Eukalyptus

(Koala Environment Simulator)

The cortex – a first approach

Thabo Beeler, University of Applied Sciences, Rapperswil (HSR)

Revision History

Version	When	What	Who
0.1	05.06.2004	Initial version	Thabo Beeler

Table of contents

Introduction

This document is the collection of some ideas considering map building and path finding.

General notations

Space

At the moment, the Koala moves in a 2-Dimensional space. For the mental map, we will add the orientation of the Koala as the third dimension.

Location

The location of the Koala is always in this 3-Dimensional space. It is called $\vec{L}$ and looks like this: $\vec{L} = \begin{matrix} x \\ y \\ k \end{matrix}$, here k is a constant factor to fit the orientation to the Cartesian position of the Koala.

Position

Sometimes we don't care about the orientation. Then we talk about the position of an object. The position is therefore the 2-Dimensional vector $\vec{P} = \begin{matrix} x \\ y \end{matrix}$.

Map building

The map building approach consists of multiple parts. The complete solution should be able to

- cartograph a 2-Dimensional space
- perform error correction

Mental map

First, we will look at the cartography task only. So we assume the world to be perfect, which, of course, it is not.
The Koala should be able to build a mental map, which in turn should serve as the base for the path finding task as well as for the goal directed navigation.
As the Koala does not know how big the space to explore is, the mental map should be created dynamically.

Landmarks

Landmarks are perceived at a given point in time for some period of time. As the Koala has not a 360 ° view field, it will loose sight of a landmark, even if it is still in reach. E.g. when scanning for new landmarks.
So we need to remember the landmarks and to forget them while we move along. We will call this the landmark oblivion function.
We will return to this function in the context of the landmark neurons. For the moment, just assume that we receive a value in the range [0,1], where 1 means the Koala just now perceives the landmark, and 0 means the Koala did completely forget about the landmark. These values will be called landmark powers. They are represented as an N-Dimensional vector, where N is the amount of landmarks perceived so far: $\overrightarrow{LM} = [LM_1, LM_2, ..., LM_N]$

S-Neuron

The mental map consists only of so called state neurons. They separate the 3-Dimensional space into N slices, where N is the amount of neurons in the net.
A state neuron can schematically be drawn like this:

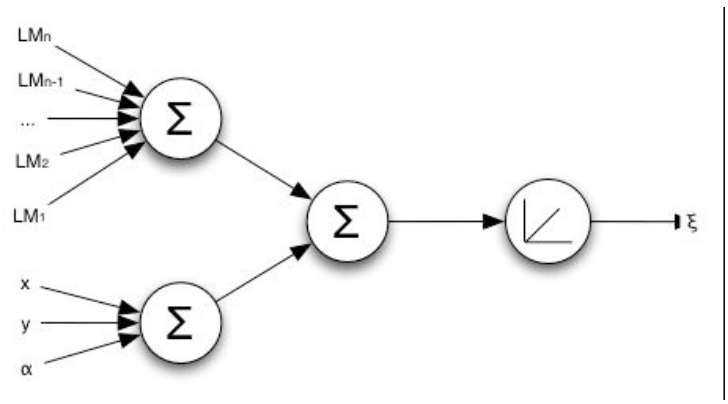

Figure 1: S-Neuron schematic

When a state neuron is created it sets the weights of the links to the current state. This means, the weights of the links from the landmark neurons are set to the corresponding landmark powers and the weights of the links from the location of the Koala to the current location. We will call these weight vectors $\overrightarrow{LM_w}$ and $\overrightarrow{L_w}$, respectively.

At any point in time, the S-Neuron receives as input the current landmark powers $\overrightarrow{LM}$ as well as the current location $\vec{L}$ of the Koala.

The output is the inactivity , which is calculated by the inactivity function.

In addition to the landmark and location links, the S-Neurons are connected to some of the other S-Neurons as well. This linkage is dynamic. We will describe the process later in detail. The following figure of the S-Neuron contains these links, labeled $N_1,..,N_n$.

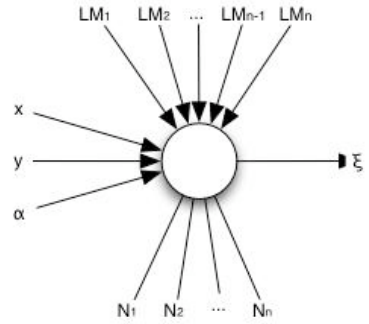

Figure 2: S-Neuron

Inactivity

Every neuron in the mental map has inactivity, which is the result of the inactivity function. The higher the inactivity is, the less the neuron claims to be representative for the current state. An inactivity of 0 is the minimal inactivity, which in turn means the neuron is maximal active. The range of the inactivity is [0,+);

Inactivity function

The inactivity of a S-Neuron is calculated as follows:

$$= \sum_{i=1}^{N} \left| LM_i \quad LM_{w_i} \right| + \sum_{i=1}^{3} \left| L_i \quad L_{w_i} \right|$$

= inactivity

= location weighting factor

N = amount of landmarks

(1)

Alternatively, we could also use the following formula. Note, that the threshold must be adapted, too.

$$' = \sum_{i=1}^{N} \left(LM_i \quad LM_{w_i} \right)^2 + \sum_{i=1}^{3} \left(L_i \quad L_{w_i} \right)^2 \qquad (2)$$

State determination

The network selects the neuron with the lowest inactivity to determine the actual state. This neuron is called the active S-Neuron. There can only be one active S-Neuron at a time, so we have a ‚winner takes all' network.

