CAUSAL COGNITIVE ARCHITECTURE 3 (CCA3): A SOLUTION TO THE BINDING PROBLEM

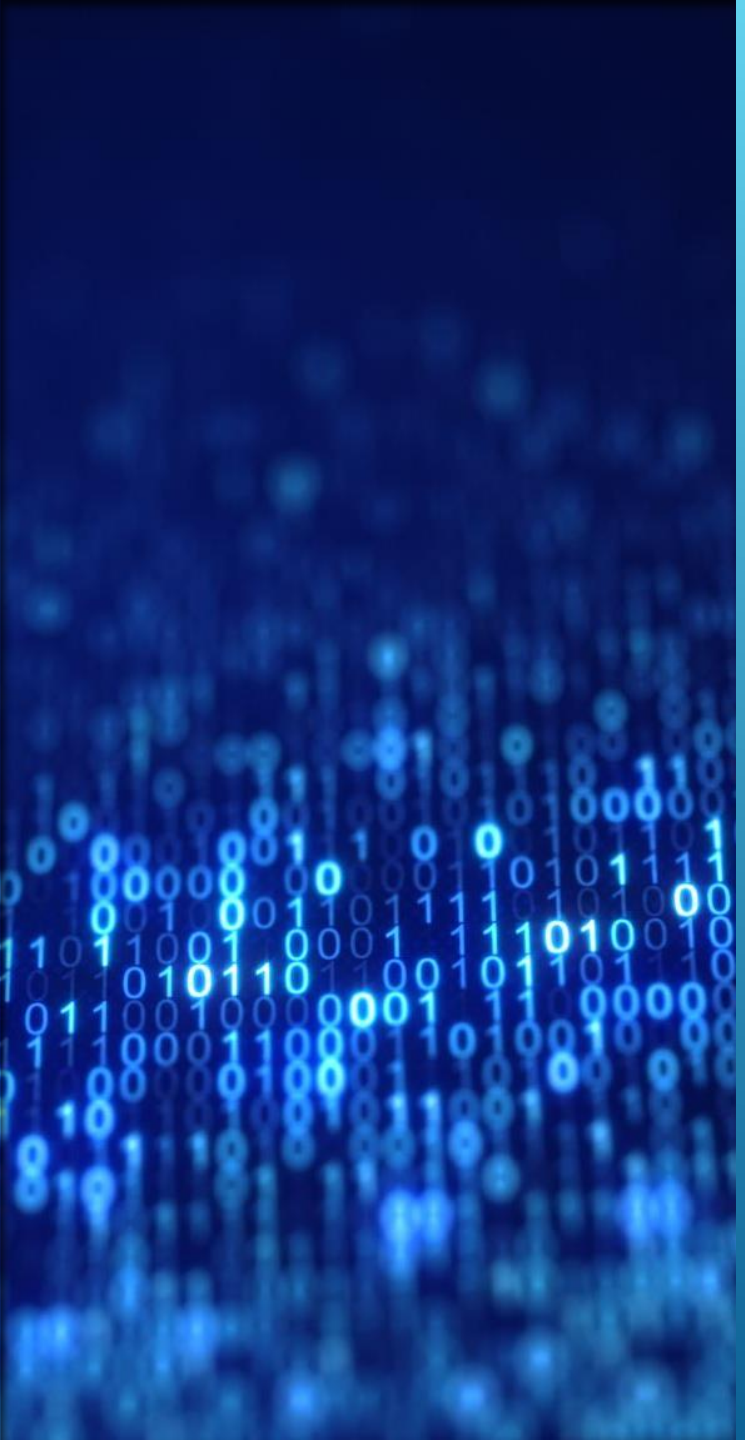
Howard Schneider

Sheppard Clinic North, Ontario, Canada

Cognitive Systems Research, in press
Supplementary Video File

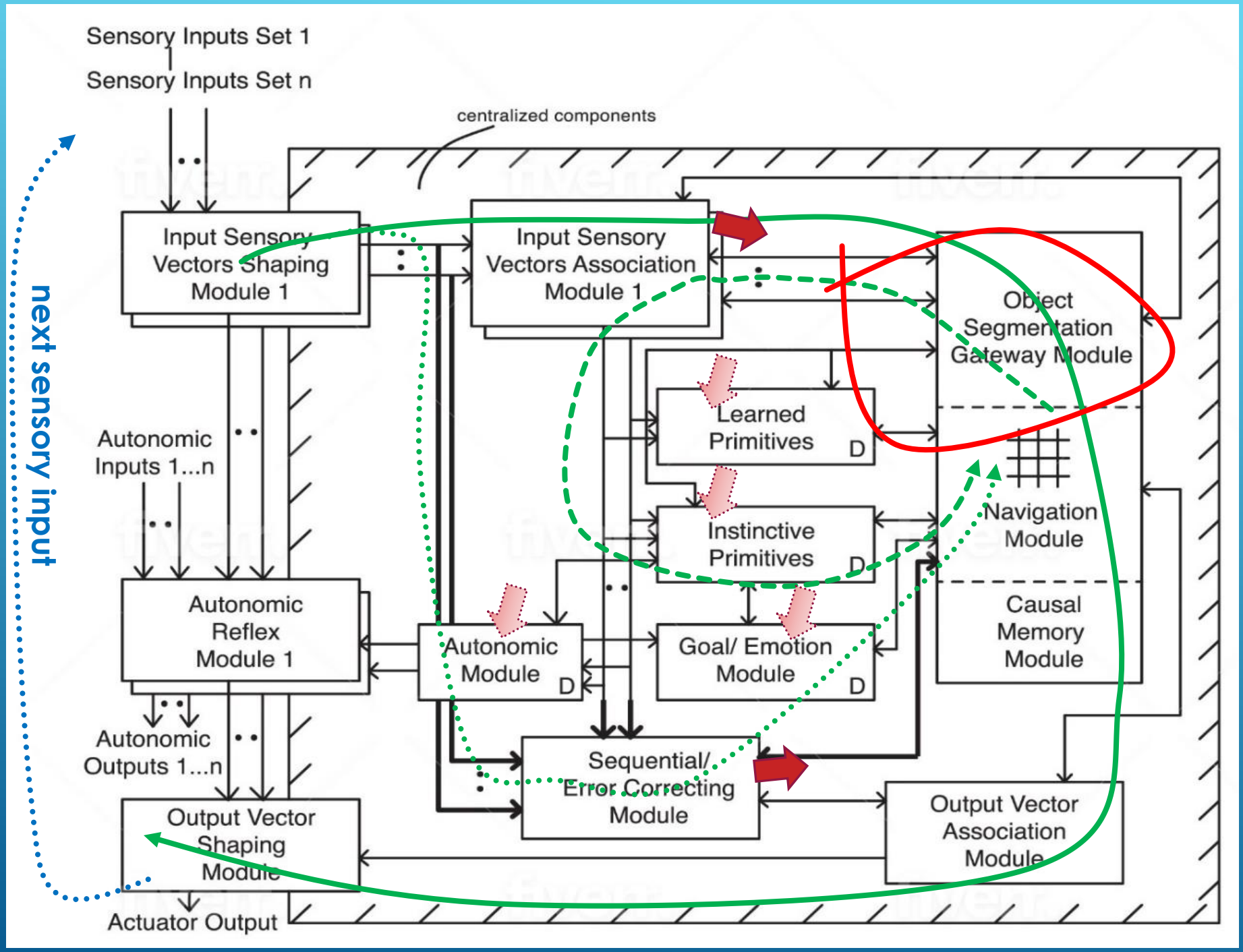
GITHUB Username: "CausalCog"
https://github.com/CausalCog

VIDEO # 5



- CCA3 Overview ✔
- Binding Problem Overview ✔
- Software Overview ✔
- Operations Overview ⬅ ⬅ ⬅
- Operations Causal
- Software in More Detail
- More videos, code on GitHub "CausalCog"

(If interest, continued updating on GitHub)


Sensory Inputs Set 1
Sensory Inputs Set n
centralized components
next sensory input
Input Sensory Vectors Shaping Module 1
Input Sensory Vectors Association Module 1
Object Segmentation Gateway Module
Learned Primitives
D
Instinctive Primitives
D
Navigation Module
Causal Memory Module
Autonomic Inputs 1...n
Autonomic Reflex Module 1
Autonomic Module
D
Goal/ Emotion Module
D
Autonomic Outputs 1...n
Sequential/ Error Correcting Module
Output Vector Shaping Module
Output Vector Association Module
Actuator Output

main_mech.cycles()

```
def cycles(d, g, h, m):
    # -->SENSORY INPUTS -> CCA3 -> MOTOR
    for d.evaluation_cycles in range(sys.
        autonomic_check(g)
        next_scene_from_envrt = (h.envrt_
        h.input_sensory_vectors_associati
        h.sequential_error_correcting_mod
        h.object_segmentation_module(g)
        h.navigation_module(d, g)
        h.output_vector_assocation_module
        if (next_scene_from_envrt < 0 or
            d, g, h = update_expected_val
    return d, g, h, m
```

OBJECT SEGMENTATION MODULE:

The object_segmentation_module calls the following methods:

