rat grid and place cells during a foraging task, taken from Moser, Rowland, and Moser [5]. On the left, grid cell activity overlaid on the rats trajectory, with a blue raster highlighting the hexagonal geometry of the receptive field. On the right, grid cell activity (left) compared to place cell activity (right), showing the raster- and Gaussian-like firing patters of grid and place cells, respectively.

It is believed, that the sparse geometric encoding of spatial features via the grid cells gives rise or enables the emergence of place-selective cells. These place cells thus represent a cognitive map, that stores spatial as well as non-spatial information information and is able to retrieve it both when at that location and when recalling it from memory [5].

However, with evermore research emerging on the For instance, grid-like representations emerge in humans during non-spatial tasks involving conceptual or social ”spaces,” while place cells dynamically remap to incorporate contextual cues like colors or rewards. This flexibility raises critical questions: What computational principles allow cognitive maps to integrate multimodal information? How do biological systems learn and stabilize these representations amid noisy sensory inputs?

Computational models have explored these questions by simulating hippocampal-entorhinal circuits. Seminal work demonstrated that grid cells' hexagonal firing patterns naturally arise from neural networks trained to perform path integration under constraints like non-negative synaptic weights [6]–[8]. Alternative models produce square or irregular grids under different assumptions, highlighting the role of optimization pressures in shaping representational geometry (reviewed in [9]). More recently, reinforcement learning frameworks have proposed that cognitive maps encode not just spatial layouts but also policies—associations between states, actions, and outcomes [4]. This might be the key mechanism underlying the computational task of path integration that place cells need to perform in order to maintain their location-specific firing.

1.2 Approach

In this context, we investigate a minimal agent that constructs a cognitive map integrating spatial and color-sequence information. The agent's task — to categorize noisy RGB inputs, remember color transitions, and navigate a grid world — mirrors challenges faced by biological systems. By modeling grid cells (for spatial encoding), place cells (for location-specific activity), and associative memory networks (for place-dependent color-sequence prediction), we test hypotheses about:

1. The sufficiency of biologically inspired mechanisms to handle noisy, high-dimensional sensory data.
2. The role of error-driven learning versus Hebbian learning in forming a cognitive map.
3. The emergence of non-spatial representations (color sequences) within a spatial framework

Here, rather than encorporating semantic pointers for encoding and simpler action rule generation, we encode everything in ensemble states. While there has been a recent work by Dumont, Orchard, and Eliasmith [8] that showed how neural and symbolic representations can be connected to model the grid/place-cell complex, we aimed to start simple and extend with additional modules to explore the contribution of SPA encodings. This should maintain more biological realism, and allows for unsupervised learning, which simultaneously offers greater potential for robotic applications with adaptive memories, such as roombas that need to learn spatial maps. To this end, we make use of the predictive coding

framework, to test and at the same time train the cognitive map in an ecologically valid way, by folding it over and comparing its output to its input. In this vein, we consider the model as an autoencoder in a way, as it aims to reproduce its input, by first encoding it into a high-dimensional space, and then decoding it into a lower-dimensional space again.

1.3 Cognitive Architecture

The cognitive architecture was implemented using the Nengo library [10], expanding on the provided base code in `critter.py` and `grid.py`, and locally run through the `nengo_gui` on an M1 chip. All code and the report are available on github.

The architecture was set up as a combination of nengo.Ensembles and nengo.Nodes linked by custom nengo.Connections to form a nengo.Network, that takes in information through 3 sensors and returns a motor command. The network consists of four core subsystems performing distinct tasks: sensory processing, spatial encoding, associative memory, and motor control.

The sensors include one proximity sensor and two color sensors: the proximity sensor measures the distance to walls at three angles relative to the agent's orientation, slightly to the left, slightly to the right, and straight ahead. Two color sensors provide the RGB values of the current cell and the closest colored cell in the agent's forward direction. These inputs are processed by the `walldist`, `color_cur`, and `color_ah` ensembles, respectively, which all have self-recurrent connections (synaptic weight = 0.02) to stabilize the potentially noisy sensory inputs.