Dynamic linkage

S-Neurons are linked to each other. These links are important for the path finding task. The links are created whenever the active S-Neuron changes. Then a link is established between the former active S-Neuron and the newly active S-Neuron, when no link existed.
Links are always bidirectional.

An advanced version of the dynamic linkage is to use weighted links. Instead of just connecting two neurons, they will be connected with a specific strength. Whenever the active neuron changes, a link is established with weight w_0 when no link existed. Otherwise the weight will be increased according to some function. This sticks the neurons together.
What this can be useful for is described in the path finding chapter.

Dynamic creation of neurons

The network is able to dynamically create neurons. When the inactivity of the active S-Neuron is higher than a given threshold , the network will create a new S-Neuron. Dynamic linking applies as described above.

The state network

Such a state network is able to cartograph any space (easy to expand to 3D) as follows:

- It will create many neurons where it perceives many landmarks, because we can distinguish many different states.
- It will create some neurons when the Koala is moving blindly.

When setting to 0 it will create neurons only when perceiving landmarks. Nevertheless, it makes sense to create neurons in the ‚empty' space as well. We will see why in the chapter ‚Path finding'.

Thanks to the interlinkage of the S-Neurons, this network is very well suited for path finding and goal directed navigation. See the according chapters.

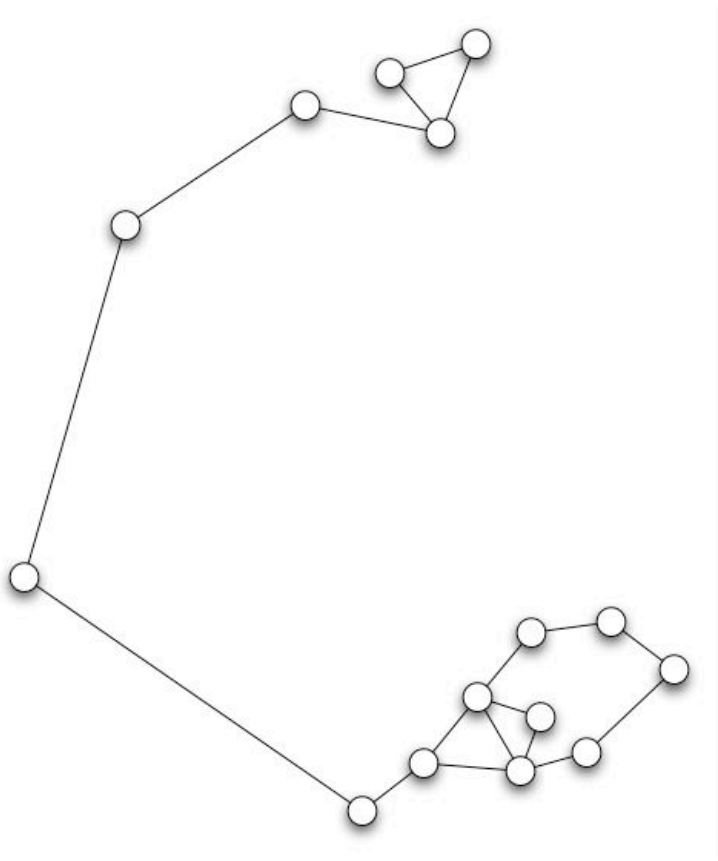

Figure 3: State network example

Error Correction

As I mentioned before, the state Network performs very well in a perfect environment. Of course, we don't have a perfect environment.
Even when we omit events like people disturbing the Koala or changing the environment, we find many errors. A detailed listing of the identified errors can be found in the document „Error specification".
Error correction is a complicated and sensitive task, so we need to expand the state network with two new concepts:

- The L-Neuron (landmark neuron)
- The K-Neuron (Koala neuron)

Together with the state network, the final network looks like this:

