Re-Implementing a Dynamic Field Theory Model of Mental Maps using Python and Nengo

Rabea Turon (rabea.turon@uniklinik-freiburg.de)[1]
Paulina Friemann (friemanp@cs.uni-freiburg.de)[1]
Terrence C. Stewart (terrence.stewart@nrc-cnrc.gc.ca)[2]
Marco Ragni (ragni@informatik.uni-freiburg.de)[1]
[1]Albert-Ludwigs-Universität Freiburg, Germany
[2]National Research Council of Canada, University of Waterloo Collaboration Centre, Canada

Abstract

In Dynamic Field Theory (DFT) cognition is modeled as the interaction of a complex dynamical system. The connection to the brain is established by smaller parts of this system, neural fields, that mimic the behavior of neuron populations. We reimplemented a spatial reasoning model from DFT in Python using the Nengo framework in order to provide a more flexible implementation, and to facilitate future research on a more general comparison between DFT and the Neural Engineering Framework (NEF). Our results show that it is possible to recreate the DFT spatial reasoning model using Nengo, since we were able to duplicate both the behavior of single neural fields and the whole model. However, there are statistical differences in performance between the two implementations, and future work is needed to determine the cause of these differences.

Keywords: Dynamic Field Theory; Nengo; mental maps; model reimplementation; spatial relational reasoning

Introduction

The way humans build maps from descriptions of relations of objects to each other is not yet fully understood. One approach to model how these maps arise is the spatial reasoning architecture from Kounatidou, Richter, and Schöner (2018). Their model implements an application of the Dynamic Field Theory (DFT), which views cognition as the development of a complex dynamical system (Schöner, Spencer, & Group, 2016). The model can be supplied with sentences that describe the relative location of two colored objects in 2-dimensional space, e.g., "There is a cyan object above a green object". From that, it builds a spatial scene in a 2-dimensional space, represented by activity in a 2-dimensional sheet of simulated neurons. A more complex spatial scene with more objects can be built by supplying multiple sentences. The implementation of the spatial reasoning architecture is realized in the Graphical User Interface (GUI) framework cedar (Lomp, Zibner, Richter, Ranó, & Schöner, 2013).

In this paper, we present a re-implementation of this model from Kounatidou et al. (2018). Rather than using their graphical framework, we implement the various components using Python, and then use the neural modelling toolkit Nengo (Bekolay et al., 2014) to combine the components together. There are two main reasons for this endeavor: The cedar framework, and therefore the spatial reasoning model itself, is difficult to modify and to use in scenarios that are different to the one presented in Kounatidou et al. (2018). Its application to future research as a general-purpose cognitive model of spatial reasoning is therefore limited by its implementation. The second reason to re-implement the model in Nengo is to approach a more general comparison between NEF and DFT using the spatial reasoning architecture as an example.

In the following, we will first briefly describe DFT and the spatial reasoning architecture. We will then specify the cedar model components used in this architecture, and describe the reimplementation in Nengo. Lastly, we compare the two implementations using an examplary spatial scene.

The spatial reasoning model

Background

The assumption of Dynamic Field Theory (DFT) is that cognition and behavior arise from the brain's development as a complex dynamical system (Schöner et al., 2016). Attractors in this system are then "functionally significant states of cognitive processes". An example for such an attractor of the spatial reasoning architecture is the spatial scene that results after supplying it with relational information.

The complex system in DFT consists of many subparts which DFT calls neural fields. They model the activation of populations of neurons. A complex system like the spatial reasoning architecture consists of multiple neural fields and some additional computations between these. Neural fields themselves are dynamical systems, too, whose dynamics are described by a differential equation:

$$\tau\dot{u}(x,t) = -u(x,t) + h + s(x,t) + \int k(x-x')g(u(x',t))dx' \tag{1}$$

In this equation $u(x,t)$ is the activation of the neural field at location x and time point t, h is the resting level of the neural field, $s(x,t)$ is some external input to the field, and integral computes local interactions in the field. The neural fields described by this equation can be of different dimensionality. What they have in common is that they can form peaks of activation that get passed on to other neural fields and can lead to further peaks there.

