A neural dynamic architecture for goal-directed reaching

Virtual DFT Summer School 2020

Goal of the Project

The goal of this project is to implement a dynamic field architecture that enables a robot to reach for objects in its visual field. The architecture creates a velocity signal for the arm based on the relative translation between the endeffector of the arm and the reaching target. We will use the Visual Search architecture from the previous exercise to determine the reaching target. If you have not completed the Visual Search architecture with the Group *Camera Source* as input, please do so before continuing.

Architecture Components

The essential part of the architecture consists of three fields (see Figure 1):

- **Reaching Target**: This field is defined over the same two spatial dimensions as the *Spatial Attention* field. A peak in the field represents the target location of the endeffector. This field has a selective kernel as there should be only one target at all times.
- **Endeffector Position**: The current position of the endeffector (EEF) is represented in this field. It shares the same spatial dimensions as the reaching target field.
- **Relative Target**: This field represents the target location in a different relative coordinate frame centered on the EEF. A peak in this field represents the target location with respect to the EEF. For example in Figure 1 the peak in the top center location of the Relative Target fields encodes that the target is located upwards of the endeffector. The spatial dimensions of the field are twice the size of the other two fields to capture all possible target/EEF variations.

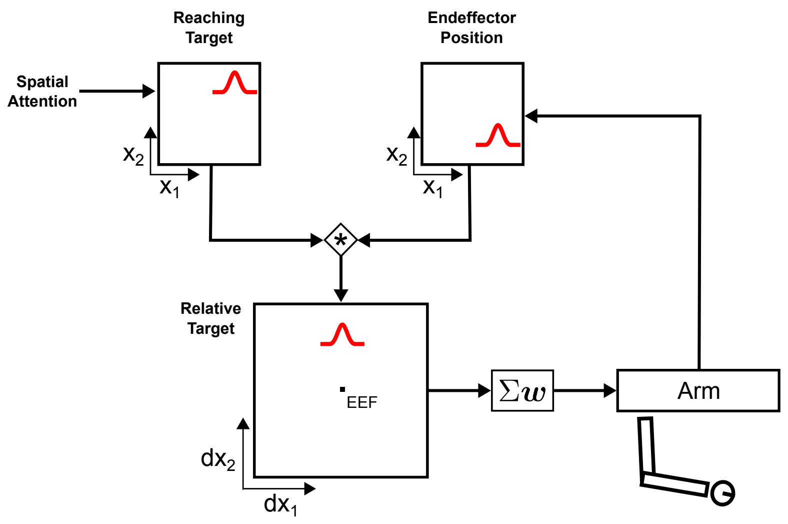

Figure 1: Architecture Sketch.

Implementation

- **Reaching Target:** The Visual Search architecture selects all objects in the visual scene that share a cued feature. To boil down this selection to a single reaching target, create a new `NeuralField` and name it *Reaching Target.* It needs to share the same spatial dimensions as the *Spatial Attention* field of the Visual Search architecture. Connect the latter to the new field and tune the field's parameters to achieve a selective regime.

  ➤ Verify your parameter tuning by choosing different color cues. Each time only a single peak should emerge in the new *Reaching Target* field.

- **Arm Interface:** Your architecture template contains two `Groups` that relate to the arm. The *Arm Motor* group contains the arm's kinematic chain and may take two-dimensional velocity commands. The *Arm Proprioception* group provides a two-dimensional spatial representation of the arm's endeffector. To see the components in a group right-click on it and toggle *collapse* in the menu.

  The main components of the *Arm Motor* group are the *VirtualArm* `Component` itself and a `PseudoInverse-Kinematics` step. The latter transforms Cartesian- into joint-velocities, which is a non-neural approach borrowed from robotics. It acts as a placeholder until a feasible neural implementation is available. For now we are only interested in the *VirtualArm* component. Use the right-click menu (actions $\rightarrow$ apply…) to set the arm to one of the predefined starting positions. To get a better look at the simulated arm select the *visualisation* tab of the main canvas.

  The *Arm Propioception* group is the input-interface to your architecture. Open a plot of the current endeffector position by opening the group's right-click menu and clicking on "plot all".

  ➤ Inspect how the endeffector position varies, as you change to different arm positions.

- **Endeffector Position:** Create a new `NeuralField` and name it *Endeffector Position.* Make sure that it shares the spatial dimensions of the *Arm Proprioception's* output and connect it to the field. Tune the field parameters until a single peak emerges at the eef position.

  ➤ Verify that the peak follows changing input by again varying the arm's position using the predefined starting positions.

- **Coordinate Transformation:** With representations of the target and the endeffector in place, it is now possible to create the input to the *Relative Target* field. The reaching target has to be transformed in a new coordinate system centered on the endeffector. To keep this implementation simple we will use a cross-correlation to achieve this transformation (see [2] for a neural approach).

  In cedar a cross-correlation can be implemented using a `Flip` and a `Convolution` step. Create a `Flip` step and connect the output of the *EEF Position* field to it. The transform is then performed with a `Convolution` step. Create it and connect the *Reaching Target* field to the "matrix" input and the output of the Flip step to the "kernel" input of the Convolution step. Select the Convolution step and change the convolution mode from "same" to "full" to account for all locations.

  ➤ Verify that the transformation works by observing the output of the Convolution step as you select different targets and/or endeffector positions.

- **Relative Target:** Create a new `NeuralField`. Set its spatial dimensions to the output of the Convolution step and name it *Relative Target.* If you chose the mode "full", it should be $2N - 1$, where N is the size of your other fields. Connect the output of the Convolution to the field and tune its resting level until a supra-threshold peak of activation emerges in the field. Set the field's sigmoid parameter to "SemiLinearTransferFunction" to ensure that below threshold regions have an output value of 0.