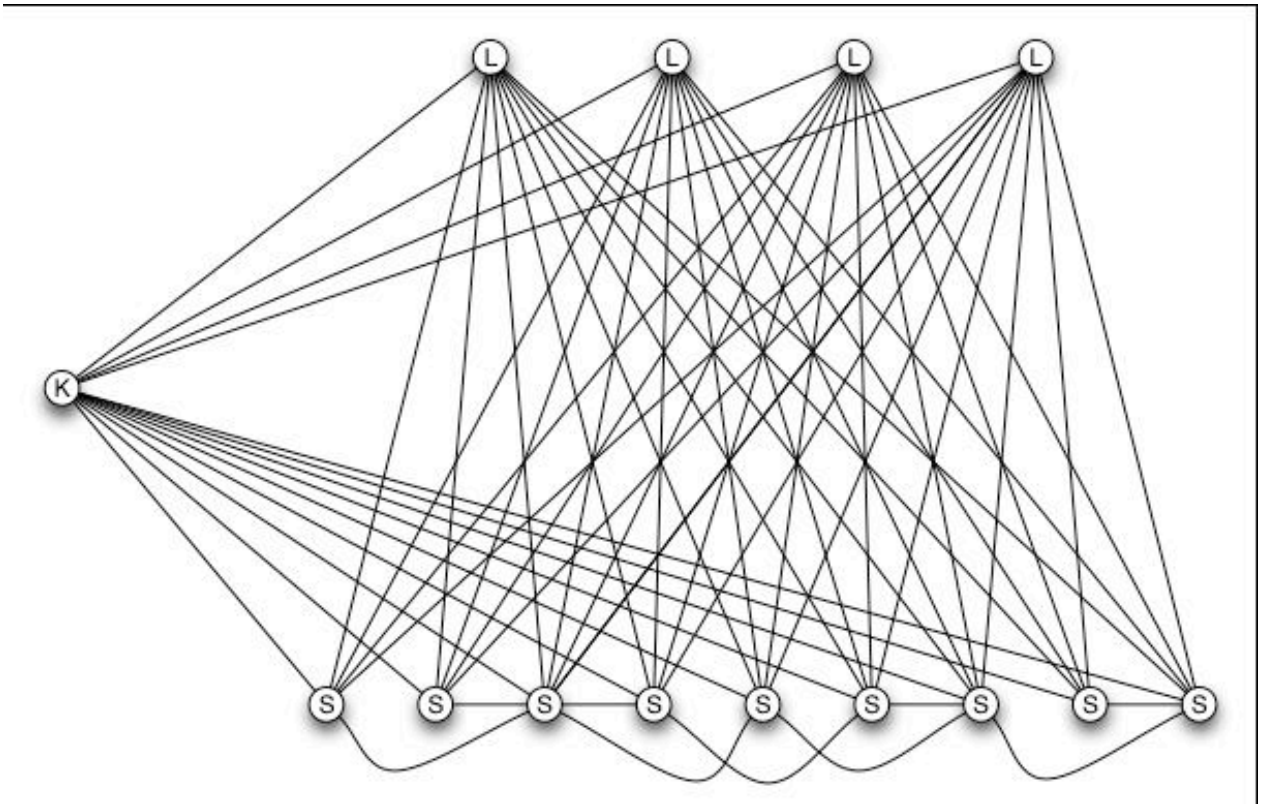

Figure 4: Fully meshed cortex (L = L-Neuron, K = K-Neuron, S = S-Neuron)

In Figure 4 every L-Neuron is connected to every S-Neuron. This fully meshed network contains amount L-Neurons multiplied by the amount of S-Neurons links. The amount of links scales poorly because it is exponential.

This can be optimized. Instead of connecting every L-Neuron to every S-Neuron, we just connect the L-Neurons, that have a power > 0 when the S-Neuron is created. The amount of links drops to a fraction of the fully meshed network, but the network could act erroneous. This is so, because it is more likely that a S-Neuron with just a few links will become the active neuron.
To avoid this error, we need to connect every S-Neuron to a neuron, which sums all landmark powers. The result of such a partially meshed network is exactly the same as with the fully meshed network. The advantage is, that when operating with many L-Neurons and S-Neurons, the amount of links and therefore the computational effort will be massively lower.

Another advantage is, that we do not need to change the S-Neurons at all.

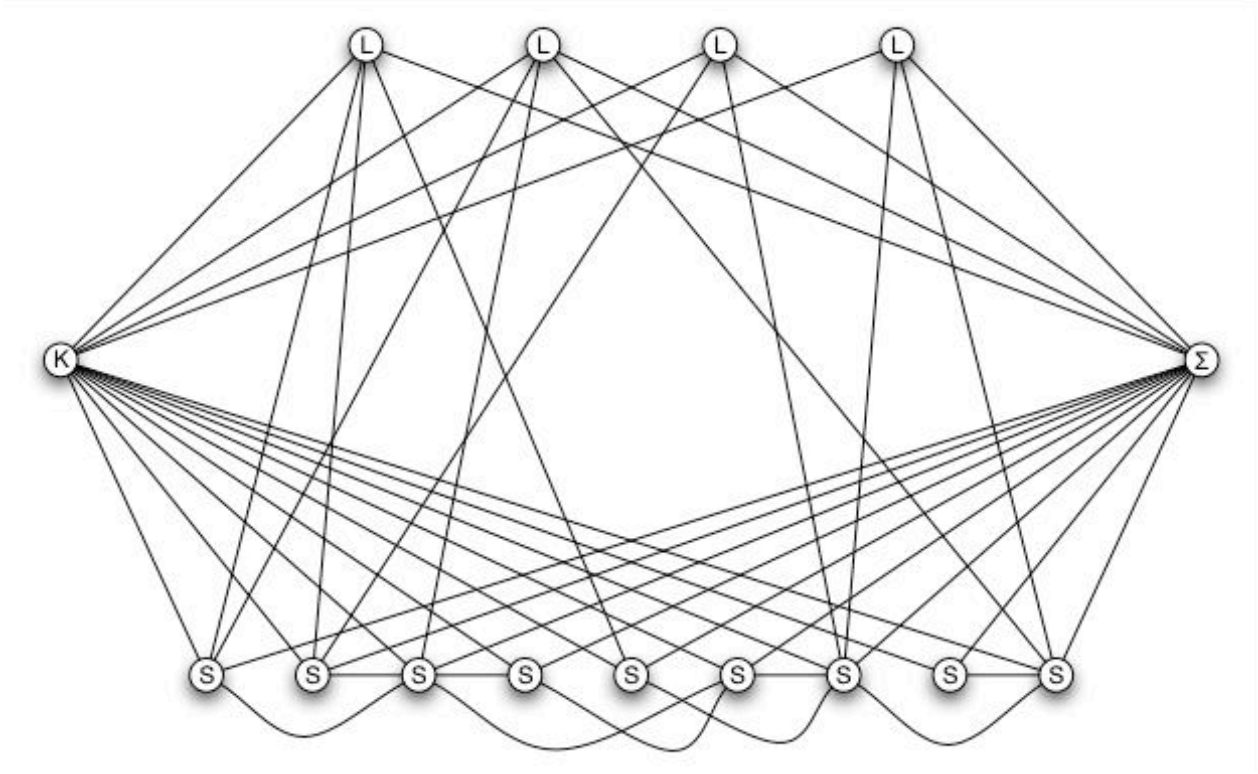

Figure 5: Partially meshed cortex (L = L-Neuron, K = K-Neuron, S = S-Neuron, = sums all landmark powers)

L-Neuron

The L-Neuron is responsible for everything related to landmarks. That is pretty much and it is unlikely to be done by one single neuron. So instead of L-Neuron we might call it an L-Network. The cortex creates one L-Neuron per landmark. Its main tasks are

- Landmark oblivion
- Landmark recognition error avoidance
- Landmark position estimation / error avoidance
- Landmark destination estimation / error avoidance
- Koala location correction

The L-Neuron receives a binary input. The values are just ‚seen' or ‚not seen', or mathematically {0,1}.