The spatial encoding subsystem comprises grid cells and place cells, as well as a helper ensemble that computes the derivative, in place of the velocity sensors that project to grid cells. The grid cells ensemble (`grid_cells`) encodes spatial location using custom encoders sampled uniformly and scaled to multiple levels $[0.5, 1, 2]$, simulating multiple hexagonal grid-cell populations that capture different scales. The place cells ensemble (`place_cells`) integrates grid cell activity with both color sensor inputs to form unique location representations.

The associative memory subsystem includes a hierarchical memory system within which firstly, the `col_seq_int` ensemble concatenates the two color sensory streams, after which the `col_place_enc` ensemble concatenates the color sequence representation with the place representation. Finally, the prediction ensemble `col_pred`) takes in the place + color sequence representation and passes its output to predict the next color ahead, given the sequence and the place representation. The prediction is passed to the error ensemble (`error_col`), which computes the difference between the prediction and the actual color value ahead, which is used to train the decoding weights (more below in subsection 1.4).

The cerebellar subsystem consists of a cerebellum ensemble (`cb`) that shifts the gain between exploration (`expl`) and exploitation (`scan`) modules using the prediction error magnitude. The `expl` ensemble emulates random, exploratory movements, with a random factor scaling the turning angle and the speed increasing as the error goes down, and vice versa. The `scan` ensemble in turn implements the exploitation strategy of not moving when the error is large, but turning in increments that are inversely proportional to the error. With these two modules being inversely modulated by the prediction error, a naturally balanced strategy of exploring random directions when familiar with the environment, and not moving and scanning the environment, when unfamiliar with the environment, i.e. producing a large prediction error. Additionally, the base movement function implements basic wall-avoiding behavior, by turning to be in between obstacles detected off-center and slowing down proportional to the distance of the obstacle ahead. Finally, the motor commands are summed by a movement ensemble (`mov_out`) and passed to the simulated environment via a movement node (`movement`).

1.4 Learning

The connections that are central to this project are the encoders and decoders to the ensemble that should represent the cognitive map, that is the connections from and to the `col_place_enc` ensemble. We apply the a-priori constraints of non-redundant representations of places and color sequences for a sparse encoding in the memory, which justifies the choice of the Voja learning rule for the encoding weights. The decoding weights in turn were optimised for predicting the closest color ahead, for which the PES learning rule was employed, with the error computed in the `error_col` ensemble.

To learn the different connections within the cognitive map in an unsupervised bottom-up manner, a dual-phase learning schedule was implemented. In the first phase ($t < 10000$ ms), only the Voja learning rules were activated to ensure non-redundant encoding of color sequences and place representations, before storing them in memory, i.e. integrating them in the cognitive map. During this phase, the PES learning rule was disabled to prevent the map from being formed on the basis of premature color sequence and place representations

Following the initial phase ($t \geq 10000$ ms), the Voja learning rules were deactivated, and the PES learning rule was activated for the prediction pathway (`col_place_enc` to `col_pred`).

The learning schedule was controlled by a phase node (`phase_node`) and a gate node (`gate_node`), which modulated the learning rules based on the simulation time.