To make the building of complex dynamical systems as easy as possible DFT researchers have built software that helps with this task. In the case of the spatial reasoning architecture this software is cedar, a graphical-user-interface where computational elements can be added to an architecture with a simple drag and drop interface (Lomp et al., 2013). In addition to neural fields other elements can be added to the architecture, like inputs to the system or projections from a lower to a higher dimensional space.

Importantly, the spatial reasoning architecture developed in Kounatidou et al. (2018) was not created with the current version of cedar and does not perform correctly in the current version. The version that was used for this project can be found in an article by the Autonomous Robotics Group (2018). Models created in cedar can be saved to and loaded from JSON files.

The architecture

An overall image of the spatial reasoning architecture from Kounatidou et al. (2018) can be found in Figure 1. It consists of five conceptually distinct parts.

The first part of the architecture deals with the concepts that can be activated by the user. These are the spatial relations and the objects that are placed in a scene. The architecture is able to represent up to five different objects which are identified by their color, i.e. a red object, a blue object, a cyan object, a green object and an orange object. The translation from these colors to a continuous space is implemented by mapping them to the hue dimension. For all objects two input nodes exist since in each supplied sentence it has to be specified if an object is the reference object of the sentence or the target object. There are four spatial relations corresponding to the cardinal directions *Left*, *Right*, *Above* and *Below*.

The second part of the architecture is the attentional system. This system is responsible of ‘attending to’ or activating objects (i.e., the colored objects described earlier) that are already in the scene or that should be added to the scene in a new place, depending on the interaction with other neural fields from outside the attentional system. It consists of the color attention field and the 3-dimensional attention field and forms peaks for objects that are attended.

The scene representation forms the third part of the architecture. This includes the scene representation field where the locations of the objects are depicted over two-dimensional space while the third dimension of the field depicts the color of the existing objects.

The fourth part of the architecture is the spatial transformation and object creation system. It enables the architecture to place objects according to the relational premises that are supplied to the system. It represents each part of a premise, i.e. the reference object, the target object and the relation in a separate neural field and performs the transformations that are necessary to place a new object in the scene.

The fifth part of the architecture is responsible for the organization of all processes. It consists of intention nodes that determine whether a process is currently active or not and of condition-of-satisfaction (CoS) nodes that represent whether a process has been finished successfully. A more detailed description of the five parts can be found in the original paper.

Model components in cedar

A complex model in cedar is built from a set of basic components. The following components are needed for this model:

- `NeuralField`: This module implements the neural field equation from dynamic field theory. After an update step with the neural field equation a sigmoid function is applied to the activations before passing them on to another module.
- `GaussInput`: A GaussInput module is one of the input modules of cedar. This means that it does not receive any input but constantly sends the same signal. In the case of the GaussInput this signal is a two-dimensional gaussian peak whose peak position and amplitude are defined as parameters of the module.
- `ConstMatrix`: This module is another input module. It sends a constant 2-dimensional matrix output with one constant value at all positions of the matrix.
- `SpatialTemplate`: The SpatialTemplate is another input module. With the right parameters it creates a funnel-like pattern directed towards one of the cardinal directions *right*, *left*, *above* or *below*.
- `Projection`: The Projection module projects an input to a different dimensionality. It can either upscale an input from a lower dimension to a higher dimension by repeating values along the new dimension or it can downscale an input from a higher dimension to a lower dimension by performing a compression along the dimensions that should be reduced. As a compression operation the sum, maximum, minimum or average along a dimension can be used.
- `StaticGain`: The StaticGain module multiplies an input by a constant value that can be set as a parameter.
- `Boost`: The Boost module is another input module. It sends a scalar value that can be set as its strength parameter. However, it can be set to being active or not active during a simulation. Depending whether it is active or not it either sends no signal or the strength value defined. The Boost module is the module through which changing input is supplied to a system.
- `ComponentMultiply`: The ComponentMultiply module performs a componentwise multiplication of two inputs. If the two inputs have exactly the same shape this is an elementwise multiplication. Otherwise one of the inputs has to be of a lower dimensionality and each of its values is then multiplied with the other input’s values along the additional dimension.
- `Convolution`: The Convolution module takes two inputs and performs a convolution on one of the inputs with the other input as the kernel.
- `Flip`: The Flip module receives a two-dimensional input and flips it along the first dimension, the second dimension or both.
- `Group`: A container to organize other components.