Landmark oblivion

As the Koala proceeds in space, it wont be able to see a landmark all the time, even if the landmark could be seen theoretically. To prevent the Koala from forgetting a landmark instantly, we need to apply an oblivion function. Ideally, the oblivion should not occur over time, but over the distance the Koala moved. For simplicity, we will take a time based oblivion function, but this is a subject of enhancement!

The currently used formula is:

$$of(p_{in}) := \left(\frac{p_{in}}{2}\right)^{m} \tag{3}$$

p_{in} = input landmark power [0,2]

The output of the *of* will be a value within [0,1].

The following plot shows the function for different *m*'s.

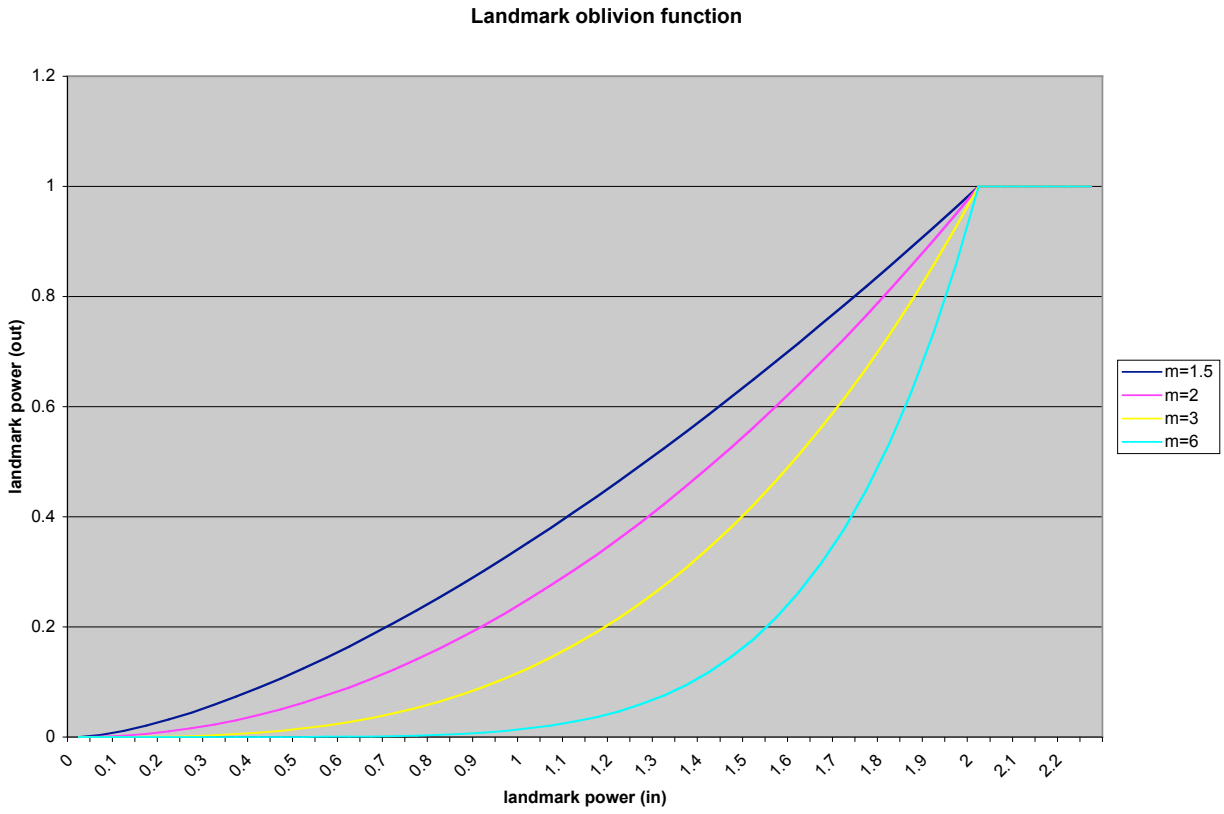

Figure 6: Oblivion function with different m

Landmark recognition error

The first error we will deal with is the so-called recognition error. The Koala perceives a landmark and identifies it wrong.
The solution is that the landmark neuron fires only when a landmark has been continuously identified over some time.
This goes hand in hand with the landmark oblivion function.

When a landmark is perceived, the potential of the neuron raises. When the landmark is not seen, the potential is reduced. Together with an activation threshold, the L-Neuron fires only when a landmark has been recognized for some time, and not too long ago.