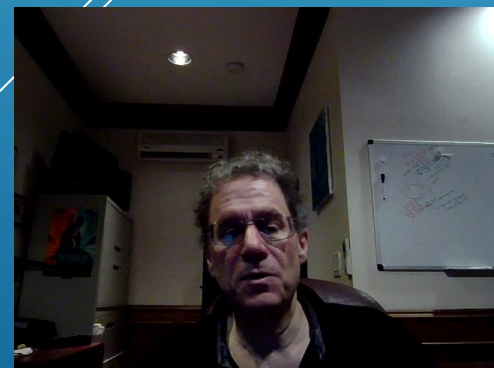
```
self.visual_zoom_out_into_navmap(g)
self.auditory_into_navmap(g
self.olfactory_into_navmap(g)
self.visual_into_navmap(g)
best_navmap = self.match_to_best_causal_memory_navmap(g)
self.current_navmap_zoom_in_largest_mismatch(best_navmap)
self.update_navmap(g, best_navmap)
```

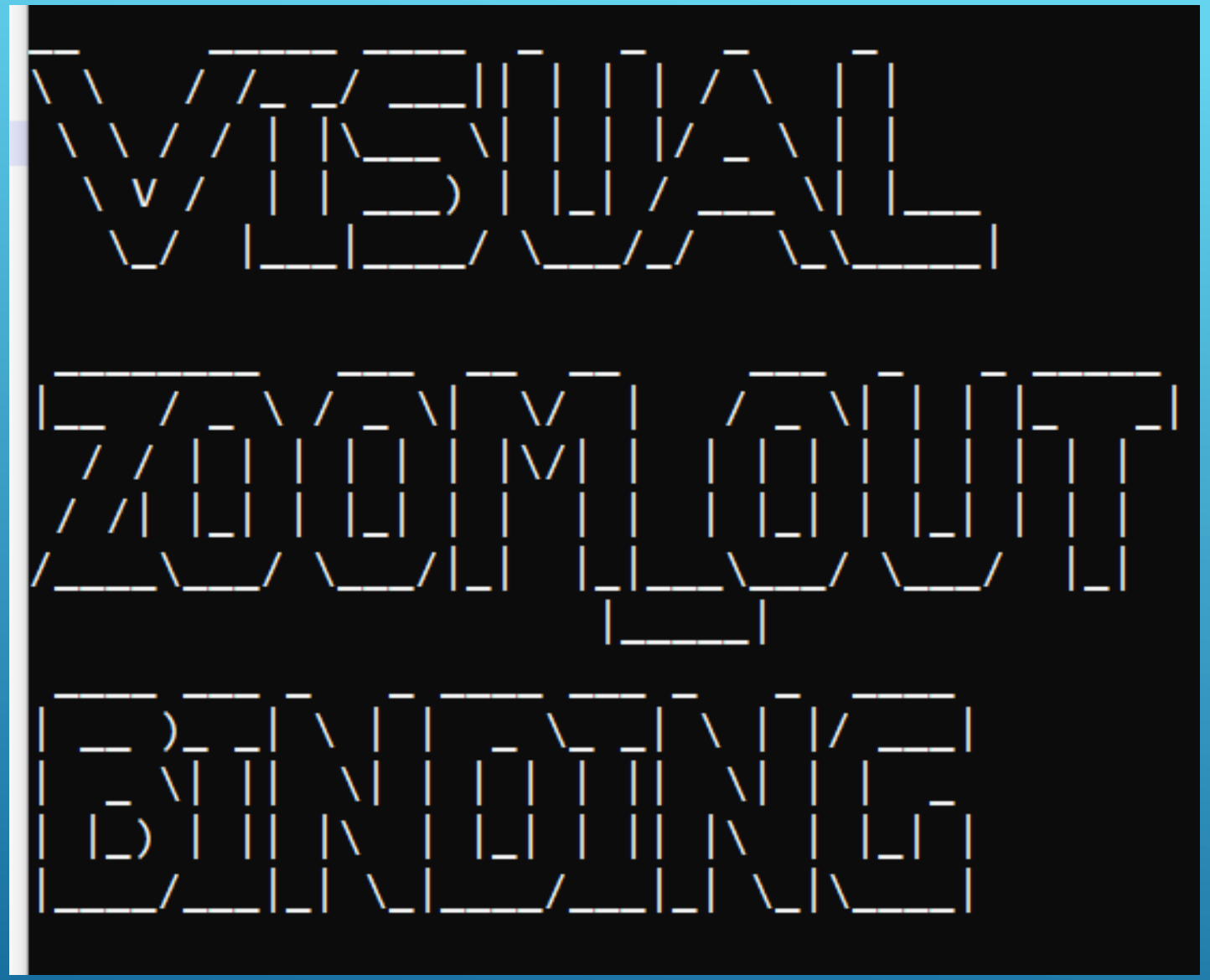
We will link the equations from the CCA3--A Solution to the Binding to the software operations as we proceed through these methods.

h.object_segmentation_module()

```
def object_segmentation_module(self, g) -> bool:
        # SIMULATION OF object_segmentation_module
        self.visual_zoom_out_into_navmap(g)
        self.auditory_into_navmap(g)
        self.olfactory_into_navmap(g)
        self.visual_into_navmap(g)
        best_navmap = self.match_to_best_causal_memory_navmap(g)
        self.current_navmap_zoom_in_largest_mismatch(best_navmap)
        self.update_navmap(g, best_navmap)
        return True
```

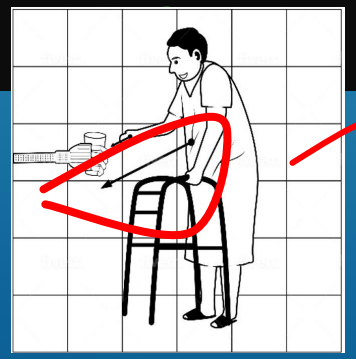
two visual sensory systems (close and far) used in the simulation




VISUAL
ZOOMOUT
BINDING

The visual_zoom_out features are combined with the
motion features from visual_zoom_out here.
LMN'(1,best_match,t) (56) is the best matching visual navmap
CONTEXT(t) (57) is the contextual parameter which is not used actively at
present in the simulation. VNM''(t) contains the motion prediction vectors for
visual and auditory changes.
VSNM(t) = Object_Segmentation_Gateway_Module.visualsegment(LNM'(1, ϒ ,t), VNM''t,
(58) which then has visual motion for segmented objects added.
In the code: self.visual_zoom_out_map, shown below

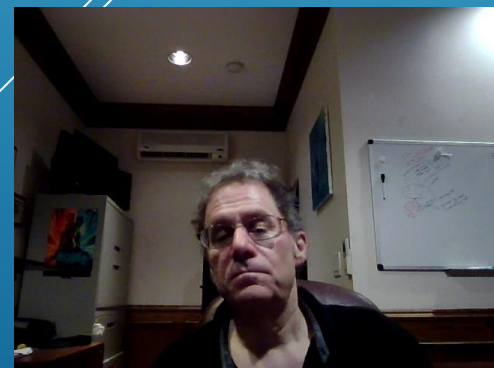
```
Visualization of Map Number n=112 with segment numbers (self.gb[n,x,y,z=0,s] or self.visual_zoom_out_map):
nb. (x,y,z=0,s) visualization   (0,0) is at the ground/floor level  ~ is motion degrees
Map type: visual   Note: eg, 'blob-7' == 'blob' in segment 7

y= 5: |                |                |body-0;                |                |
y= 4: |                |                |body-0;                |                |
y= 3: |                |                |body-0;                |                |
y= 2: |                |                |body-0;metal-1;270~-2; |body-0;         |
y= 1: |                |                |body-0;metal-1;        |body-0;         |
y= 0: |visual-10;      |                |body-0;metal-1;        |body-0;         |
x=              0                 2                 3                 4
```

