# Knowledge Processing with Neural Networks

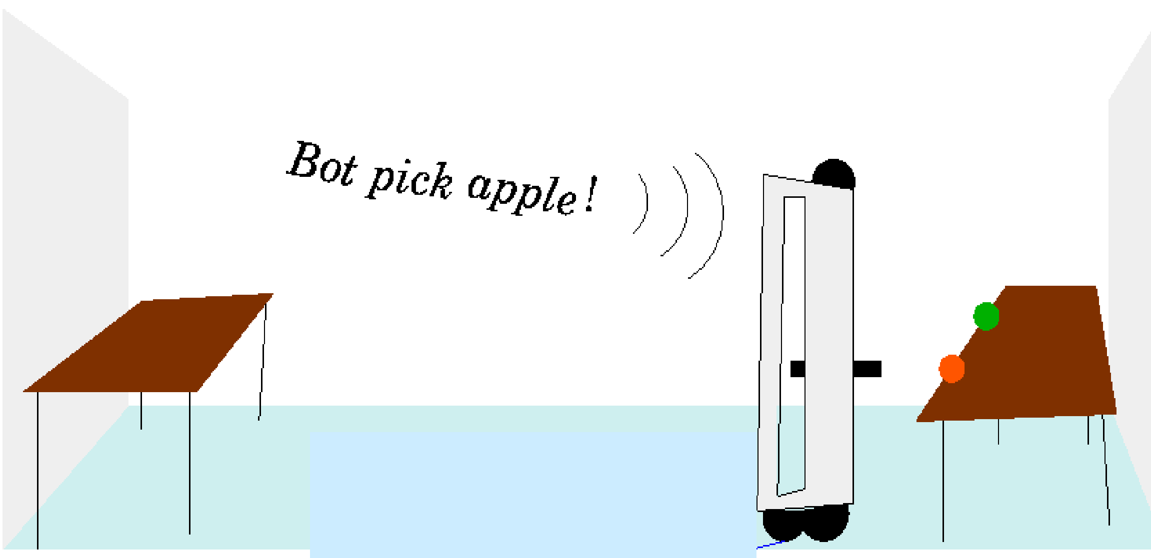
Lecture 10: Hybrid Neural Reinforcement Architectures for Approaching a Target