Nengo

Nengo (Bekolay et al., 2014) is a software tool that was originally created to build and simulate large-scale neural models based on the Neural Engineering Framework (NEF) (Eliasmith & Anderson, 2003). More recently, the toolkit has

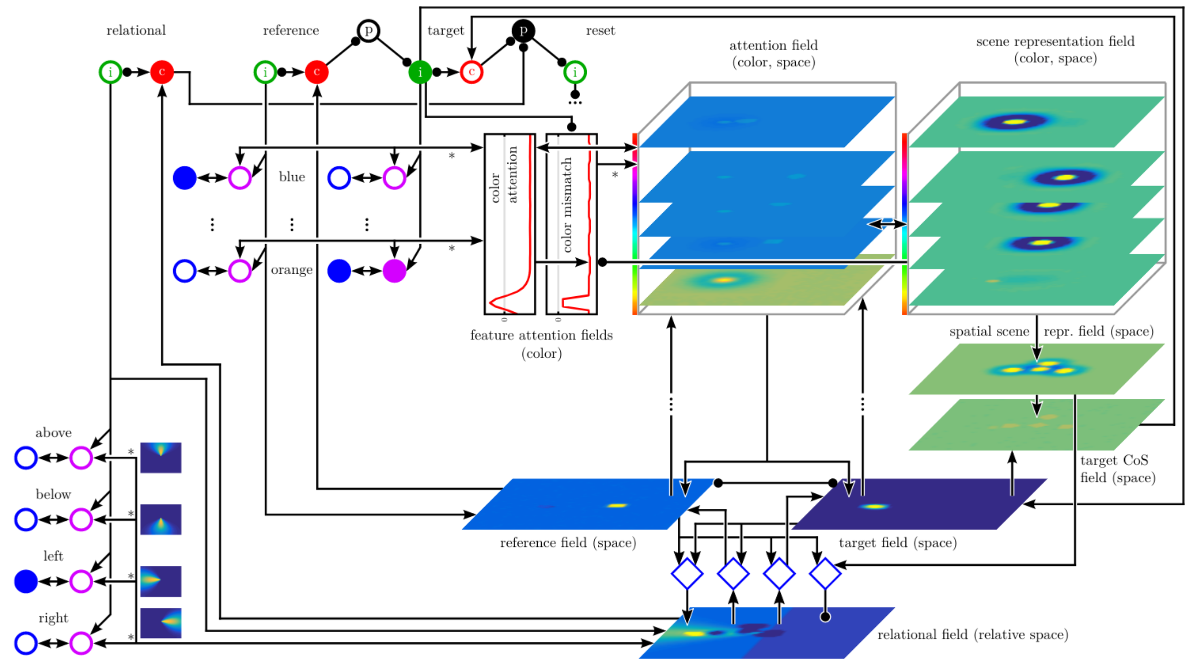

relational
reference
target
reset
attention field (color, space)
scene representation field (color, space)
blue
orange
color attention
color mismatch
feature attention fields (color)
spatial scene repr. field (space)
target CoS field (space)
above
below
left
right
reference field (space)
target field (space)
relational field (relative space)

Figure 1: A conceptual image of the DFT spatial reasoning architecture from Kounatidou et al. (2018) reprinted from the original paper. The blue and pink circles signify nodes that represent the input concepts of the architecture. The attentional system consists of the feature attention fields in the middle and the attention field to the right of these. The scene representation fields can be seen on the right of the image. At the bottom the fourth part of the architecture, the *spatial transformation and object creation system* can be seen. The intention nodes are depicted as the red circles at the top, the CoS nodes as the green circles.

expanded to support deep learning and vector-based cognitive modelling, making Nengo now support a wider range of modelling approaches. Most importantly, Nengo provides a simple Python interface for defining new components, and then the standard Nengo framework can be used for combining these components, running simulations, and gathering data. By constructing our re-implementation in this way, we can to run any cedar model that uses the components we have re-implemented simply by taking the cedar version, saving it as a JSON file, and loading that saved JSON file into Nengo.