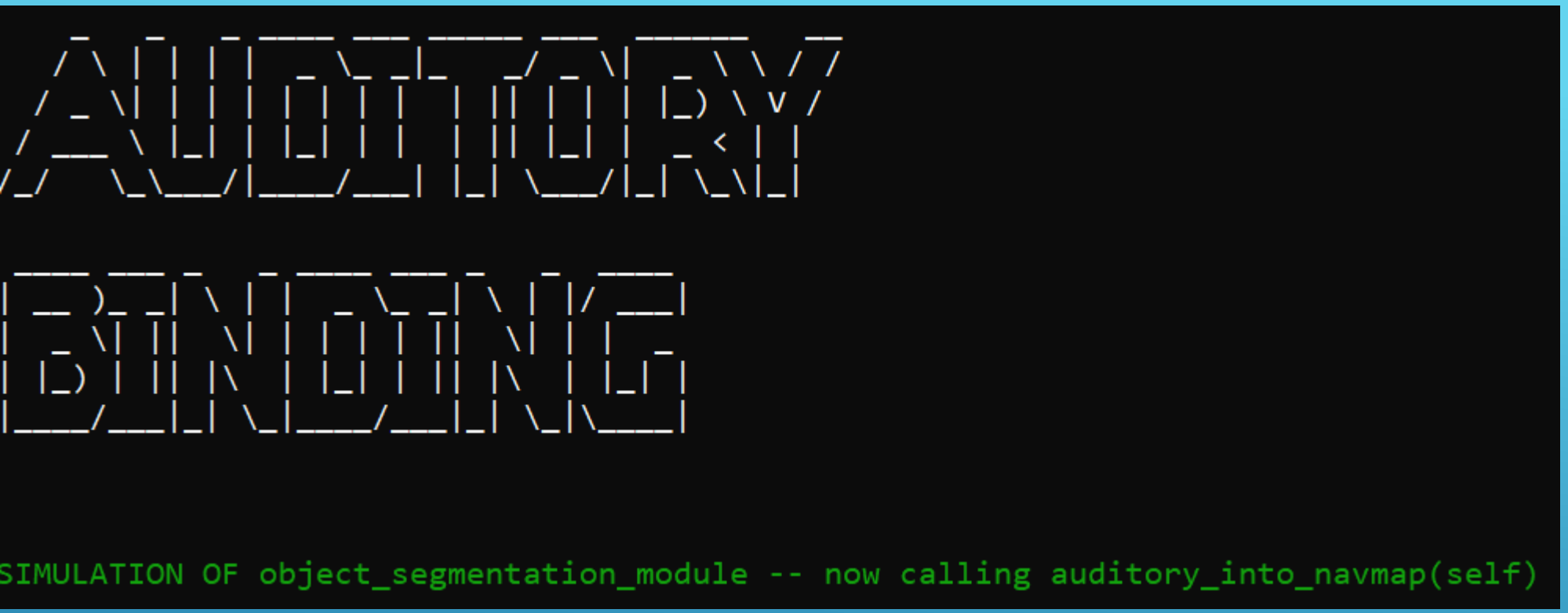


Sensory scene in front of the CCA3

h.object_segmentation_module()

```
def object_segmentation_module(self, g) -> bool:
        # SIMULATION OF object_segmentation_module
        self.visual_zoom_out_into_navmap(g)
        self.auditory_into_navmap(g)
        self.olfactory_into_navmap(g)
        self.visual_into_navmap(g)
        best_navmap = self.match_to_best_causal_memory_navmap(g)
        self.current_navmap_zoom_in_largest_mismatch(best_navmap)
        self.update_navmap(g, best_navmap)
        return True
```




SIMULATION OF object_segmentation_module -- now calling auditory_into_navmap(self)

Auditory inputs are put into a navmap for segmentation.
Motion possible but not used in simulation at present.
AVNM(t) (51) which represents the processed
auditory data exists at end of this method.
In code as: self.auditory_map

```
Visualization of Map Number n=113 with segment numbers (self.gb[n,x,y,z=0,s] or self.auditory_map):
nb. (x,y,z=0,s) visualization   (0,0) is at the ground/floor level  ~ is motion degrees
Map type: auditory   Note: eg, 'blob-7' == 'blob' in segment 7

y= 5: |                |              |              |              |              |
y= 4: |                |              |              |              |              |
y= 3: |                |              |              |              |              |get>glass_water-0;
y= 2: |                |              |              |              |              |
y= 1: |                |              |              |              |              |
y= 0: |auditory-10;    |              |              |              |              |
x=              0              1              2              3              4              5
```

5,3 "get>glass_water-0;"

h.object_segmentation_module()

```
def object_segmentation_module(self, g) -> bool:
        # SIMULATION OF object_segmentation_module
        self.visual_zoom_out_into_navmap(g)
        self.auditory_into_navmap(g)
        self.olfactory_into_navmap(g)
        self.visual_into_navmap(g)
        best_navmap = self.match_to_best_causal_memory_navmap(g)
        self.current_navmap_zoom_in_largest_mismatch(best_navmap)
        self.update_navmap(g, best_navmap)
        return True
```

OLEXTOOLS

BINDING

Olfactory inputs are put into a navmap for segmentation.
However, at this time, simulation just requires in LNM suitable for Object Segmentation Module.
Effectively this represents LNM'(3,best_match, t)

```
Visualization of Map Number n=114 with segment numbers (self.gb[n,x,y,z=0,s] or self.olfactory_map)
nb. (x,y,z=0,s) visualization   (0,0) is at the ground/floor level  ~ is motion degrees
Map type: olfactory   Note: eg, 'blob-7' == 'blob' in segment 7

y= 5: |                |              |              |              |                |
y= 4: |                |              |              |              |                |
y= 3: |                |              |              |              |                |
y= 2: |                |              |              |              |                |
y= 1: |                |              |              |              |hospital_room_odor-0;
y= 0: |olfactory-10;   |              |              |              |                |
x=            0                1              2              3              4              5
```

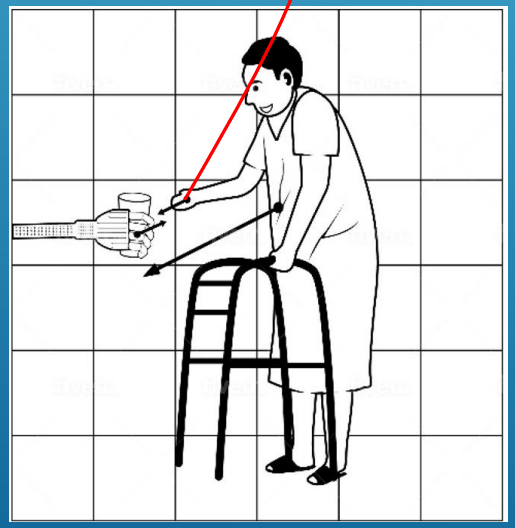
h.object_segmentation_module()

```
def object_segmentation_module(self, g) -> bool:
        # SIMULATION OF object_segmentation_module
        self.visual_zoom_out_into_navmap(g)
        self.auditory_into_navmap(g)
        self.olfactory_into_navmap(g)
        self.visual_into_navmap(g)
        best_navmap = self.match_to_best_causal_memory_navmap(g)
        self.current_navmap_zoom_in_largest_mismatch(best_navmap)
        self.update_navmap(g, best_navmap)
        return True
```

VISUAL BINDING

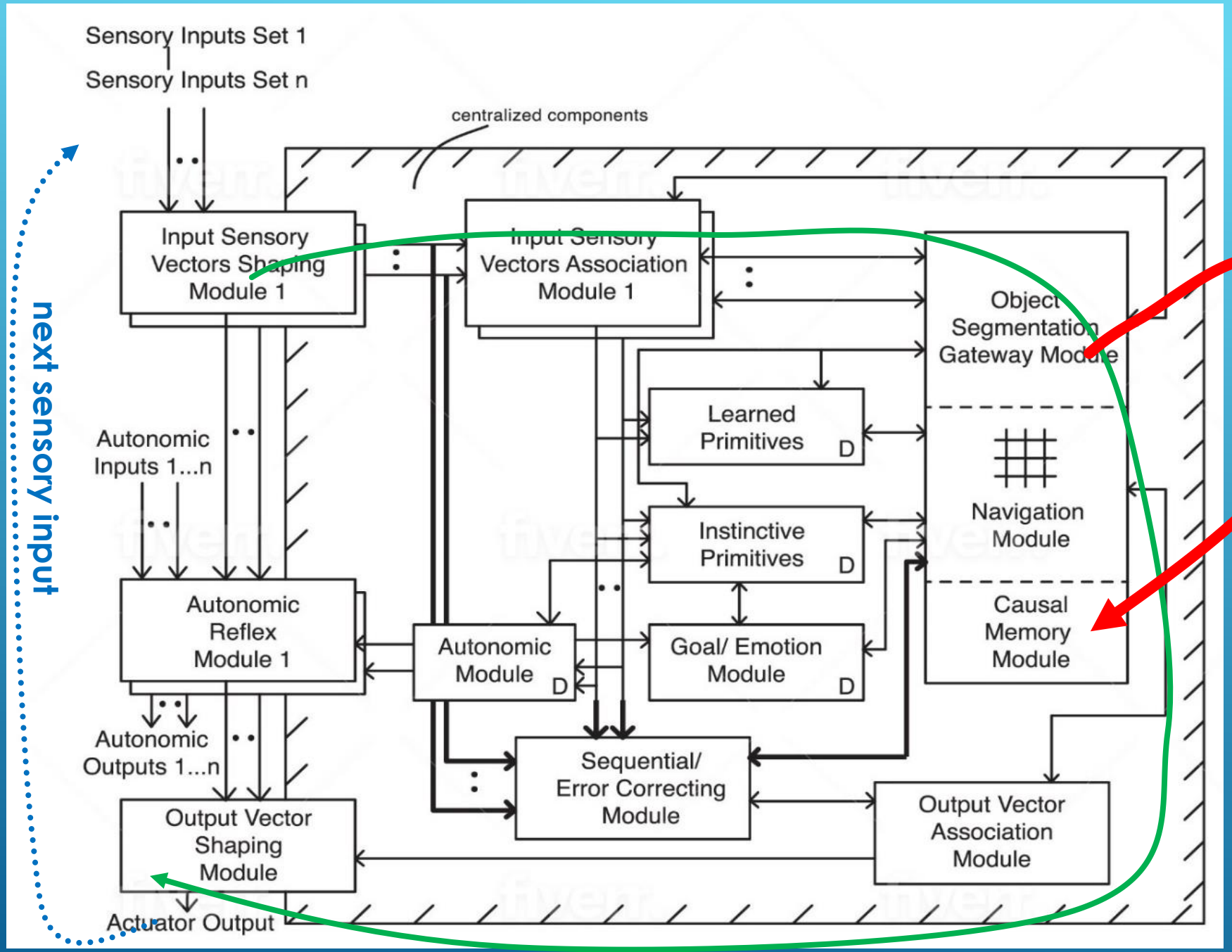
```
The visual features are combined with the
motion features from visual here.
LMN'(1,best_match,t)  was visual_zoom_out, thus consider as LNM'(4, best_match, t)
```

```
Visualization of Map Number n=115 with segment numbers (self.gb[n,x,y,z=0,s] or self.visual_map):
nb. (x,y,z=0,s) visualization   (0,0) is at the ground/floor level  ~ is motion degrees
Map type: visual   Note: eg, 'blob-7' == 'blob' in segment 7

y= 5: |                |                |                |                |                |
y= 4: |                |                |                |                |                |
y= 3: |                |                |                |270~-1;         |                |
y= 2: |                |                |                |left_hand>walker-0;|             |
y= 1: |                |                |                |                |                |
y= 0: |visual-10;      |                |                |                |                |
x=              0                1                2                3                4                5
```



h.object_segmentation_module()