The output of the *Relative Target* may now be used to generate a velocity command for the arm. To this end we will use a fixed weight matrix that assigns a particular velocity value to each location in the *Relative Target Field*. For example locations in the left half of the field are assigned a positive velocity in x_1 direction, while locations on the right half are assigned a negative velocity. The center location is assigned a velocity of zero, because a peak in the center encodes that the target is at the same location as the endeffector.

Create a new `WeightedSum` step from the Tab "Algebra", which implements the desired weight matrix and sums across it. Match its dimensionality and size parameters to the *Relative Target* field and connect it to the step. Its output is a two-dimensional vector encoding velocity in x_1 and x_2 direction. To vary the amplitude of the velocities, create a `StaticGain` step and connect it to the output of the WeightedSum. For safety reasons choose a gain factor of 0.1 to start with an initially low velocity.

Now you are ready to move the arm. Connect the StaticGain to the input of the *Arm Motor* group. If there is a peak in the *Relative Target* field the arm should move towards the selected target.

- ➤ How is the target representation affected by the arm as it occludes the target? Can you retune the *Reaching Target* field to change this behavior?
- ➤ Again play around with different target configurations and different color cues to verify the correct tuning of the architecture.

Extending the Architecture

We propose two possible ways to extend this project. Staying close to the reaching architecture one could implement means of behavioral organisation, which may lead to the autonomous execution of action sequences. Or one could extend the visual front-end to incorporate spatial language, which could lead to more sophisticated target selections.

1. **Generating Action Sequences:** The current architecture needs to be extended before one can start with the Sequences project. We need to implement a pair of neural nodes that represent, whether the current action is active or reached its terminal state: The intention and Condition-of-Satisfaction node (see [1] for details).

   - ➤ To continue with this extension refer to the implementation hints on the next page.

2. **Extending Target Selection through Spatial Language:** This project extends the Visual Search architecture to incorporate the grounding of spatial language. Instead of selecting one of multiple blue targets one could specify target selection through spatial language phrases such as *"Select the blue object to the right of the red object"*.

   - ➤ To continue with this extension download the exercise sheet "A neural dynamic architecture for the perceptual grounding of simple spatial language" from our homepage.

References

[1] Mathis Richter, Yulia Sandamirskaya, and Gregor Schöner. A robotic architecture for action selection and behavioral organization inspired by human cognition. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2457–2464. IEEE, 2012.

[2] Sebastian Schneegans and Gregor Schöner. A neural mechanism for coordinate transformation predicts pre-saccadic remapping. *Biological cybernetics*, 106(2):89–109, 2012.

Adding Behavioral Organisation

You are now hopefully a bit more familiar with building neural architectures with `cedar` and therefore this part of the project is explained in less detail. The architecture is shown in Figure 2 and its main components are:

- **Intention Node**: Activation of this node boosts the *Reaching Target* field causing it to form a peak, which causes the arm to reach for it.
- **Target-EEF Match field**: This field receives input from the *Reaching Target* and the *EEF Position* field. It only forms a supra-threshold peak whenever the current reaching target and the eef position overlap.
- **CoS Node**: Any supra-threshold peak in the *Target-EEF Match* field activates this node, which signals the termination of the reaching action. It inhibits the *Intention* node, which causes a deselection of the current target.

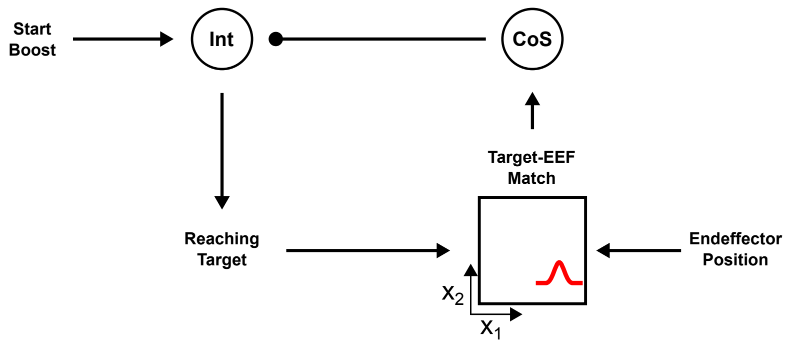

Figure 2: Extension of the architecture shown in Figure 1.

Implementation Tips:

➤ Later we will activate the *intention node* through other parts of the architecture. For now you can use a `Boost` step to easily set its input.

➤ Change the resting level of the *Reaching Target* field such that it only forms peaks upon activation of the *intention node*. As long as the *intention* node is active, peaks should sustain in the *Reaching Target* field without visual input.

➤ Connecting the *CoS* node to the *intention* node should be the last step, because this closes a loop. First verify that you detect the termination of a movement.

➤ To prevent selection of a new target you can add self-excitation to the *CoS* node. To allow it to deactivate again, change its resting level and also connect it to the boost node. Deactivation of the boost node should deactivate the *CoS* node.

Next Steps:

Proceed to the exercise on "Learning and Replaying Sequences", if activation of the `Boost` leads to the following:

1. The *intention* node activates and causes the selection of a reaching target, which leads to a subsequent reaching movement.
2. Once the eef reaches its target location, a peak forms in the *Target-EEF Match* field, which causes the *CoS* node to activate. This deactivates the *intention* node through inhibition and the peak in the *Reaching Target* field vanishes.