Model components in Nengo

The Nengo implementation of the cedar modules is based on the Nengo `Node` object. Some of the cedar modules (e.g. the `NeuralField`) require an update of their state with each simulation step while others (e.g. the `GaussInput`) pass on a constant signal during the whole simulation. The `Node` object provides easy-to-use functionality for both of these options. In the first case one can define an update function that depends on time and initialize a `Node` instance with this update function as the output parameter. In the latter case the constant signal is simply passed to a `Node` instance as the `output` parameter. In the implementation of the cedar modules in Nengo each cedar module has a corresponding object class of the same name. After initialization a Nengo `Node` instance can be created from this class's instance by calling the `make_node()` method. The `Node` instance is then accessible via the `node` attribute.

The Nengo implementation of the cedar modules is based on Schöner et al. (2016), as well as on the cedar documentation and the cedar source code (Autonomous Robotics Group, 2018). Another source of information is the behavior of the modules in cedar. For each module test instances of the cedar modules were created to observe their behavior for different parameter settings or different inputs. The observed behavior of the modules is also the principal validation for a correct implementation.

Since the goal was to implement the spatial reasoning architecture, not all of cedar's functionality had to be implemented. This means only the cedar modules that are part of the spatial reasoning architecture were implemented in Nengo. Moreover, some parameters of the cedar modules were not implemented in Nengo if they are not used in the spatial reasoning architecture or if their value is constant among all instances of the spatial reasoning architecture.

Nengo Results

Verification of NeuralField implementation

To make sure that the neural field equation is implemented correctly the temporal development of the `NeuralFields` of cedar and Nengo were compared. To examine different parameters (e.g. different dimensionality or different border types) several `NeuralFields` with gaussian input were created and compared visually. For the visual comparison color maps of the `NeuralFields'` activation, their lateral interaction and their sigmoided output were created for both the cedar and the Nengo simulations. The appearance of the color maps and the mininum and maximum values were used as a measure of resemblance. To avoid random fluctuations the noise values were set to zero in these comparisons.

Apart from the identical evolution of the neural fields in terms of activation values the timing of this evolution was another tested aspect. To compare the timing the maximum activation value for the comparisons from above was tracked and contrasted in a plot. Moreover this comparison was performed for several different values of `tau`. Figure 2 contains one such comparison for three different tau values. As can be seen, each pair of curves with the same configuration progresses similarly through time, suggesting that the timing of the two implementations is the same.

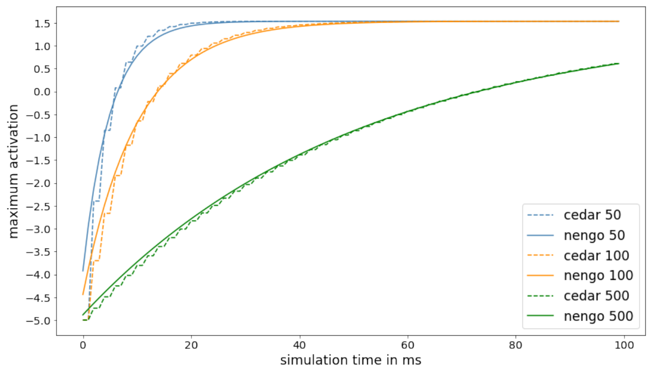


Figure 2: Maximum activation of a test NeuralField for different tau values in cedar and Nengo.