The activation function (*af*) is a step function. The following plot shows the function for different *m*'s. The thresholds TH are chosen, so that the activation function will step when a landmark has been seen for more than 3 times. It can be calculated by applying the oblivion function n times.

$$TH = of_n(of_{n-1}(...(of_1(1))))$$
$$TH = of_n \circ of_{n-1} \circ ... \circ of_1(1)$$

Note: the plot is not good because the steps are not plotted correctly

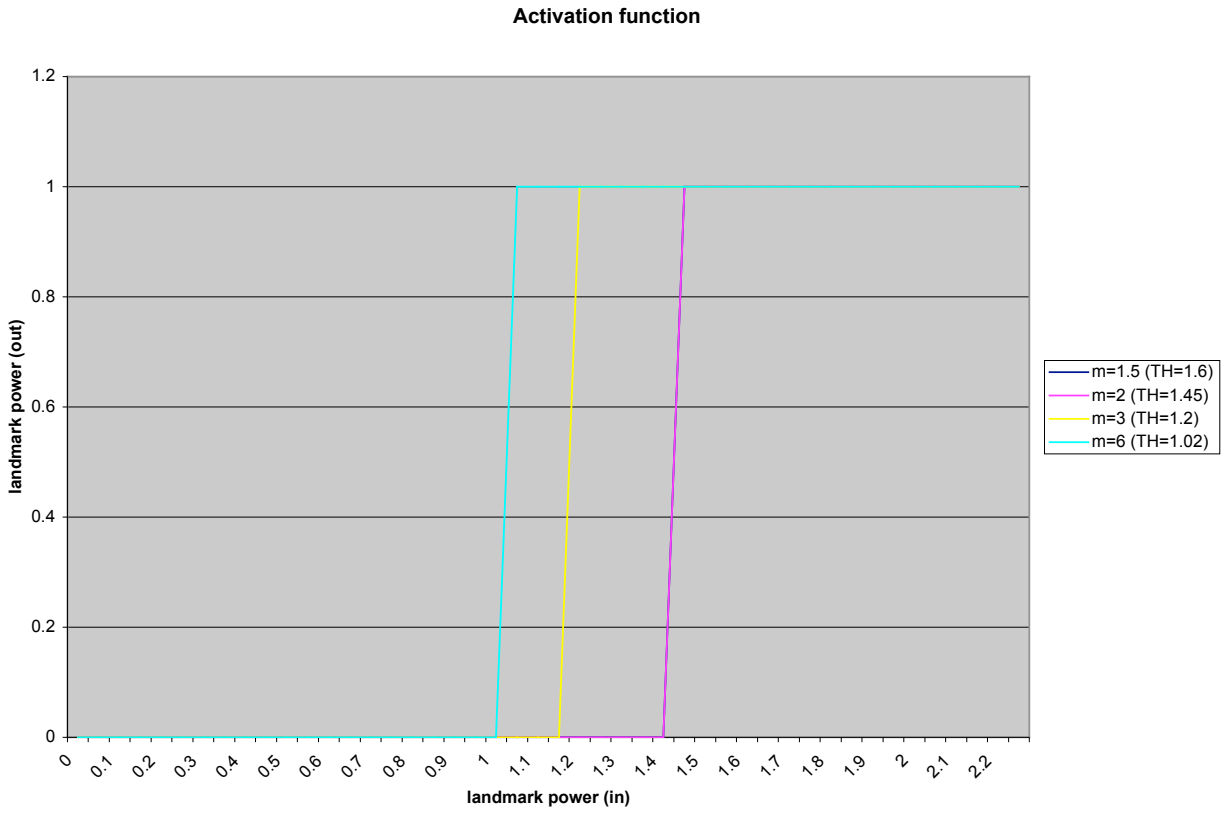

Figure 7: Activation function for different m after 3 times seen

We will call this the landmark perception function. It consists of the oblivion function concatenated with an activation function.

$$pf(p_{in}) := af(of(p_{in})) = af \circ of(p_{in}) \tag{4}$$

The following plot shows the perception function for different *m*'s.
Note: the plot is not good because the steps are not plotted correctly