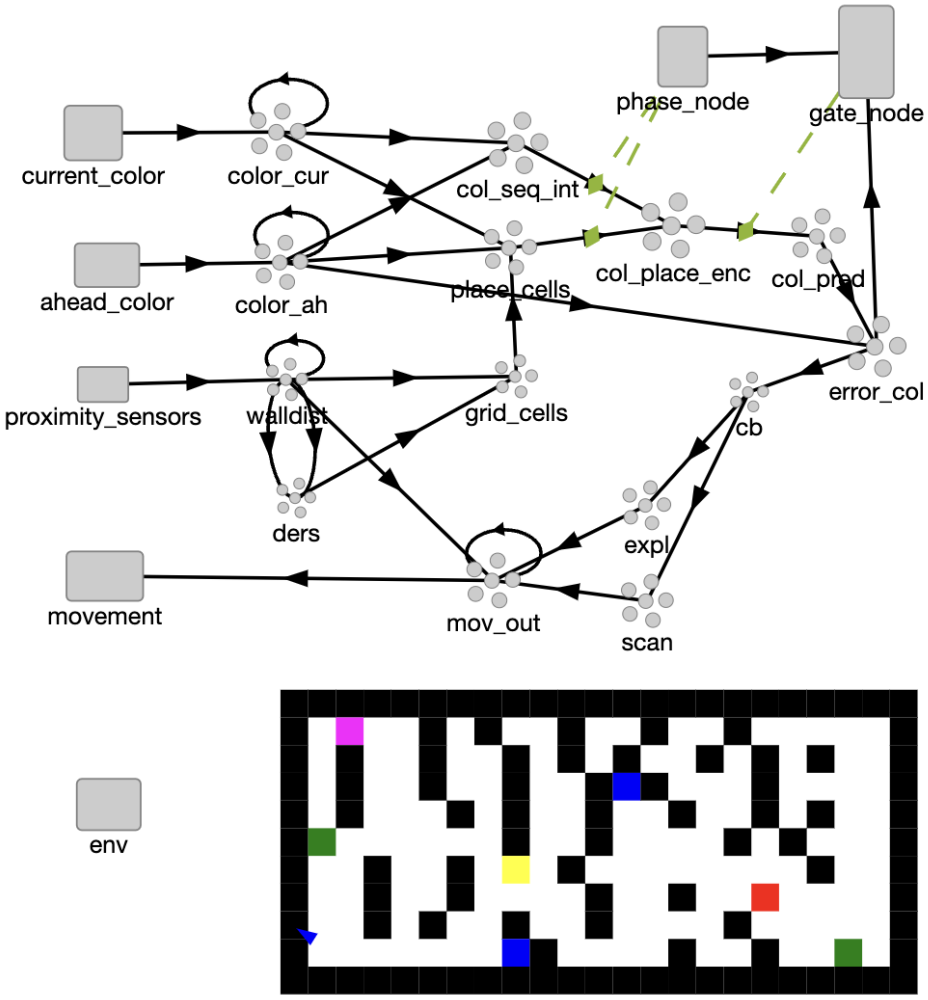

Figure 2: The cognitive architecture as simulated using the nengo_gui application (top) and an illustration of the agent in the grid world (bottom). The dashed green lines indicate the learning rule modulations implementing the learning phase switching.

1.5 Evaluation

As the model did not produce meaningful results, neither in terms of learned behavior nor in terms of learned representatiions, the rest of the presentation focusses on discussing the expected outcomes for a working implementation, as well as the limitations that have emerged so far.

1.5.1 Behaviour

The agent's behavior should be evaluated based on its ability to navigate the environment and adapt its movement strategies. Mo meaningful exploration or exploitation patterns emerged during simulation, as the agent failed to transition between exploration and scanning behaviors effectively. Additionally, it would get stuck going in circles as if stuck in a limit cycle, which indicates that the base motor unit might dominate in such instances. Generally, the agent keeps moving until it gets stuck in a corner or a cull-de-sack, indicating that the scanning behavior does not take hold.

Implementing a Winner-Takes-All mechanism in the `mov_out` ensemble, similar to the spa.BasalGanglia module, should solve this particular issue, since it would allow for only one motor unit to set the motor output, rather than all units getting summed together, as they are now. Following that, the model could be extended to include both a basal-ganglia-like module and a cerebellum-like module, such that the basal-ganglia module diffusely disinhibits larger behavioral sequences, while the cerebellar module locally modulates the gain of specific motor primitives.

1.5.2 Internal Representations

In terms of evaluating the learned representations, the correlation of cell spikes with locations or geometries is a central focus. The ideal results of the learned internal representations would match the receptive fields depicted in Figure 1, however, since we do not enforce non-negative connection weights to the grid cell ensemble or the place cell ensemble, a different geometry might emerge within the grid cell receptive fields. Additionally, the color sequence representations should evenly span some representational space, given the applied learning rule, such that they evenly distribute the cosine angle between all of them. Lastly, a well-trained `col_pred` ensemble should be able to reconstruct the RGB value, such that it can be directly plotted as the color it represents. This extension would enable real-time observation of the prediction dynamics and accuracy.