One Spatial Scene in Detail

Kounatidou et al. (2018) gives one example for a spatial scene that can be created with the spatial reasoning architecture. The scene is created from four successively supplied sentences:

1. There is a cyan object above a green object.
2. There is a red object to the left of the green object.
3. There is a blue object to the right of the red object.
4. There is an orange object to the right of the red object.

This scene was used as a test to see if the whole spatial reasoning architecture in Nengo works and to compare the scenes that arise in the original implementation in cedar and in the Nengo implementation. Figure 3a shows the development of the scene in cedar. The precise temporal sequence of inputs needed to create this scene is given in Table 1. The same experiment structure was used in the Nengo simulations.

In Figure 3b the development of the scene in Nengo can be seen. Even though the scenes of cedar and Nengo are slightly different, overall scene arrangement and development are the same. Differences in the development of the scene are also normal in separate runs in cedar due to additive random noise in the neural fields.

Simulation time

While our results show that the Nengo version of the model works, our initial Python implementation is much slower than the cedar version. When running in cedar, there is a “Factor for the fake DT” (default 0.26) which controls the time resolution of the simulation of the dynamic equations. In Nengo, the default time resolution is 1ms. This meant that a simulation which takes 2.3 minutes in cedar took 180 minutes in Nengo, i.e., a speed factor of .013.

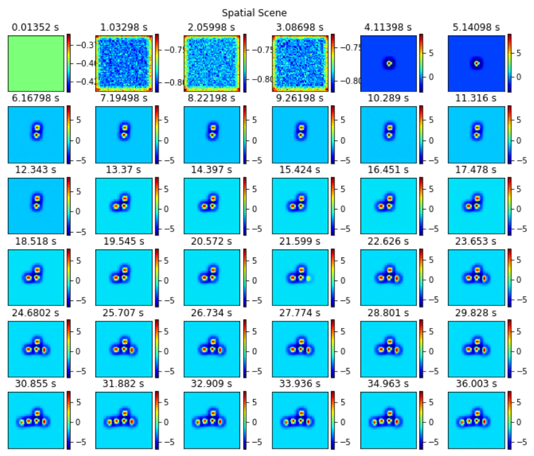
While Nengo does allow components to define their own adaptive time step, we have not yet implemented this. Instead, we adjust the time step by a factor `tau`, where `tau`=1 is the original (1ms per time step) and tau=0.01 would be 100ms per time step. Note that this is the same as the `tau` parameter in the NeuralFields definition. If `tau` is decreased, this leads to an increase in the step size because the tau parameter is the divisor of every update step. This adaptation can not be performed up to any arbitrary factor since the update steps are a discretization of a continuous time process and at some point this discretization is too inaccurate to capture the original development. To determine a stable value for the factor of tau at least five simulation runs of the test scene were run for different tau factors. The rate with which a `tau` factor led to the scene predicted by the cedar model and the simulation times for different `tau` factors can be seen in Table 2. The test with the different `tau` factors also revealed that the standard update size of 1 does not reliably lead to the correct scene but seems to be rather unstable since it only lead to the correct scene in 2 out of 9 runs.

As can be seen in Table 2, simulations with a tau factor below 0.15 do not always lead to the correct scene representation. Some of these failed simulations are depicted in Figure 4. For these simulations it is likely that the update steps are too big and processes that would go in the opposite direction as the previous step can not be integrated due to the few updates. However, there are `tau` factors smaller 1 for which the scene seems to reliably develop correctly and which therefore can provide a time improvement.

We are still looking into other optimizations that we believe could help speed up the Python implementation of the DFT equations, which are most of the computation time in this model. For all subsequent experiments the `tau` factor of 0.15 was used, since it provided the fastest simulation times while maintaining high confidence to result in the correct scene.