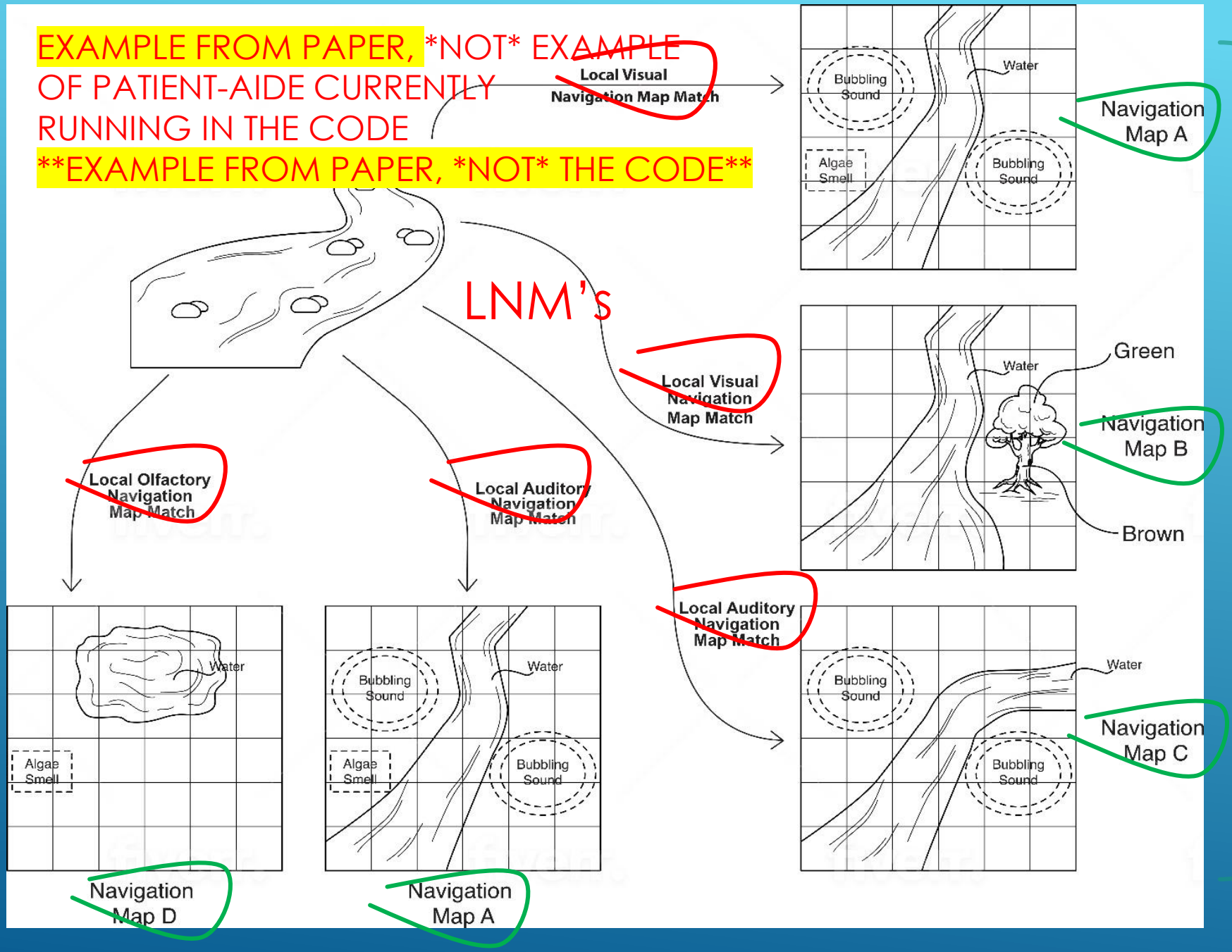
```
def object_segmentation_module(self, g) -> bool:
        # SIMULATION OF object_segmentation_module
        self.visual_zoom_out_into_navmap(g)
        self.auditory_into_navmap(g)
        self.olfactory_into_navmap(g)
        self.visual_into_navmap(g)
        best_navmap = self.match_to_best_causal_memory_navmap(g)
        self.current_navmap_zoom_in_largest_mismatch(best_navmap)
        self.update_navmap(g, best_navmap)
        return True
```



Match segmented LNM's against navmaps stored in Causal Memory Module – find best matching navmap

EXAMPLE FROM PAPER, *NOT* EXAMPLE OF PATIENT-AIDE CURRENTLY RUNNING IN THE CODE

EXAMPLE FROM PAPER, *NOT* THE CODE



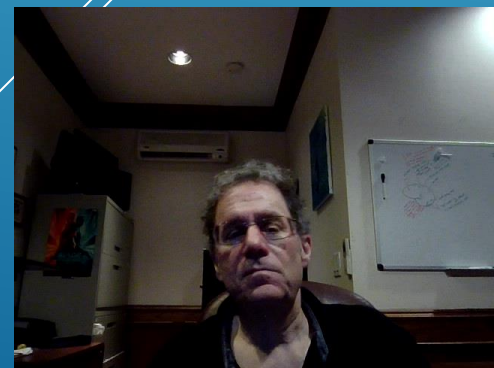
Navigation Maps A, B, C, D are navigation maps stored in the Causal Memory Module

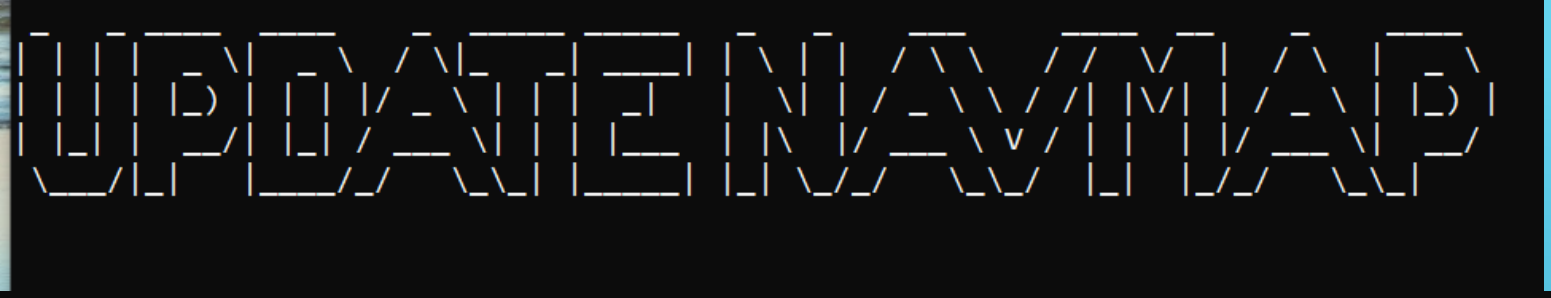

```
Segmented Local Navigation Maps are now matched against
existing stored Navigation Maps in the Causal Memory Module.
Equation 61 -- WNMt = Causal_Memory_Module.match_best_
multisensory_navigation_map((VSNM't , AVNMt , LNM'(3, ϒ ,t), LNM'(4, ϒ ,t
WNP(t) Working Navigation Map is thus considered the best match.
In the next step it will have to be updated with actual sensory inputs.
```

$$\mathbf{WNM}_t = \text{Causal\_Memory\_Module.match\_best\_multisensory\_navigation\_map}(\mathbf{VSNM'}_t, \mathbf{AVNM}_t, \mathbf{LNM'}_{(3,\Upsilon,t)}, \mathbf{LNM'}_{(4,\Upsilon,t)}, \ldots, \mathbf{LNM'}_{(n\_\sigma,\Upsilon,t)}) \quad (61)$$

h.object_segmentation_module()

```
def object_segmentation_module(self, g) -> bool:
        # SIMULATION OF object_segmentation_module
        self.visual_zoom_out_into_navmap(g)
        self.auditory_into_navmap(g)
        self.olfactory_into_navmap(g)
        self.visual_into_navmap(g)
        best_navmap = self.match_to_best_causal_memory_navmap(g)
        self.current_navmap_zoom_in_largest_mismatch(best_navmap)
        self.update_navmap(g, best_navmap)
        return True
```






```
This method must then compare the best_navmap WNM(t) with the actual
input sensory information as represented on the local nav maps (i.e., 'loca
a sensory system such as visual, auditory, etc). If match is very close the
can update the best matching WNM(t), but if not then make a copy of the bes
and modify as appropriate, and then can save new map for future use as well
Equation 62 -66 occur here.
| differences(actualt , WNMt) |  ≤h' , ⇒ WNM't = WNMt ⇒ actualt (65)
| differences(actualt , WNMt) | >h' , ⇒ WNM't = NewNMt ⇒ actualt (66)
```