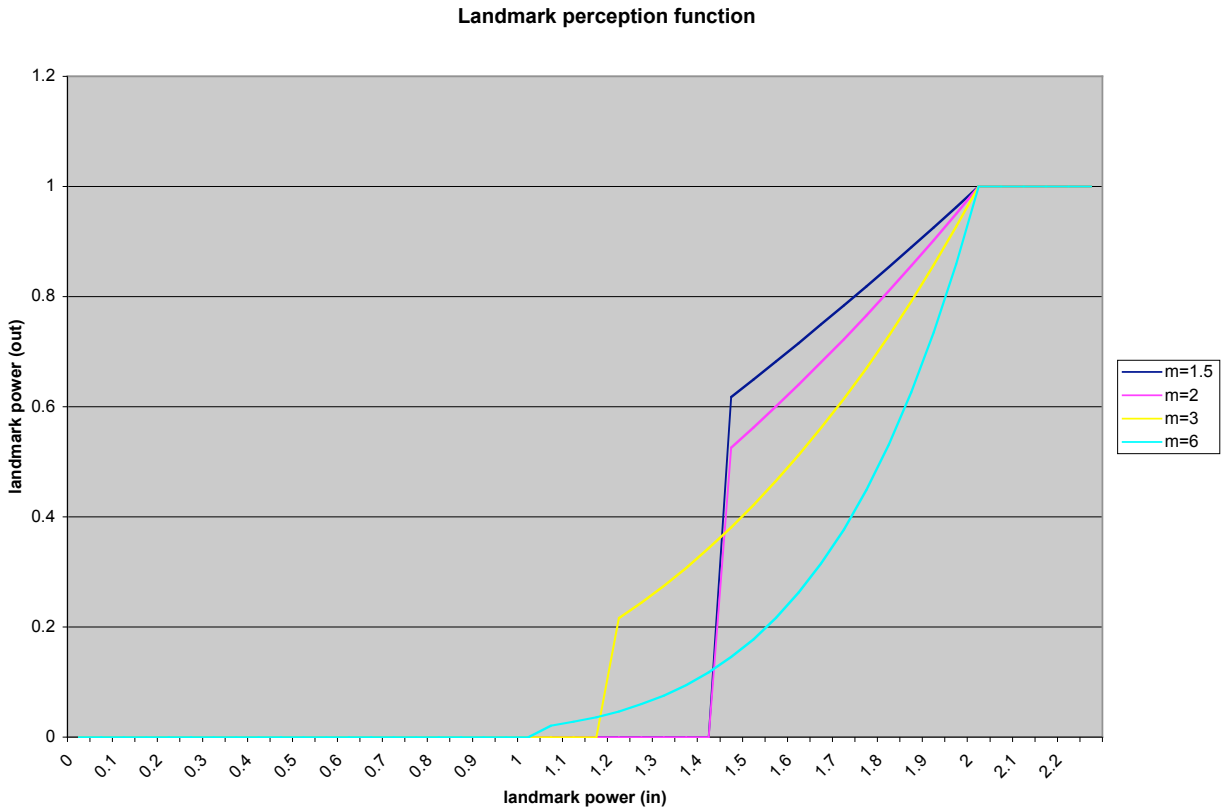

Figure 8: Landmark perception function with different m

This can be achieved by a feedback neuron. The schema may look like this:

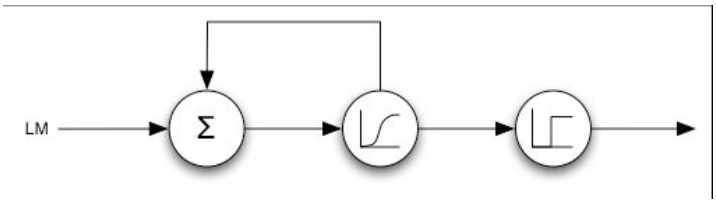

Figure 9: L-Neuron, perception function

Uncertainties

While the perception function of the L-Neuron is quite simple, there are still more problems to be solved. For this it is important to understand where we have uncertainties in the system. We can identify the following uncertainties:

- Position (x,y) of the Koala (provided by the path integrator)
- Orientation of the Koala (provided by the path integrator, gyroscope and magnetic compass)
- Perceived distance to the landmark (provided by the stereoscopic view of the cameras)
- Perceived angle towards the landmark (provided by the cameras)
- Estimated position (LMx, LMy) of the landmark (based on the uncertainty of the Koala position, Koala orientation and perceived distance)

To reduce the uncertainty of the position and the orientation of the Koala is actually the task of this network. So we will defer them to a later section.

The estimated position of the landmark seems to be most uncertain, because it bases on multiple uncertainties. For one single estimation, this might be true, but we can estimate the position many times and calculate the average. This means, the positional uncertainty of a landmark sinks over time (actually over times estimated).

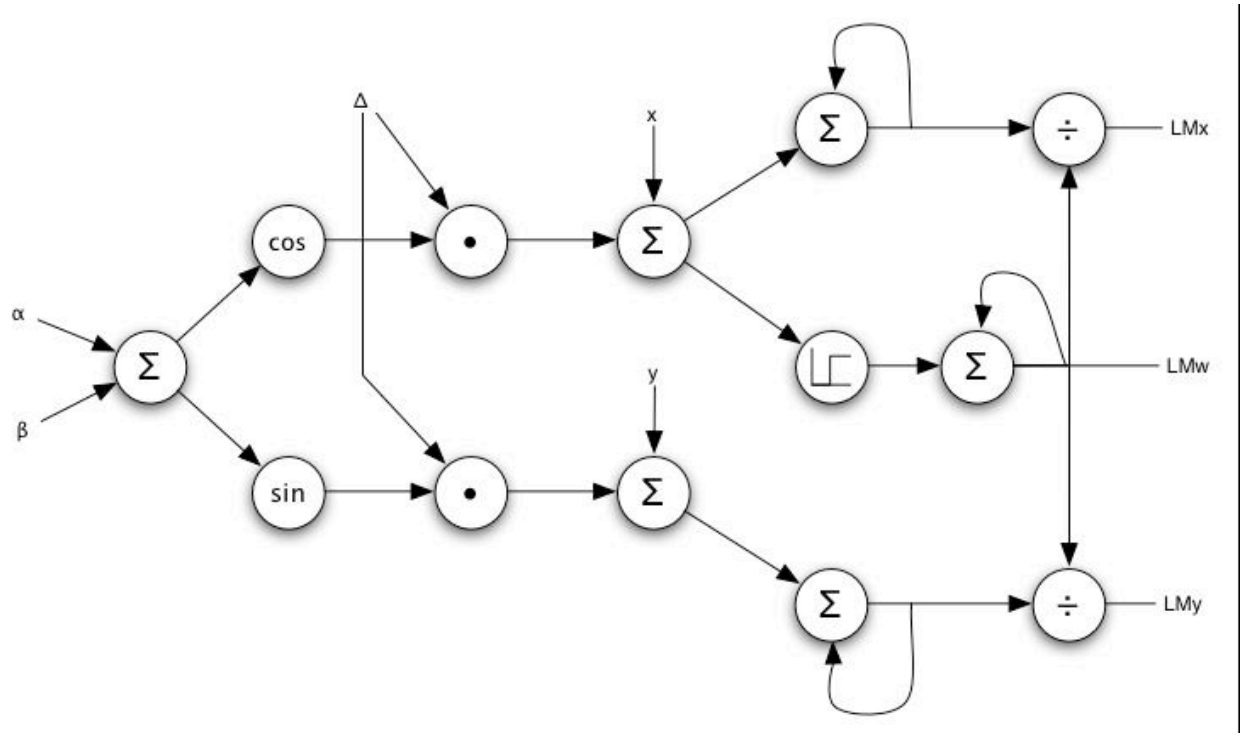

Figure 10: Landmark position estimation

The major problem is in fact the perceived distance. It is highly erroneous for far distances. In contrast to the estimated position it cannot be averaged over a longer period of time, because it changes with the movement of the Koala.
I am afraid I cannot give a good solution to this problem. We could average the distance over a short period of time. We have this time while the perception function delays the L-Neuron.