Testing Results

To explore the behaviour of our re-implementation of the DFT mental map model, we generated a set of test inputs

(a) 

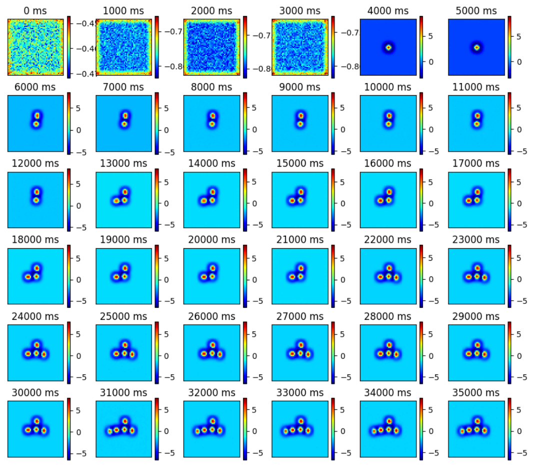
(b) 

Figure 3: Evolution of the spatial example scene in cedar (a) and Nengo (with updates every ms) (b) with the default parameters.

Table 1: Instructions from the cedar experiment file.

simulation time	actions
0.0s	Activate the Boost modules ”Reference: Green”, ”Spatial relation:Above” and ”Target:Cyan”.
0.5s	Deactivate the Boost modules ”Reference: Green”, ”Spatial relation:Above” and ”Target:Cyan” and activate the Boost module ”Action:Imagine”.
9.0s	Activate the Boost modules ”Reference:Green”, ”Target:Red” and ”Spatial relation:Left”.
9.5s	Deactivate the Boost modules ”Reference:Green”, ”Target:Red” and ”Spatial relation:Left”.
18.0s	Activate the Boost modules ”Reference:Red”, ”Target:Blue” and ”Spatial relation:Right”.
18.5s	Deactivate the Boost modules ”Reference:Red”, ”Target:Blue” and ”Spatial relation:Right”.
27.0s	Activate the Boost modules ”Reference:Blue”, ”Target:Orange” and ”Spatial Relation:Left”.
27.5s	Dectivate the Boost modules ”Reference:Blue”, ”Target:Orange” and ”Spatial Relation:Left”.
36.0s	End the experiment.

Table 2: Simulation times and success rate of the test scene for different tau factors in Nengo.

tau factor	simulation time	success rate
0.02	3.5 minutes	0/7
0.05	9 minutes	3/10
0.1	17 minutes	9/11
0.15	26 minutes	5/5
0.2	35 minutes	8/8
0.5	1.5 hours	6/6
1.0	3 hours	2/9

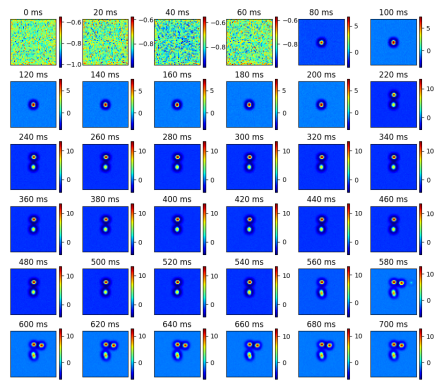
describing from two to four relational premises with all combinations of directions to generate a resulting scene. These inputs were run in both cedar and Nengo for comparison (see Figure 5 for examples).

Given these input scenes, we measured the proportion of time the models generated a correct final representation, i.e., the representation predicted by the cedar model. We knew that for some of the scenes the model would not create a scene consistent with the input statements due to the phrasing of the statements. However, they still gave us some information about the workings of the models. Since the models introduce random variability, we ran each input multiple times. Interestingly, the Nengo implementation was found to be more reliable than the cedar version; the Nengo simulations resulted in a correct scene in 73.26% of its simulations while the cedar simulations lead to the right scene in only 50.23% of its simulation runs. Determining the cause of this difference is the topic of ongoing work.

It should also be noted that for each test input, there was always at least one simulation run in cedar that resulted in the same mental map as a Nengo simulation run. This indicates that the models are doing similar things in those runs. For this reason, we believe that the core Python re-implementation in Nengo is working correctly, but that there are subtle differences with time steps and noise that are causing the differences in behaviour.