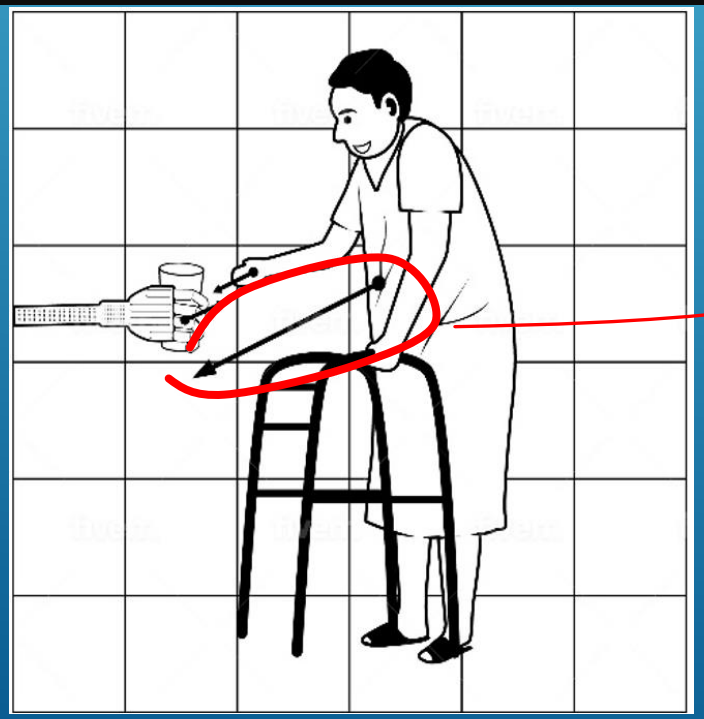
$$actual_t = [VSNM'_t , AVNM_t , LNM'_{(3,\Upsilon,t)}, LNM'_{(4,\Upsilon,t)}, \ldots, LNM'_{(n\_\sigma,\Upsilon,t}] \quad (63)$$

$$| differences(actual_t , WNM_t) | \leq h' , \Rightarrow WNM'_t = WNM_t \cup actual_t \quad (65)$$

$$| differences(actual_t , WNM_t) | > h' , \Rightarrow WNM'_t = NewNM_t \cup actual_t \quad (66)$$

Working Navigation Map WNM'(t)

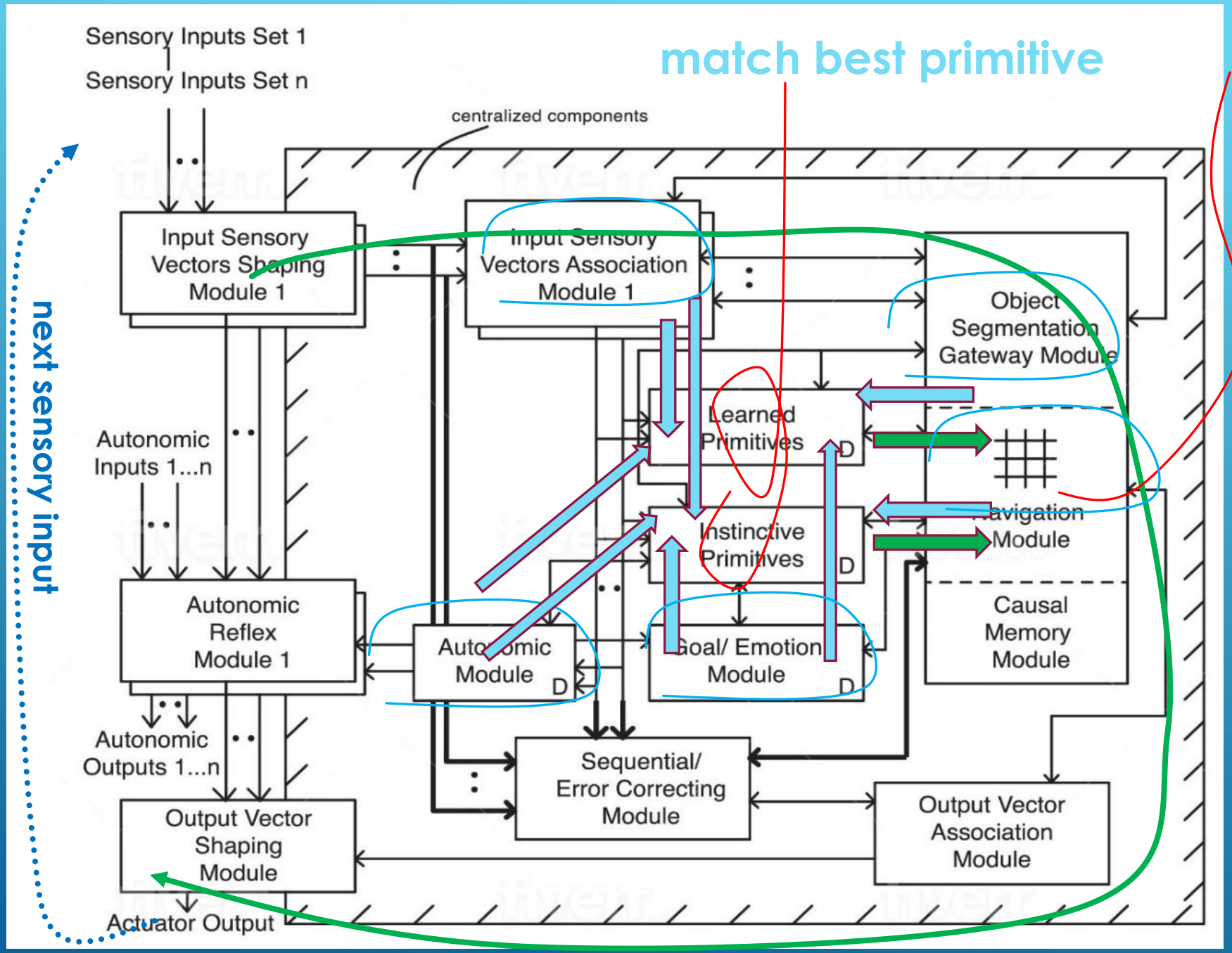
```
Visualization of Map Number n=116 with segment numbers (self.gb[n,x,y,z=0,s] or self.working_navmap):
nb. (x,y,z=0,s) visualization   (0,0) is at the ground/floor level  ~ is motion degrees
Map type: None   Note: eg, 'blob-7' == 'blob' in segment 7

y= 5: |                 |                 |body-0;          |                 |
y= 4: |                 |                 |body-0;          |                 |
y= 3: |                 |body-0;          |body-0;          |                 |get>glass_water-7;
y= 2: |                 |metal-1;         |body-0;metal-1;270~-2;|body-0;          |
y= 1: |                 |metal-1;         |body-0;metal-1;  |body-0;          |hospital_room_odor-8;
y= 0: |navmap-9;        |metal-1;         |body-0;metal-1;  |body-0;          |
x=          0                 1                 2                 3                 4                 5
```



main_mech.cycles()

```
def cycles(d, g, h, m):
    # -->SENSORY INPUTS -> CCA3 -> MOTOR
    for d.evaluation_cycles in range(sys.
        autonomic_check(g)
        next_scene_from_envrt = (h.envrt_
        h.input_sensory_vectors_associati
        h.sequential_error_correcting_mod
        h.object_segmentation_module(g)
        h.navigation_module(d, g)
        h.output_vector_assocation_module
        if (next_scene_from_envrt < 0 or
            d, g, h = update_expected_val
    return d, g, h, m
```

x



Navigation Module will apply operations on WNM(t) (the Working Navigation Map)

match primitives

navmap

$$\boldsymbol{emotion} \in R \quad (67)$$

$$\mathbf{GOAL} \in R^{mxnxo} \quad (68)$$

$$\boldsymbol{autonomic} \in R \quad (69)$$

$$[\boldsymbol{emotion}_t, \mathbf{GOAL}_t] = \text{Goal/Emotion\_Module.set\_emotion\_goal}(\boldsymbol{autonomic}_t, \mathbf{WNM'}_t) \quad (70)$$

$$\mathbf{WIP} \in R^{mxnxo} \quad (71)$$

$$\mathbf{WIP}_t = \text{Instinctive\_Primitives\_Module.match\_best\_primitive}(\boldsymbol{actual}_t, \boldsymbol{emotion}_t, \mathbf{GOAL}_t) \quad (72)$$

$$\mathbf{WLP} \in R^{mxnxo} \quad (73)$$

$$\mathbf{WLP}_t = \text{Learned\_Primitives\_Module.match\_best\_primitive}(\boldsymbol{actual}_t, \boldsymbol{emotion}_t, \mathbf{GOAL}_t) \quad (74)$$

$$\mathbf{WPR} \in R^{mxnxo} \quad (75)$$

$$\mathbf{WLP}_t = [\,], \Rightarrow \mathbf{WPR}_t = \mathbf{WIP}_t \quad (76)$$

$$\mathbf{WLP}_t \neq [\,], \Rightarrow \mathbf{WPR}_t = \mathbf{WLP}_t \quad (77)$$

$$\boldsymbol{action}_t = \text{Navigation\_Module.apply\_primitive}(\mathbf{WPR}_t, \mathbf{WNM'}_t) \quad (78)$$

$\mathbf{WLP_t} = [\,], \Rightarrow \mathbf{WPR_t} = \mathbf{WIP_t}$ (76)

$\mathbf{WLP_t} \neq [\,], \Rightarrow \mathbf{WPR_t} = \mathbf{WLP_t}$ (77)

There will always be some instinctive primitive (WIP) which the navigation module can apply against the updated Working Navigation Map (WNM')
If there is a match for a learned primitive (WLP) then that learned primitive will be used instead of the instinctive primitive

$\boldsymbol{action_t}$ = Navigation_Module.apply_primitive($\mathbf{WPR_t}$, $\mathbf{WNM'_t}$) (78)