http://www.informatik.uni-hamburg.de/WTM/

# Search and fetch scenario: easy for a 3 year old child but hard for a robot

acoustic tracking;
language processing

visual object
recognition;
visual tracking




grasping

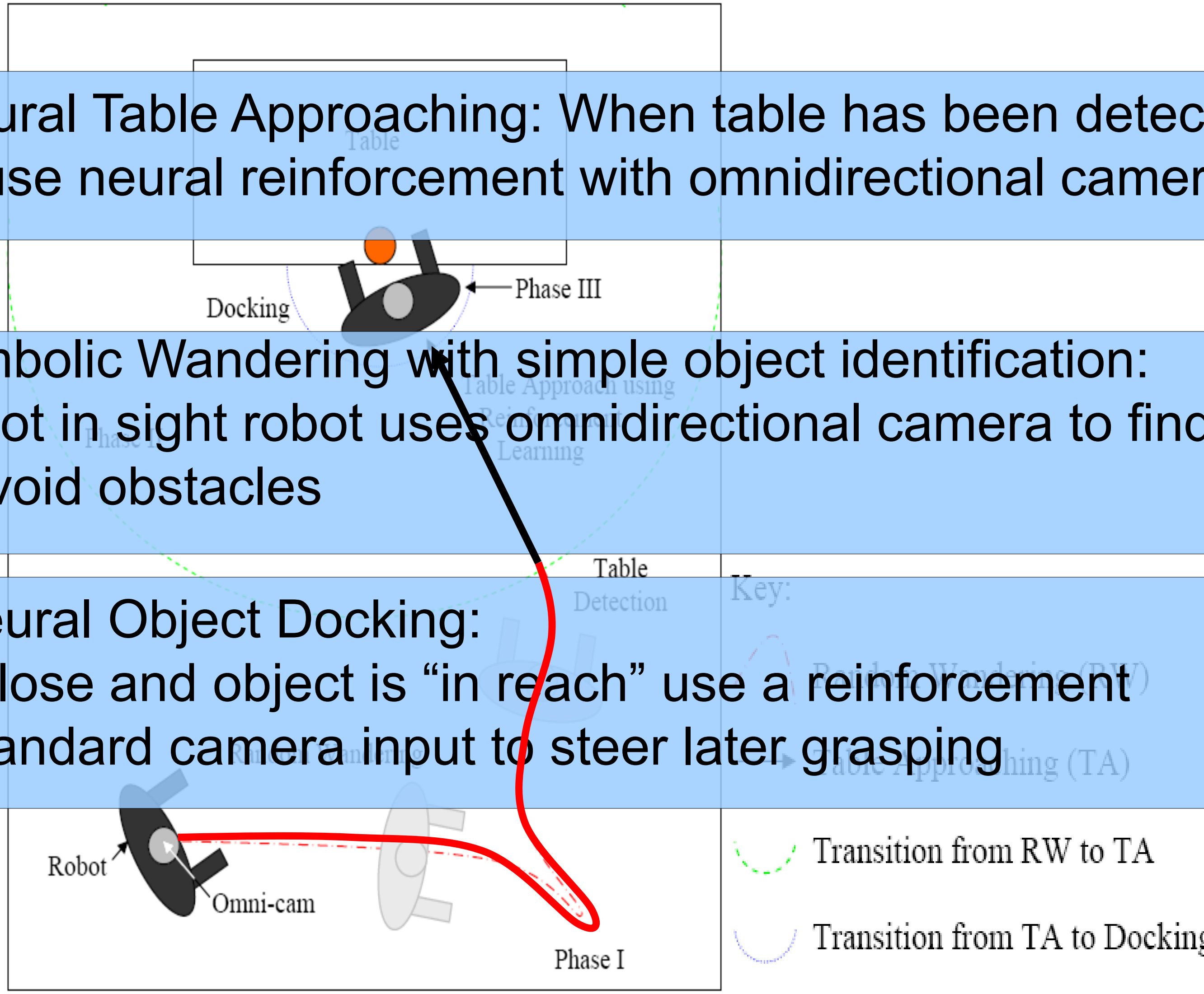
approaching: wandering, searching,
table identification, docking

# Hybrid Approaching

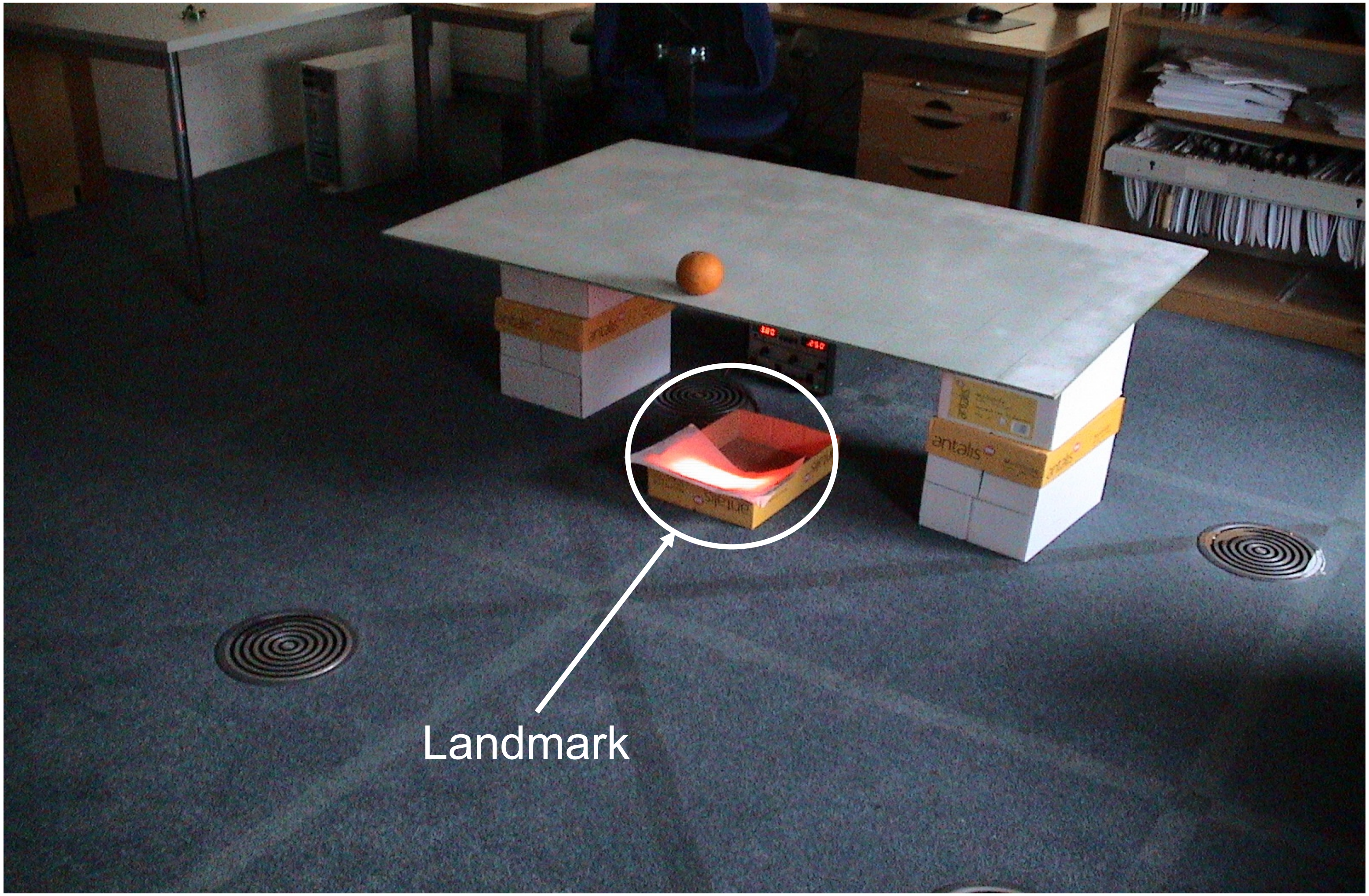
• Phase II – Neural Table Approaching: When table has been detected at large range use neural reinforcement with omnidirectional camera