Conclusion and Future Work

Our goal was to reimplement the spatial reasoning architecture from Kounatidou et al. (2018) in the Python framework Nengo (Bekolay et al., 2014). The findings from the results section suggest that this goal is achieved, in that the system produces the desired behaviour. However, there are significant differences between the implementations that may be


(a) Tau factor 0.02

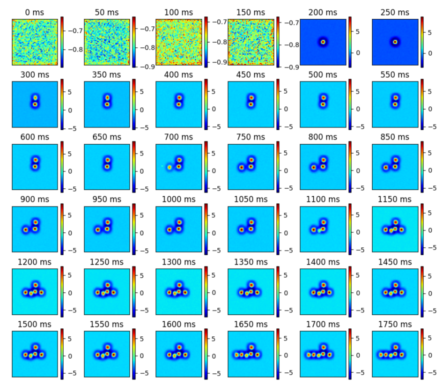

(b) Tau factor 0.05

Figure 4: Failed scene evolution of the example scene (see Table 1) in Nengo for different tau factors. The shown evolutions are only exemplary and can be different in each run.

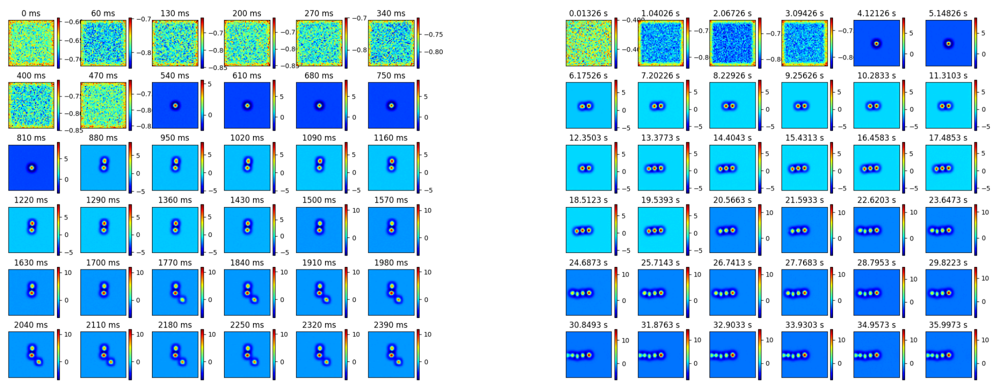


Figure 5: Example scenes from our test datasets. The scene development on the right is from a cedar simulation. The scene on the left is from a Nengo simulation.

due to the precise details of the random noise and the time steps used for calculation.

Importantly, since these low-level implementation details affect the overall performance of the model (as seen in the variability in the testing), understanding exactly what is causing these differences is important for interpretting any DFT model. We intend to continue to analyze these details and characterize them.

References

Autonomous Robotics Group, R.-U. B., Institut fuer Neuroinformatik. (2018). *Cedar 2018.* `https://github.com/cedar/cedar/tree/eeda0d6f79f5a0e420a877c642ce1b9ff48ba8dd`. GitHub.

Bekolay, T., Bergstra, J., Hunsberger, E., DeWolf, T., Stewart, T. C., Rasmussen, D., . . . Eliasmith, C. (2014). Nengo: a python tool for building large-scale functional brain models. *Frontiers in neuroinformatics*, *7*, 48.

Eliasmith, C., & Anderson, C. H. (2003). *Neural engineering: Computation, representation, and dynamics in neurobiological systems*. MIT press.

Kounatidou, P., Richter, M., & Schöner, G. (2018). A neural dynamic architecture that autonomously builds mental models. In *Cogsci.*

Lomp, O., Zibner, S. K. U., Richter, M., Ranó, I., & Schöner, G. (2013). A software framework for cognition, embodiment, dynamics, and autonomy in robotics: cedar. In *International conference on artificial neural networks* (pp. 475–482).

Schöner, G., Spencer, J. P., & Group, D. R. (2016). *Dynamic thinking: A primer on dynamic field theory*. Oxford University Press.