The Working Primitive (WPR(t)) is applied against the Working Navigation Map → action(t)

```
 _   _    ___    _____ ___    _    _____ ___ ___  _   _
| \ | |  / \ \  / /_ _/ __|  / \|_   _|_ _/ _ \| \ | |
|  \| | / _ \ \/ / | | |_  / _ \ | |  | | | | |  \| |
| |\  |/ ___ \  /  | | |_| / ___ \| |  | | |_| | |\  |
|_| \_/_/   \_\/  |___\____/_/   \_\_| |___\___/|_| \_|

 __  __  ___  ____  _   _ _     _____
|  \/  |/ _ \|  _ \| | | | |   | ____|
| |\/| | | | | | | | | | | |   |  _|
| |  | | |_| | |_| | |_| | |___| |___
|_|  |_|\___/|____/ \___/|_____|_____|

 ___ _   _ ____ _____ ___ _   _  ____ _____ ___ _   _ _____
|_ _| \ | / ___|_   _|_ _| \ | |/ ___|_   _|_ _| | | | ____|
 | ||  \| \___ \ | |  | ||  \| | |     | |  | || | | |  _|
 | || |\  |___) || |  | || |\  | |___  | |  | || |_| | |___
|___|_| \_|____/ |_| |___|_| \_|\____| |_| |___\___/|_____|
```

The instinctive primitives and the learned primitives are matched against the sensory input local navigation maps as well as an emotion and goal
A 'Working Primitive' WPR is assigned to the best matching primitive.
Thus equations 67 -77 and there is a WPR(t) i.e., the primitive which will be
actiont = Navigation_Module.apply_primitive(WPRt, WNM't) (78)
actin(t) is then sent to the Output_Vector_Assoc_Module as well as to
to the Seq/Error Correction Module.

WPR "Working Primitive" applied against WNM "Working Nav Map"

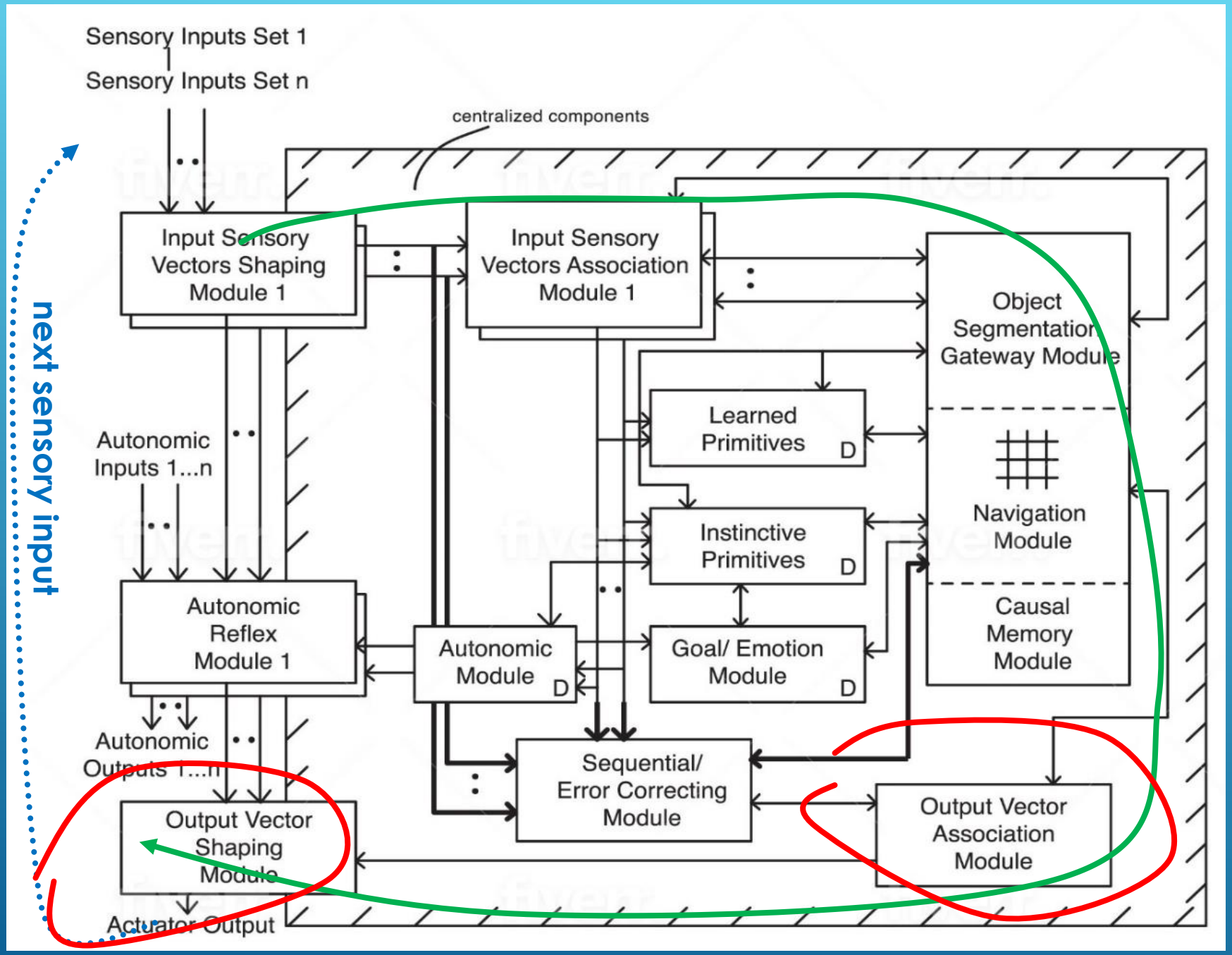
```
Visualization of Map Number n=106 with segment numbers (self.gb[n,x,y,z=0,s] or best match of instinctive_primitives to presenting
ap):
nb. (x,y,z=0,s) visualization   (0,0) is at the ground/floor level  ~ is motion degrees
Map type: instinctive   Note: eg, 'blob-7' == 'blob' in segment 7

y= 5: |                 |                 |                 |                 |
y= 4: |                 |                 |body-0;          |                 |
                                            |hospital_odor-0; |robot_left_arm-9;patient with walker triggers ge
e of help-10;|
y= 3: |                 |                 |                 |                 |
y= 2: |                 |body-0;          |body-0;          |                 |
y= 1: |                 |metal-1;         |body-0;metal-1;  |body-0;          |
y= 0: |instinctive-10;  |metal-1;         |body-0;metal-1;  |body-0;          |
x=            0                 1                 2                 3                 4                 5
```

action(t):
Robot arm to move to (4,4)

main_mech.cycles()

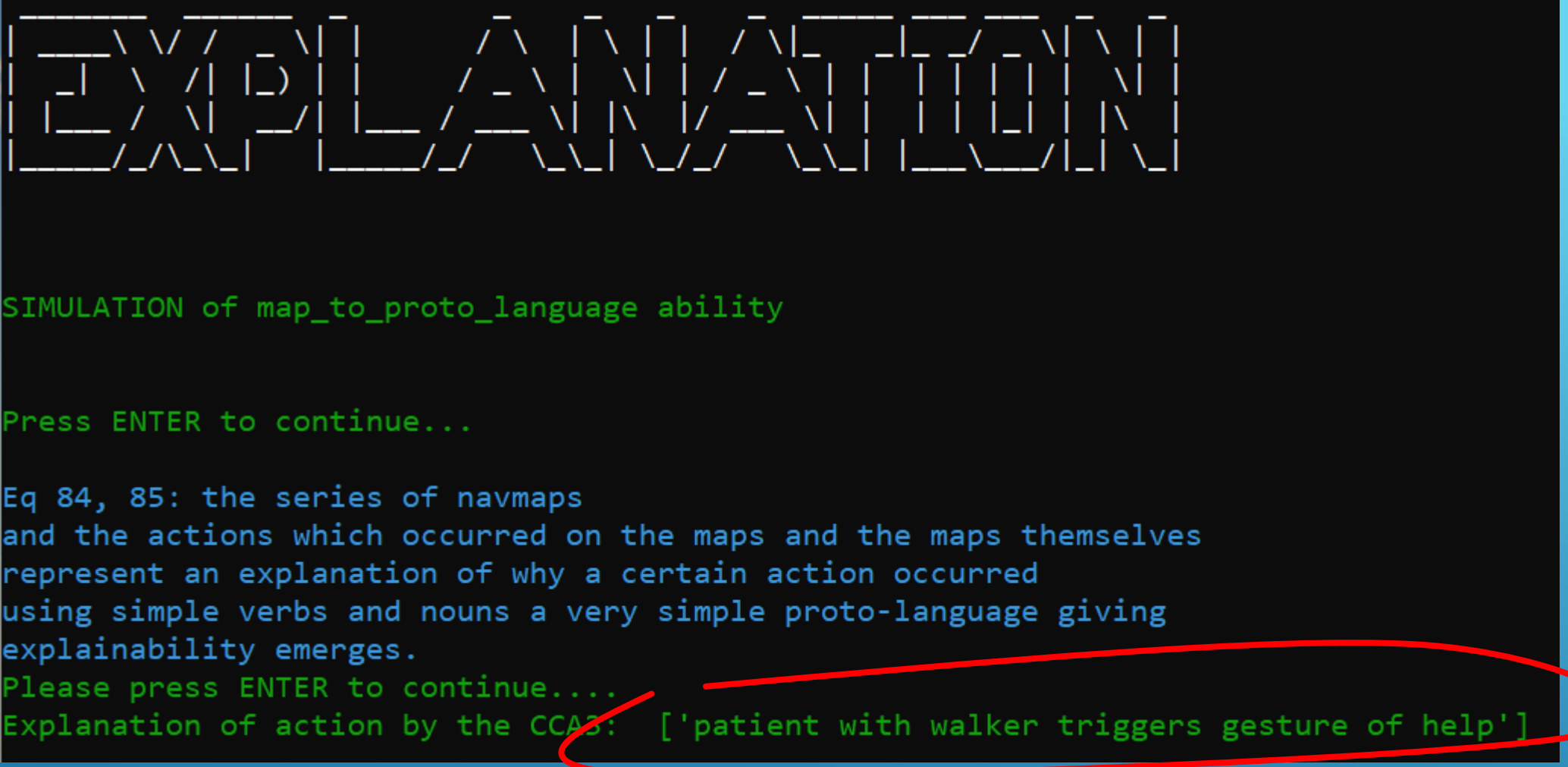
```
def cycles(d, g, h, m):
    # -->SENSORY INPUTS -> CCA3 -> MOTOR
    for d.evaluation_cycles in range(sys.
        autonomic_check(g)
        next_scene_from_envrt = (h.envrt_
        h.input_sensory_vectors_associati
        h.sequential_error_correcting_mod
        h.object_segmentation_module(g)
        h.navigation_module(d, g)
        h.output_vector_association_module
        if (next_scene_from_envrt < 0 or
            d, g, h = update_expected_val
        return d, g, h, m
```