2 Reflection

The implementation of a minimal cognitive agent to model cognitive maps and spatial navigation raised some interesting considerations and insights into disentangling neural representations. While the architecture was designed to emulate biological principles, such as grid and place cell dynamics, the lack of meaningful results highlights challenges in translating cognitive models into functional computational models.

Grid cells, with their periodic firing patterns, are often interpreted as implementing a path integration mechanism in physical space [11]. However, their role in abstract cognitive tasks suggests a more general computational principle: the transformation of complex cognitive functions to path integrals in abstract spaces [4], [8]. This raises questions about the nature of these spaces and the mechanisms by which neural systems construct and navigate them. For instance, would grid-like representations emerge in non-spatial domains, such as conceptual or social "spaces," and if so what would be the driving factors in such spaces that induce this geometry?

The lack of meaningful grid or place cell activity underscores the importance of identifying appropriate base functions for neural computations. While hexagonal grid patterns are well-documented in biological systems, their emergence in artificial networks depends critically on constraints such as non-negative weights and recurrent connectivity [9]. Alternative base functions, such as those derived from dynamic field theory or Riemannian geometry, might offer more robust frameworks for modeling cognitive maps [12]. For example, Riemannian manifolds could provide a mathematical foundation for encoding curved or high-dimensional spaces, while dynamic field theory could capture the temporal dynamics of neural activity. Ultimately, it might be a matter of integrating the formalisms rather than proving one to be right.

The lack of meaningful behavior or ability to navigate effectively highlights the criticality of embodiment in cognitive modeling. The nervous system is a hierarchically organised network of layered and stacked feedback circuits sensorimotor loops to integrate sensory inputs with motor outputs, enabling adaptive behavior in dynamic environments [13]. In contrast, the current model treats sensory processing, spatial encoding, and motor control as largely independent modules.

At the same time, the dual-phase learning schedule, while conceptually sound, failed to produce meaningful representations or behavior. Adjusting the learning schedule to be more flexible and context dependent, rather than preset could enable the agent to generalize across tasks and environments and maintain a stable map structure, and might even be able to smoothly learn new maps based on this structure, as biological agents are thought to do [5].

The challenges encountered in this work have broader implications: While grid and place cells provide a compelling framework for spatial navigation, their extension to non-spatial domains remains to be properly integrated within the

conceptualisations of cognitive maps; Similarly, the role of embodiment and sensorimotor integration in shaping cognitive maps needs to be considered a stronger constraint, that might possibly make the difference between a realistic or malfunctioning model.

References

[1] D. Caligiore, M. A. Arbib, R. C. Miall, and G. Baldassarre, "The super-learning hypothesis: Integrating learning processes across cortex, cerebellum and basal ganglia," *Neuroscience & Biobehavioral Reviews*, vol. 100, pp. 19–34, May 2019, ISSN: 0149-7634. DOI: 10.1016/j.neubiorev.2019.02.008. (visited on 06/07/2023).

[2] J. M. Shine, "The thalamus integrates the macrosystems of the brain to facilitate complex, adaptive brain network dynamics," *Progress in Neurobiology*, vol. 199, p. 101 951, Apr. 2021, ISSN: 0301-0082. DOI: 10.1016/j.pneurobio.2020.101951. (visited on 01/29/2024).

[3] J. L. S. Bellmund, P. Gärdenfors, E. I. Moser, and C. F. Doeller, "Navigating cognition: Spatial codes for human thinking," *Science*, vol. 362, no. 6415, eaat6766, Nov. 2018. DOI: 10.1126/science.aat6766. (visited on 01/28/2025).

[4] K. L. Stachenfeld, M. M. Botvinick, and S. J. Gershman, "The hippocampus as a predictive map," *Nature Neuroscience*, vol. 20, no. 11, pp. 1643–1653, Nov. 2017, ISSN: 1546-1726. DOI: 10.1038/nn.4650. (visited on 01/28/2025).

