RatInABox: A unified Python framework for modelling spatial behaviour and neural data

Summary Studying the brain's role in spatial navigation often necessitates creating *synthetic* behaviour and/or neural data. Without a common framework, this time consuming process raises the entry barrier to computational research and makes it difficult to reproduce or compare research which generates data in different ways. Here we present RatInABox, an open source Python toolkit for modelling locomotion and synthetic neural data from spatially modulated cell types. RatInABox provides users with (i) the ability to construct environments with configurable barriers, holes, visual cues and rewards, (ii) a physically realistic random motion model fitted to experimental data, (iii) fast online calculation of neural data for most of the known self-location or velocity selective cell types in the hippocampal formation (including place cells, grid cells, boundary vector cells, head direction cells etc.) and (iv) a framework for constructing bespoke complex cell types, multi-layer network models and data- or policy-controlled motion trajectories. Motion and neural models are spatially and temporally continuous and topographically sensitive to boundary conditions and walls. RatInABox was built to be intuitive, visual and easy to learn, supported by a set of diverse and well-documented tutorials. We hope this tool will significantly streamline computational research into spatial navigation.

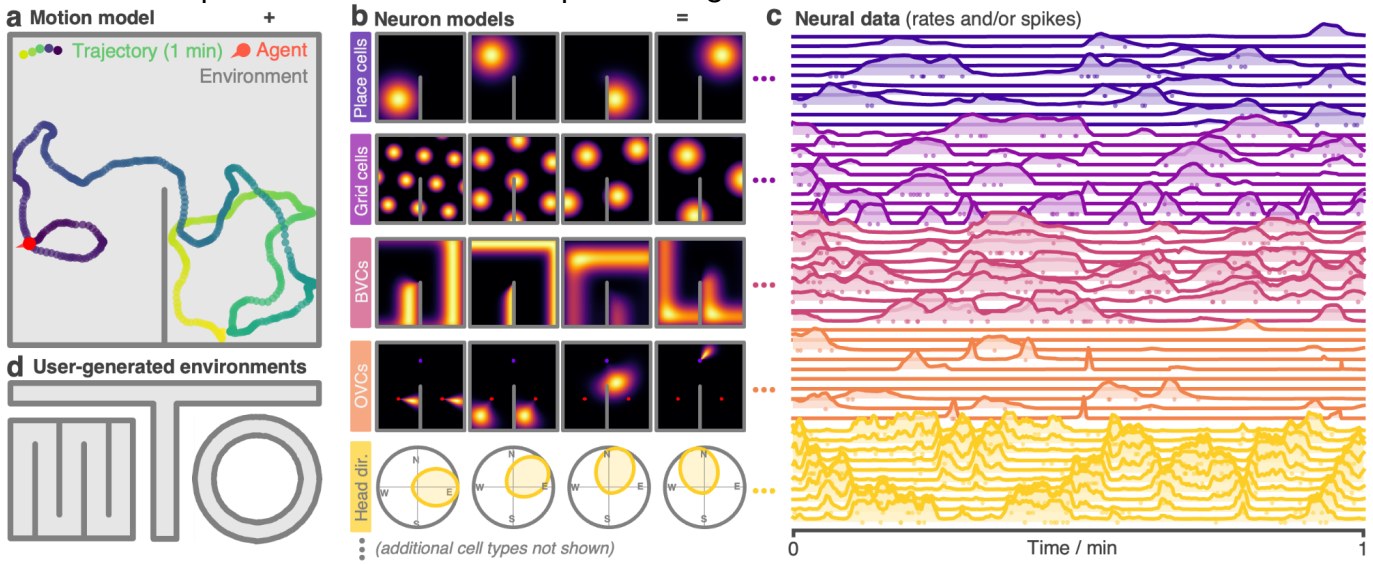

Figure 1: *RatInABox, a Python toolkit, generates synthetic spatial behaviour and neural data in continuous environments.* ***(a)*** *By default the agent follows a physically realistic random motion model, fitted to experimental data.* ***(b)*** *Premade neuron models include the most commonly observed position/velocity selective cells types (5 of many displayed here). Users can also build more complex cell classes based on these primitives.* ***(c)*** *As the agent explores the environment, neuron models generate neural data. This can be extracted for downstream analysis or visualised using in-built plotting functions.* ***(d)*** *Users construct environments by defining boundaries and adding walls/holes and objects. Three easy-to-make environments, chosen to replicate classic experimental set-ups, are shown.*