x


Sensory Inputs Set 1
Sensory Inputs Set n
centralized components
Input Sensory Vectors Shaping Module 1
Input Sensory Vectors Association Module 1
Object Segmentation Gateway Module
Learned Primitives
D
Navigation Module
Instinctive Primitives
D
Causal Memory Module
Autonomic Inputs 1...n
Autonomic Reflex Module 1
Autonomic Module
D
Goal/ Emotion Module
D
Autonomic Outputs 1...n
Sequential/ Error Correcting Module
Output Vector Association Module
Output Vector Shaping Module
Actuator Output
next sensory input

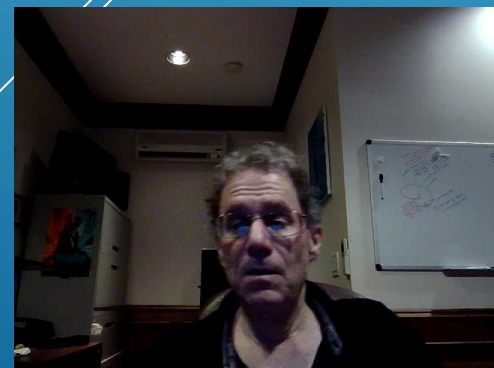
OUTPUT VECTOR ASSOCIATION MODULE

```
action(t) (78) has been sent from the navigation module to the
output_vector_assocn_module where it will create the output_vector(t) (80
output_vector(t) is then corrected by a motion correction signal from the
seq/error correcting module although not used as this time by the simulat
Please press ENTER to continue....

CCA3 Robot takes this action: move  robot_left_arm to 4,4,0
```



$$\textit{explanation}_t = \text{Navigation\_Module.navmap\_to\_proto\_lang}(\ \mathbf{WPR}_t, \mathbf{WNM'}_t, \textit{action}_t\ ) \quad (85)$$

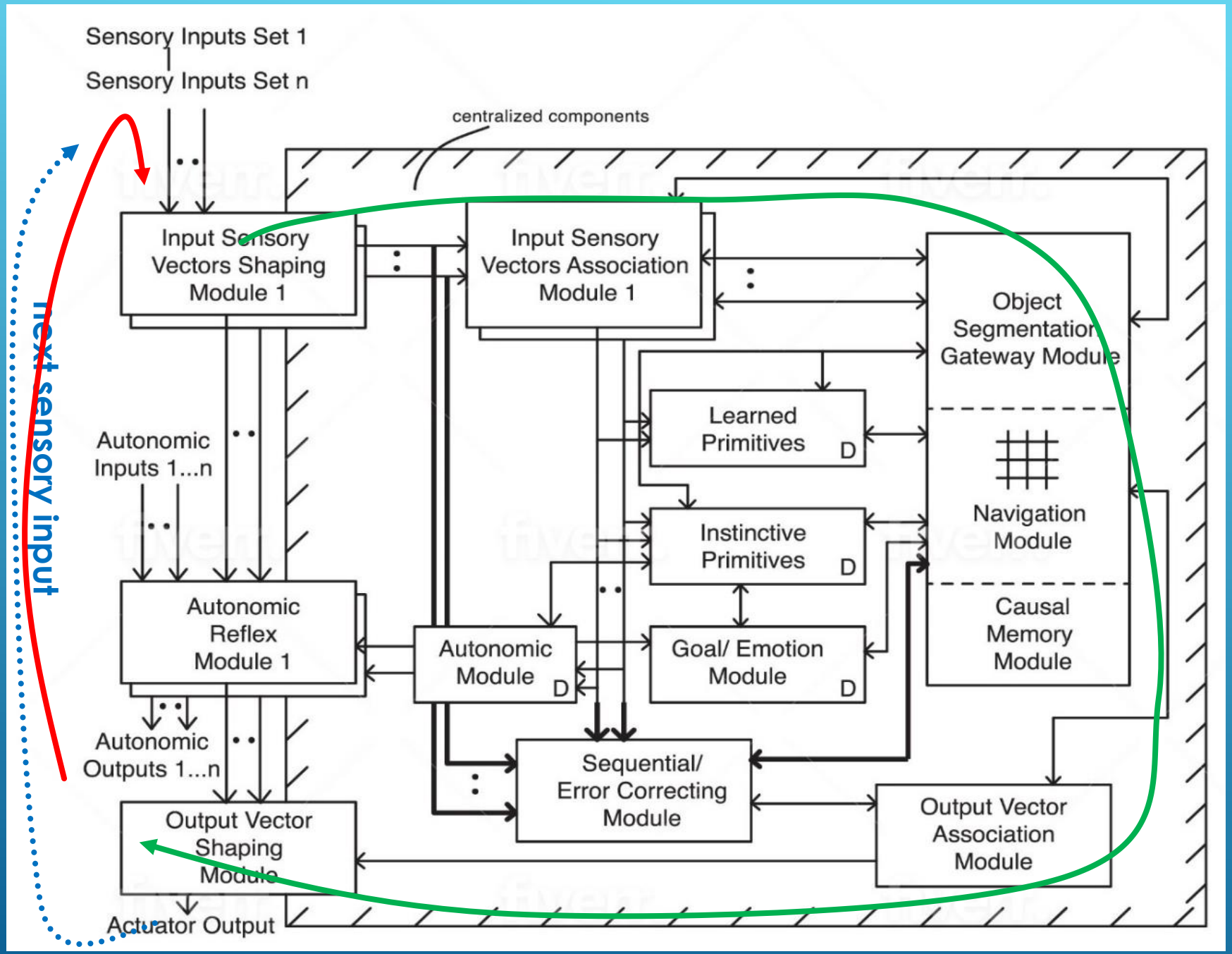
dot notation



main_mech.cycles()

```
def cycles(d, g, h, m):
    # -->SENSORY INPUTS -> CCA3 -> MOTOR
    for d.evaluation_cycles in range(sys.
        autonomic_check(g)
        next_scene_from_envrt = (h.envrt_
        h.input_sensory_vectors_associati
        h.sequential_error_correcting_mod
        h.object_segmentation_module(g)
        h.navigation_module(d, g)
        h.output_vector_assocation_module
        if (next_scene_from_envrt < 0 or
            d, g, h = update_expected_val
        return d, g, h, m
```

x


Sensory Inputs Set 1
Sensory Inputs Set n
centralized components
Input Sensory Vectors Shaping Module 1
Input Sensory Vectors Association Module 1
Object Segmentation Gateway Module
Learned Primitives
D
Navigation Module
Instinctive Primitives
D
Causal Memory Module
Autonomic Inputs 1...n
Autonomic Reflex Module 1
Autonomic Module
D
Goal/ Emotion Module
D
Autonomic Outputs 1...n
Sequential/ Error Correcting Module
Output Vector Shaping Module
Output Vector Association Module
Actuator Output
next sensory input

main_mech.cycles()

