

practical tasks

[all line numbers refer to the attached printout of the code]

1. /dac.khepera/dac.c

line 192: endless loop executing `DAC()`;

in `DAC()`, the input to the proximity layer (line 145), collision layer (line 153), and the motor layer gets calculated. After this, a learning step is executed on line 183. → `Learn()`;

Hebbian Learning:

In `Learn()`, we're calculating the weights between the proximity and collision layer.

If there is no collision, we're actually not learning or forgetting anything. If there are one or several collisions, the learning step affects the proximity-collision weights. Active forgetting is implemented in the same formula.

3. Node activation (collision layer)

$$a_i = g\left(\sum w_{ij} o_j\right) = g(h_i)$$

node activation for the collision layer, line 165/166:

```
// Input from proximity layer (fully connected)
for (i=0; i<NUM_PROXIMITY; i++)          // [0,1]*[0,1]
    h = h + FAC_COLLISION*ProxCollWeights[i][k]*Proximity[i];
```

4. Learning Rule (Hebbian Learning)

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij} \quad \Delta w_{ij} = \frac{1}{N} (\mathbf{h} \cdot o_j \cdot a_i - \mathbf{e} \cdot \bar{a} \cdot w_{ij})$$

The learning rule is implemented on line 130-132 in the `Learn()`-function:

```
ProxCollWeights[i][k] = ProxCollWeights[i][k] +
    ((ETA*Proximity[i]*Collision[k] -
    EPS*a*OldProxCollWeights[i][k])/NUM_PROXIMITY);
```

N: NUM_PROXIMITY

η : ETA (learning rate)

ϵ : EPS (forgetting rate)

a_i : Collision[k]

o_j : Proximity[k]

...

5. CollisionNeuronOut()

This returns [0,1]. This function gets called in `DAC()` and returns 1 if the input from the proximity layer reaches a certain threshold (e.g. 0.3).

```
float CollisionNeuronOut(float h) {
    if (h>0.3f) return 1.0f; // collision
    else      return 0.0f; // no collision
}
```

6. DAC testing

When Khepera starts running, the weight-arrays get initialized by zero values in `InitRobot()`. At the beginning Khepera keeps on bumping into the walls. Every time there is a collision, Khepera learns and the weights between the proximity and collision layer grow. Khepera gets smarter and smarter and pretty soon it doesn't bump anymore into the walls.

Still, the weights don't constantly grow as active forgetting is implemented in the Hebbian Learning rule. The amount of forgetting depends on the actual weights and this way, at a certain point, the weights stay pretty much the same.

If we raise the learning rate (η), Khepera learns much faster. Especially at the beginning of Kheperas mission we notice this. Khepera learns already a lot after its first bump into the wall. Well, such a bump hurts, so I strongly recommend Khepera to acquire a high learning rate!

7. ProxColl-weights

This prints out the 2-dimensional `ProxCollWeights`-array. The weights for sensor 6 and 7 stay zero as they are the two sensors on the back of Khepera and as Khepera only keeps on moving straight.

```
// Calculate new weights using Hebb
for (i=0; i<NUM_PROXIMITY; i++) {
    printf("%d: ", i);
    for (k=0; k<NUM_COLLISION; k++) {
        ProxCollWeights[i][k] = ProxCollWeights[i][k] +
            ((ETA*Proximity[i]*Collision[k] -
              EPS*a*OldProxCollWeights[i][k])/NUM_PROXIMITY);
        printf("%f ", ProxCollWeights[i][k]);
    }
    printf("\n");
}
```

8. Real Khepera

The real Khepera doesn't really seem to be quite smart as we need to increase the learning rate to get the same result :)

theoretical tasks

1. DAC - Distributed Adaptive Control

Behind Distributed Adaptive Control there is the idea of a "living" system, a system that is located in the real world. Biological systems are adaptive as they're able to find a suitable behavior for new situations.

Adaptivity is the ability to adjust oneself to the environment and is one of the basic concepts of a complete agent. To acquire adaptivity, the agent must be able to "learn". In a DAC, the learning rule, especially Hebbian Learning, is one of the central parts.

If we look at our robot implementation, we notice that Khepera is able to adapt to its environment. It only learns to avoid bumping into walls if there are actually walls around. If there is nothing around, it doesn't need to learn anything. Every creature has a low level of knowledge at the beginning and starts learning through an adaptive behavior.

AI - Classwork Assignment 6

Distribution: In a DAC we got several layers which are fully connected. Different layers provide different tasks or functions. In a primitive DAC we got a collision layer, a proximity layer, and a motor activation layer.

If a sensor or a neuron doesn't work properly, the whole network of the agent still works. This provides a high redundancy of the system.

2. Preventing infinite learning

Why should an agent ever stop learning or even forget what it once learned?

What's the point of forgetting? Isn't this an unnecessary burden?

Let's think about human beings. At first sight we might consider "forgetting" as something really annoying. Wouldn't it be great to just not forget anything?

Actually you'd better not wish this. "Forgetting" is not just a burden, no, it's actually important for our survival. Remember the R2D1 – the robot-relevant-deducer – a perfect example for the Frame Problem. That robot had to distinguish between relevant and irrelevant problems it faced in the real world. It ended up in an endless relevant/irrelevant-check. The way of programming a R2D1 is not the right way to go designing an agent that has to act in its changing environment.

The main talent of a human being and any other situated agent is its ability to distinguish between relevant and irrelevant things. To acquire this, it has to forget irrelevant things, otherwise this would never be possible.

There may be people who know a whole phonebook by heart but on the other side they may lack the advantage of "forgetting". They often wind up in psychiatric clinics.

In our DAC equipped robot, forgetting is implemented in the **Hebbian Learning** rule. It only forgets while it is learning. → **Active forgetting**

$$\Delta w_{ij} = \frac{1}{N} (\mathbf{h} \cdot o_j \cdot a_i - \mathbf{e} \cdot \bar{a} \cdot w_{ij})$$

The second part of the formula demonstrates the forgetting process. Every time the robot learns, it also forgets in relation to the amount of already learned things. If the learning rate is high, it forgets also more than if it is low. This completely makes sense and reproduces the behavior of any complete agent.

3. Redundancy of sensory channels

In our example of a DAC equipped Khepera robot, collision sensors really seem to be completely redundant. A collision actually is nothing else than a zero-value of the proximity sensor. Measuring proximity is achieved by IR-sensors.

To avoid the implementation of collision sensors, all we actually need to do is to set a certain threshold for the proximity value to affect a collision. Like this we got some kind of virtual collision sensors.

There may be a problem with such a virtual collision sensor if the IR-sensor doesn't deliver correct values while measuring small distances.

The main advantage of using different sensory channels is an intentional creation of **redundancy**. A robot with different sensory channels would be more **stable**, in case some sensors break.

In our agent we were using IR-sensors for both purposes, proximity and collision measurement. In some cases IR-sensors may not recognize a collision, e.g. if the agent collides with a black wall, whereas collision sensors would.