Details Generating artificial data and building computational models is a crucial part of neuroscience, particularly for studying spatial navigation. Currently the status quo has researchers write their own simulation code from scratch, an approach which lacks standardisation, wastes time, and raises the entry barrier to high-quality computational research.

RatInABox (RiaB, released this year) was built to overcome these issues. Users construct a 1D or 2D environment (Fig. 1d), populate it with one or more agents (Fig. 1a) and endow agents with neurons encoding their "state" in a rich variety of ways including cell types typically found in the hippocampal formation[1] (Fig. 1bc). As the agents explore, cells generate associated data which can be visualised using inbuilt plotting/animation functions and exported for downstream use.

Motion defaults to a smooth random walk, based on continuous-time stochastic processes with parameters fitted to real rodent locomotion data[2]. We scoured the literature for the most popular and influential cell models. Cells accept intuitive parameters (e.g. place cell size, grid scale etc.) with sensible defaults. RiaB currently contains efficient implementations of place cells, grid cells, head direction cells, velocity cells, speed cells, ego- and allocentric boundary/object/agent vector cells and random spatially tuned cells, as well as advanced function approximator cell types discussed below. Efficient protocols mean firing rates are calculated online (not precached) lowering memory requirements, accelerating simulations and improving accuracy.

In contrast to related packages[3,4] RiaB calculates motion and neural data in continuous time and space. This upgrade more accurately reflects real-world physics, making simulations smooth and amenable to fast or dynamic neural processes which are not well accommodated by discretised motion simulators e.g. grid world, or Markov decision processes.

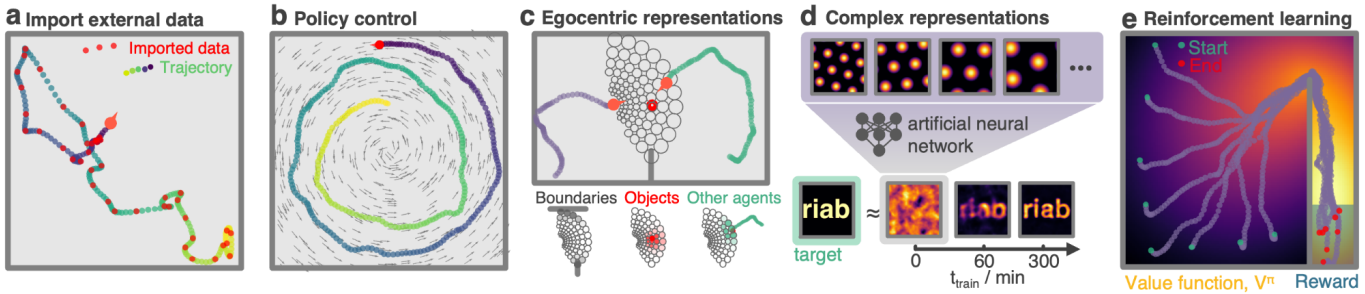

Figure 2: *Advanced functionality transitions RatInABox from a simple data generator into a comprehensive modelling framework.* ***(a)*** *Users can import their own behavioural data which will be smoothly interpolated enabling high-resolution simulations from low-resolution data.* ***(b)*** *Movement can be controlled by an external velocity signal. Here, quiver arrows show direction and magnitude of an exemplar circular drift policy.* ***(c)*** *Egocentric representation – including what the agent can see in its "field of view" – include those responsive to boundaries, objects and other agents.* ***(d)*** *Neurons containing a small embedded DNN receive grid cells as inputs and are trained over 300 minutes of random exploration to learn a non-trivial out function (here, arbitrarily, the letters "riab").* ***(e)*** *Many features are combined in a series of RL demos. Here an agent learns a value function (from place cell basis features) and uses it to control the agent's policy, navigating towards a hidden reward.*