[5] M.-B. Moser, D. C. Rowland, and E. I. Moser, "Place Cells, Grid Cells, and Memory," *Cold Spring Harbor Perspectives in Biology*, vol. 7, no. 2, a021808, Feb. 2015, ISSN: , 1943-0264. DOI: 10.1101/cshperspect.a021808. (visited on 01/28/2025).

[6] B. Komer and C. Eliasmith, "Efficient navigation using a scalable, biologically inspired spatial representation," *Proceedings of the Annual Meeting of the Cognitive Science Society*, vol. 42, no. 0, 2020. (visited on 01/21/2025).

[7] N. S.-Y. Dumont and C. Eliasmith, "Accurate representation for spatial cognition using grid cells," *Proceedings of the Annual Meeting of the Cognitive Science Society*, vol. 42, no. 0, 2020. (visited on 01/21/2025).

[8] N. S.-Y. Dumont, J. Orchard, and C. Eliasmith, "A model of path integration that connects neural and symbolic representation," *Proceedings of the Annual Meeting of the Cognitive Science Society*, vol. 44, no. 44, 2022. (visited on 01/30/2025).

[9] L. M. Giocomo, M.-B. Moser, and E. I. Moser, "Computational Models of Grid Cells," *Neuron*, vol. 71, no. 4, pp. 589–603, Aug. 2011, ISSN: 0896-6273. DOI: 10.1016/j.neuron.2011.07.023. (visited on 01/29/2025).

[10] T. Bekolay, J. Bergstra, E. Hunsberger, *et al.*, "Nengo: A Python tool for building large-scale functional brain models," *Frontiers in Neuroinformatics*, vol. 7, no. 48, pp. 1–13, 2014, ISSN: 1662-5196. DOI: 10.3389/fninf.2013.00048.

[11] B. Sorscher, G. C. Mel, S. A. Ocko, L. M. Giocomo, and S. Ganguli, "A unified theory for the computational and mechanistic origins of grid cells," *Neuron*, vol. 111, no. 1, 121–137.e13, Jan. 2023, ISSN: 0896-6273. DOI: 10.1016/j.neuron.2022.10.003. (visited on 01/28/2025).

[12] J. C. Pang, K. M. Aquino, M. Oldehinkel, *et al.*, "Geometric constraints on human brain function," *Nature*, vol. 618, no. 7965, pp. 566–574, Jun. 2023, ISSN: 1476-4687. DOI: 10.1038/s41586-023-06098-1. (visited on 07/20/2023).

[13] J. Merel, M. Botvinick, and G. Wayne, "Hierarchical motor control in mammals and machines," *Nature communications*, vol. 10, no. 1, pp. 1–12, 2019.