The following schema shows a possible way to do this.

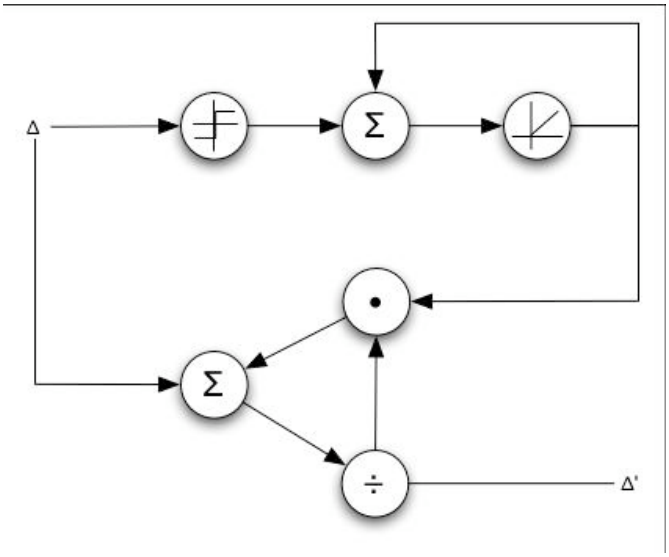

Figure 11: Distance averaging

Koala position estimation

The L-Neuron can easily be expanded to estimate the position of the Koala (x',y') based on the estimated landmark position (LMx,LMy) and the perceived distance to the landmark. Again we use the highly erroneous '.

NOTE: Maybe we could achieve better error estimation, if we take advantage of this. We use the same ', once positive and once negative.

The following schema enhances the landmark position estimation schema. The out coming value is now the estimated position of the Koala.

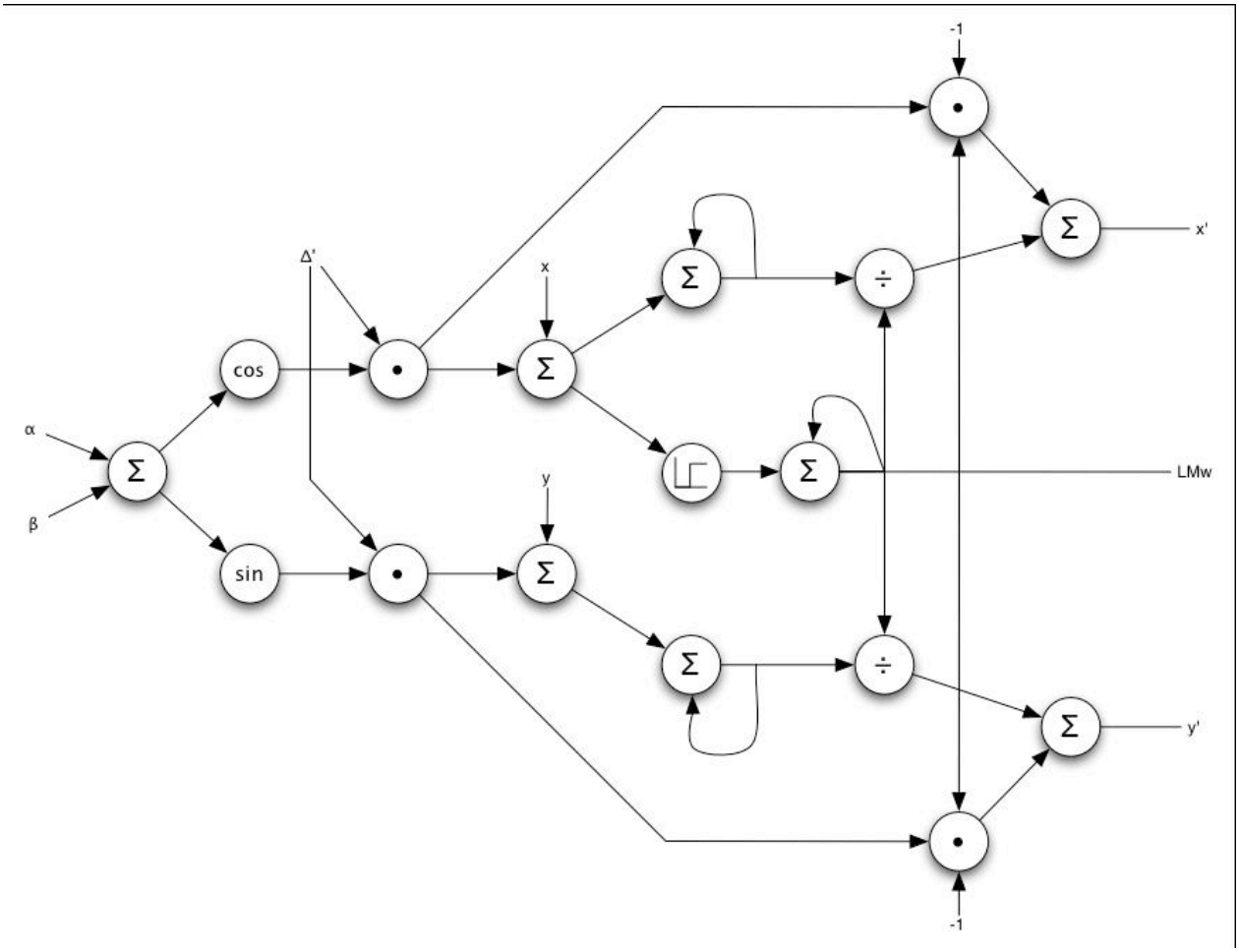

Figure 12: Koala position estimation

K-Neuron

The outputs of the L-Neurons are as many estimations of the position of the Koala as it perceives landmarks at a given point in time. The job of the K-Neuron is it to assemble all these estimations and to produce one definitive position.