From data generator to modelling framework RiaB has many use cases, one being as a standalone data generator (Fig. 1). Advanced features, some of which are described here, significantly extend its scope, elevating it into a versatile toolkit for constructing complex models and aligning the package with the modern direction of NeuroAI research[5].

Motion need not be random: instead users can import external behavioural data or control motion with a policy signal (Fig. 2ab). Egocentric cells (Fig. 2c) respond to boundaries, objects and agents in a head-centred reference frame and can be arranged in a "field of view". In addition to cells with static firing functions (see Fig. 1b), 'complex' cell types contain parameterized and learnable function approximators, e.g. linear or DNNs as in Fig. 2d, facilitating building cell types with mixed selective, highly-complex and even dynamic firing functions.

These features allow users to build closed-loop models of how behaviour and representations mutually interact. We provide a set of demos exploring potential use cases ranging from splitter cells, neural decoding and path integration, to reinforcement learning (successor features, deep actor-critic and linear RL). These examples are illustrative, not exhaustive. RiaB provides the framework and primitive classes/functions from which highly advanced simulations can be built.

A simple and modular API, supported by numerous tutorials, makes RiaB easy to learn and use. 10 lines of code are sufficient to build non-trivial data simulations (Fig. 3). Being open source and community driven, with a growing user-base, we hope that new features will continue to align with current research trends. The strength of RatInABox is its simplicity. It is not intended to address all problems within a specific field (e.g. RL[3]) but serves as a framework to power modelling more broadly. In this way RiaB democratises computational neuroscience and frees scientists to focus on cutting-edge research rather than software development.

```
1  %pip install RatInABox                              # Install RatInABox
2  from ratinabox.Environment import Environment       # Import RatInABox
3  from ratinabox.Agent import Agent
4  from ratinabox.Neurons import BoundaryVectorCells
5
6  Env = Environment(params={})                        # Default Environment
7  Env.add_wall([[0.5,0],[0.5,0.5]])                   # Add a wall
8  Ag = Agent(Env, params={})                          # Default Agent
9  BVCs = BoundaryVectorCells(Ag, params={'n':10})     # Default BVCs
10
11 while Ag.t < 60:                                    # Simulate (60 seconds)
12     Ag.update()
13     BVCs.update()
14
15 Ag.plot_trajectory()                                # Plot trajectory  (Fig. 1a)
16 BVCs.plot_rate_map()                                # Plot rate maps   (Fig. 1b)
17 BVCs.plot_rate_timeseries()                         # Plot neural data (Fig. 1c)
```

Figure 3: *This code replicates Fig. 1a-c (BVCs only) and shows a typical workflow. One minute of random exploration and 10 boundary vector cells are simulated. Data generation (lines 11-13) takes 0.67±0.01 seconds on a consumer-grade Apple M1 CPU.*

References **[1] Barry, C et al. (2014)** Neural mechanisms of self-location. **[2] Sargolini, F et al. (2006)** Conjunctive Representation of Position, Direction, and Velocity in Entorhinal Cortex. **[3] Juliani, A et al. (2022)** Neuro-Nav: A Library for Neurally-Plausible Reinforcement Learning. **[4] Domine, C et al. (2023)** NeuralPlayground: A Standardised Environment for Evaluating Models of Hippocampus and Entorhinal Cortex. **[5] Zador, A et al. (2023)** Catalyzing next-generation Artificial Intelligence through NeuroAI.