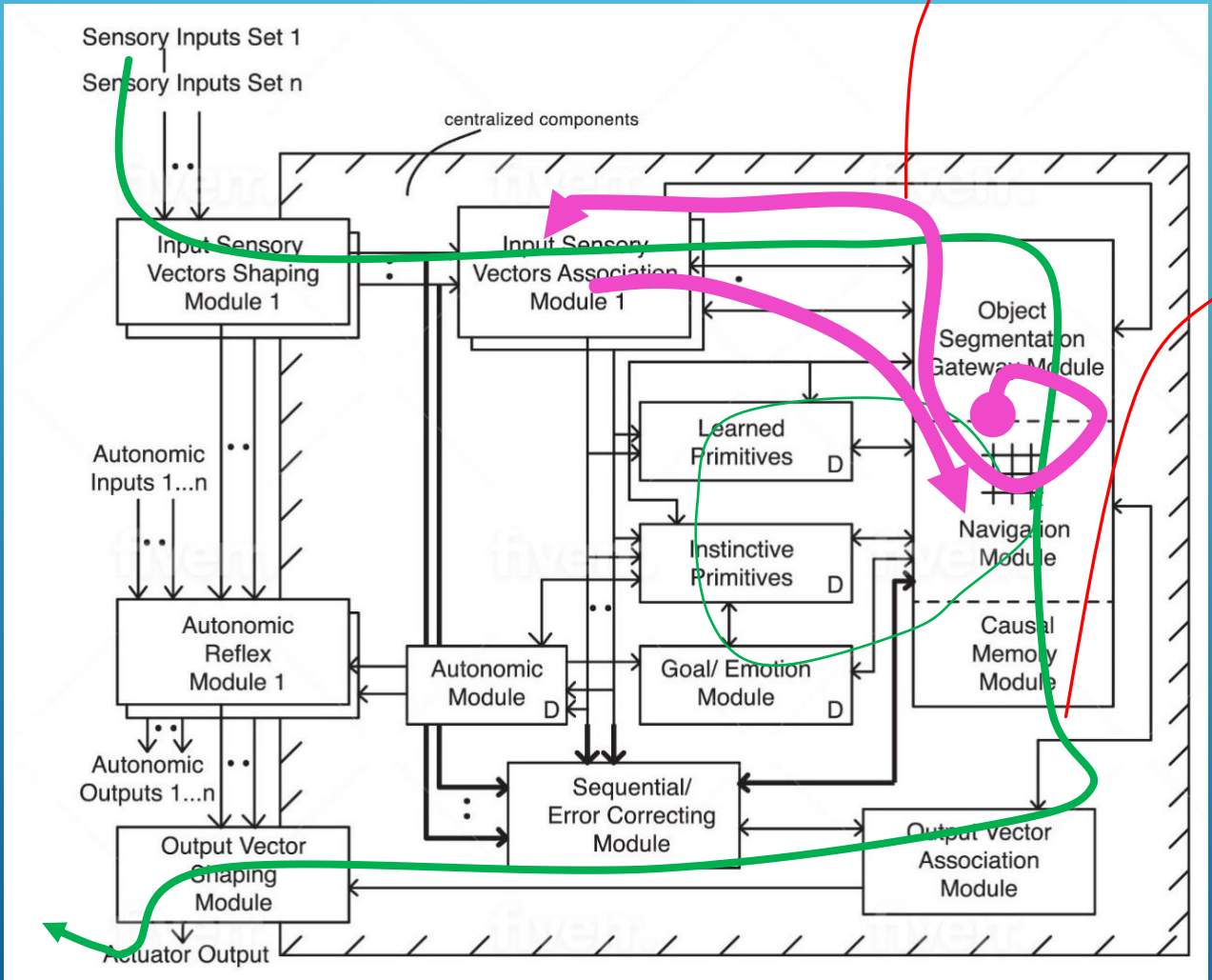
```
def cycles(d, g, h, m):
    # -->SENSORY INPUTS -> CCA3
    for d.evaluation_cycles in
        autonomic_check(g)
        next_scene_from_envrt = (h.envrt_
        h.input_sensory_vectors_associati
        h.sequential_error_correcting_mod
        h.object_segmentation_module(g)
        h.navigation_module(d, g)
        h.output_vector_assocation_module
        if (next_scene_from_envrt < 0 or
            d, g, h = update_expected_val
        return d, g, h, m
```

```
    input(to_print)
KeyboardInterrupt
^C
C:\Users\howar>
```

Feedback Signals and Intermediate Results

What about feedback operations??

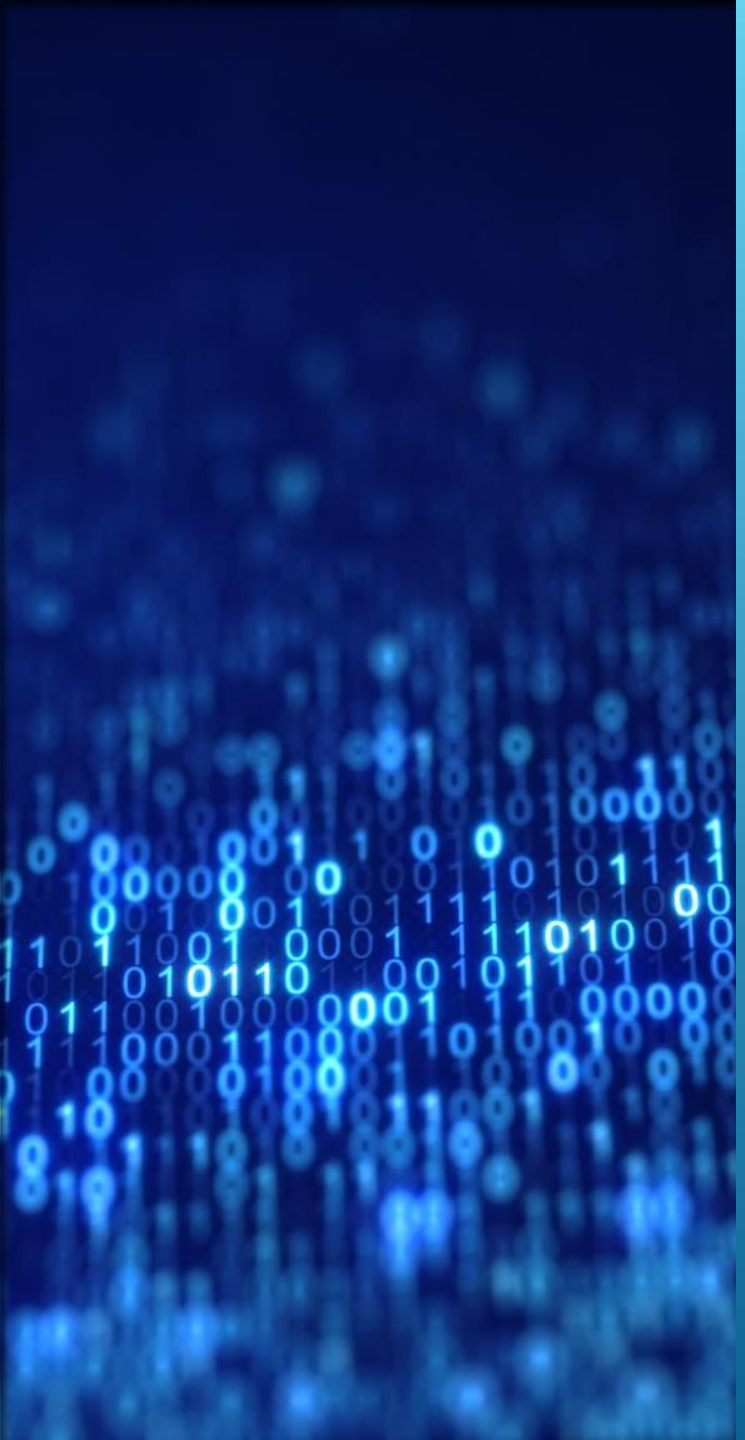
Action produced after operation of primitive on navigation map



continued in VIDEO 6....

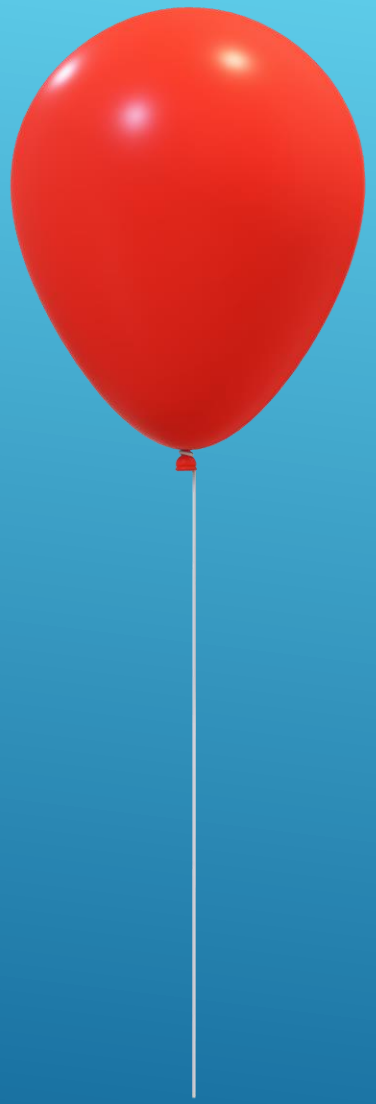
$$\mathbf{LNM}_{(\sigma,\Upsilon,t)} = \text{Input\_Sensory\_Vectors\_Associations\_Module}_{\sigma}.\text{match\_best\_local\_navigation\_map}(\ \mathbf{S'}_{\sigma,t},\ \mathbf{WNM'}_{t-1}\ ) \quad (86)$$

$$\mathbf{WPR}_{t-1} = [\text{“feedback intermediate*”}],\ \Rightarrow \forall_{\sigma}:\ \mathbf{LNM}_{(\sigma,\Upsilon,t)} = \text{Input\_Sensory\_Vectors\_Associations\_Module}_{\sigma}.\text{extract\_}\sigma(\ \mathbf{WNM'}_{t-1}\ ) \quad (87)$$



- CCA3 Overview ✔
- Binding Problem Overview ✔
- Software Overview ✔
- Operations Overview ✔
- Operations Causal Vid#6
- Software in More Detail Vid#7
- More videos, code on GitHub "CausalCog"

(If interest, continued updating on GitHub)



….continued in VIDEO 6

(but completion of series of 5 videos uploaded as supplementary material)

balloon from powerpoint stock image