As the different landmarks have different uncertainties of the estimated position, this must be considered in the calculation.

The original position, here called (x_0,y_0), must be taken in consideration as well. It will be weighted by the weight w.

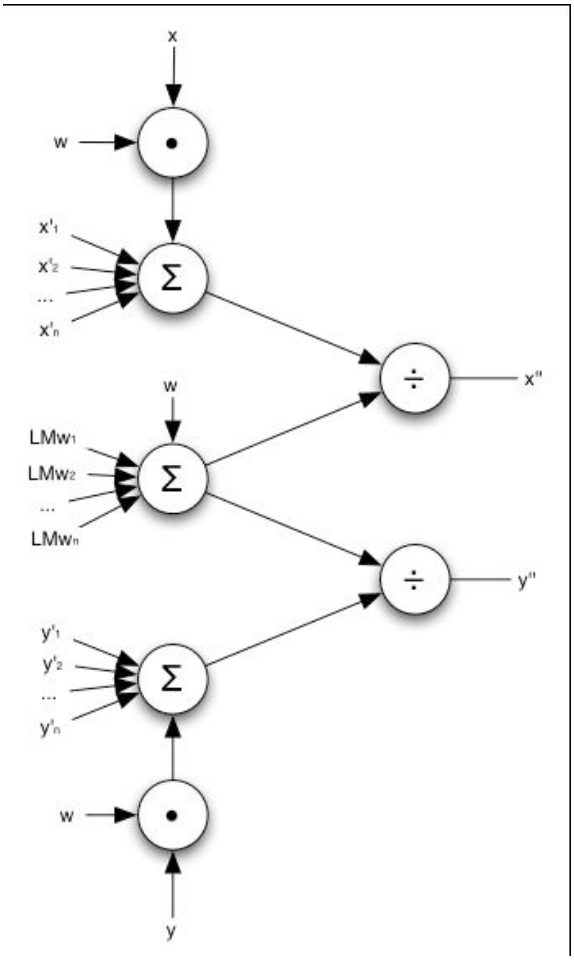

Figure 13: K-Neuron position assembly

Goal-directed navigation

With the state network, goal-directed navigation can easily be performed. To find the state where we want the Koala to move to (the goal), we simply feed the network with the landmarks the Koala will perceive when it reached the goal.
The network will then respond with the most likely state(s). The next step is to apply a graph searching algorithm such as A* to find the best path to the goal state. We reduced the problem to a path finding problem, which we will discuss in the next section.

Path finding

I will not cover this topic in detail here, because I do not know the details. This section is just an aggregation of first ideas and a topic of major changes.

When in a space without obstacles, the path-finding task would be easy. Simply head in the direction the next S-Neuron tells us.

In an environment with obstacles we will do the same. But when hitting an obstacle, we need to circumvent it. If we would not adapt the state network, we would hit the same obstacle again and again.

But how can it learn the presence of obstacles? For this task it is important to have neurons even in ‚empty' spaces (see „The state network").

The following path finding approach only works with weighted links.

I do not discuss the graph-searching algorithm here. Basically any algorithm might be used. The heuristics should at least include the distance between two nodes and the weights of the links.

Dynamic environment

When acting in a dynamic environment (multiagent), we need to be flexible. There might be doors that are open from time to time, so we cannot possibly be sure a link will lead us at any time to our goal.
To reflect this, we forget links over time and refresh them while using them. Unused links will vanish over time.

Links can be forgotten in three ways:
- Local oblivion
- Global oblivion
- Combined oblivion

Local oblivion

- Raise the link weight of the link the Koala used
- Sink the weights of the ones it did not (Only links connected to the active S-Neuron)

➔ Forget the links the Koala does not use

Global oblivion

- Raise the link weight of the link the Koala used
- Sink all others

➔ Forget about areas the Koala has not been for some time

Combined oblivion

- small global oblivion
- bigger local oblivion

➔ Forget the links the Koala does not use quickly, forget areas the Koala has not been for a long time slowly

We must be careful with link removal though. Otherwise we get separated networks with unreachable states and a graph search would be incomplete.
Such dead neurons could be removed from the network. This would mean, the Koala forgets landmarks over time (especially when using global/combined oblivion). This would be useful when landmarks have a dynamic behavior as well.

For the next section, we wont do any link removal, but set the weight to a vanishing small value instead.

Obstacles

Let us assume the Koala performed path finding and came up with a path from A to Z. The first direct segment leads from A to B. So it moves towards the location of B and hits an obstacle. It will perform these steps to circumvent the obstacle:

1) reduce the weight of AB
2) if the weight of AB <
    a. find the neighboring neuron of B, which is nearest to the current position. Lets call it X
    b. if A is already linked to X repeat step b
    c. link A to X with an initial weight > , but smaller than the usual initial weight w_0
3) repeat the path finding process with the new weights.
4) move

This method produces many links that could be removed. A better algorithm would take care of these.