Listing 1: Cognitive Architecture

```
1
2  voja = nengo.Voja(lr)
3  pes = nengo.PES(lr)
4
5  # initiate model components
6  world = grid.World(Cell, map=large_map, directions=int(4))
7
8  body = grid.ContinuousAgent()
9  world.add(body, x=1, y=2, dir=2)
10
11
12 model = nengo.Network()
13 with model:
14
15     env = grid.GridNode(world, dt=0.005)
16     # -------------- Module ---------------
17     # sensors
18     proximity_sensors = nengo.Node(detect)
19     current_color = nengo.Node(cell2rgb)
20     ahead_color = nengo.Node(look_ahead)
21
22     # perceptual modules
23     color_cur = nengo.Ensemble(n_neurons=50, dimensions=3)
24     color_ah = nengo.Ensemble(n_neurons=50, dimensions=3)
25     walldist = nengo.Ensemble(n_neurons=50, dimensions=3)
26
27
28     # derivatives
29     ders = nengo.Ensemble(n_neurons=50, dimensions=3)
30
31     # grid cells with custom encoders
32     grid_cells = nengo.Ensemble(n_neurons=1000,
33                                 dimensions = 6,
34                                 encoders=nengo.dists.Uniform(0, 1).sample(1000, 6)
35                                             *np.random.choice([0.5, 1.0, 2.0],
36                                 size=(1000, 1)))
37
38     # place cells for uniquely representing location
39     place_cells = nengo.Ensemble(n_neurons=500, dimensions = 12)
40
41     # color sequence memory
42     col_seq_int = nengo.Ensemble(n_neurons = 1000, dimensions = 6)
43     col_place_enc = nengo.Ensemble(n_neurons=1000, dimensions = 18)
44     col_pred = nengo.Ensemble(n_neurons=500, dimensions = 3)
45     error_col = nengo.Ensemble(n_neurons=50, dimensions=3)
46
47
48     # basic action selection
49     cb = nengo.Ensemble(n_neurons = 50, dimensions=2)
50
51     # movement modules
52     expl = nengo.Ensemble(n_neurons = 50, dimensions = 1)
53     scan = nengo.Ensemble(n_neurons = 50, dimensions = 1)
54
55     mov_out = nengo.Ensemble(n_neurons = 50, dimensions = 2)
56
57     # Action Output
58     movement = nengo.Node(move, size_in=2)
59
```

```
60      # learning phases
61      gate_node = nengo.Node(gate_error, size_in = 4, size_out=3)
62      phase_node = nengo.Node(learning_phases, size_out = 2)
63
64
65      # -------------------- Connections ----------------
66      # Connections from sensors to sensory ensembles
67      nengo.Connection(proximity_sensors, walldist)
68      nengo.Connection(current_color, color_cur)
69      nengo.Connection(ahead_color, color_ah)
70
71      # recurrent connections for sensory stabilisation
72      nengo.Connection(color_cur, color_cur, synapse=0.02)
73      nengo.Connection(color_ah, color_ah, synapse=0.02)
74      nengo.Connection(walldist, walldist, synapse=0.02)
75
76      # double for computing derivative
77      nengo.Connection(walldist, ders)
78      nengo.Connection(walldist, ders, transform = -1, synapse=0.01)
79
80      # walldist and derivative to grid int
81      nengo.Connection(walldist, grid_cells[0:3])
82      nengo.Connection(ders, grid_cells[3:6])
83
84
85
86      # grid to place
87      conn_1 = nengo.Connection(grid_cells, place_cells[0:6])
88      conn_2 = nengo.Connection(color_cur, place_cells[6:9])
89      conn_3 = nengo.Connection(color_ah, place_cells[9:12])
90
91
92      # memory connections
93      nengo.Connection(color_cur, col_seq_int[0:3])
94      nengo.Connection(color_ah, col_seq_int[0:3])
95      col_seq_rep = nengo.Connection(col_seq_int, col_place_enc[0:6],
96                                     learning_rule_type=voja)
97      place_enc = nengo.Connection(place_cells, col_place_enc[6:18],
98                                   learning_rule_type=voja)
99      col_pred_enc = nengo.Connection(col_place_enc, col_pred, t
100                                     ransform = np.random.randn(3,18),
101                                     learning_rule_type = pes)
102
103
104     # prediction connections
105     nengo.Connection(col_pred, error_col)
106     nengo.Connection(color_ah, error_col,
107                      transform = -1,
108                      synapse=0.05)
109
110
111     # #action selection
112     nengo.Connection(error_col, cb[0],
113                      function = lambda x: np.linalg.norm(np.array(x)))
114     nengo.Connection(cb[0], expl[0], transform=-1)
115     nengo.Connection(cb[0], scan[0])
116
117     # movement connections
118     nengo.Connection(expl, mov_out, function=mov_expl)
119     nengo.Connection(scan, mov_out, function=mov_scan)
120     nengo.Connection(walldist, mov_out, function=movement_func)
```

```
121         nengo.Connection(mov_out, movement)
122
123         # movement recurrency to stabilise
124         nengo.Connection(mov_out, mov_out, synapse=0.005)
125
126
127         # learning phases connections
128         nengo.Connection(phase_node[0], place_enc.learning_rule)
129         nengo.Connection(phase_node[0], col_seq_rep.learning_rule)
130         nengo.Connection(phase_node[1], gate_node[0])
131         nengo.Connection(error_col, gate_node[1:])
132         nengo.Connection(gate_node, col_pred_enc.learning_rule)
```