• Phase I – Symbolic Wandering with simple object identification: When table is not in sight robot uses omnidirectional camera to find the table and avoid obstacles

• Phase III – Neural Object Docking:
When table is close and object is “in reach” use a reinforcement strategy with standard camera input to steer later grasping



# Environment setup



# Omnidirectional camera

Conical Mirror

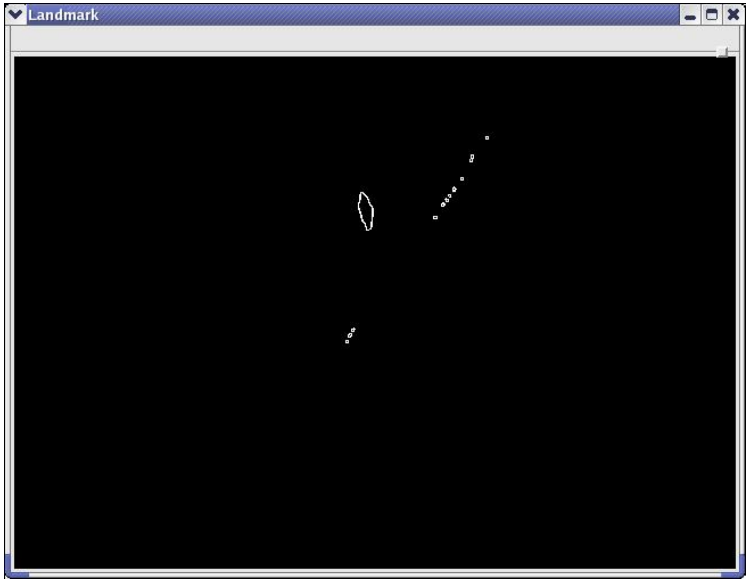
Camera

# Hybrid approaching: neural code in symbolic control algorithm

```
While robot is not at table
    Take picture (omnidirectional)
    Check if landmark is in sight
    If the landmark is not in sight
        Wander and avoid obstacles
    Else the landmark is in sight
        Pass control to neural actor critic for approaching
        If landmark is lost
            Wander and avoid obstacles
        Else robot is at the table
            Pass control to neural object docking
        End if
    End if
End While
```

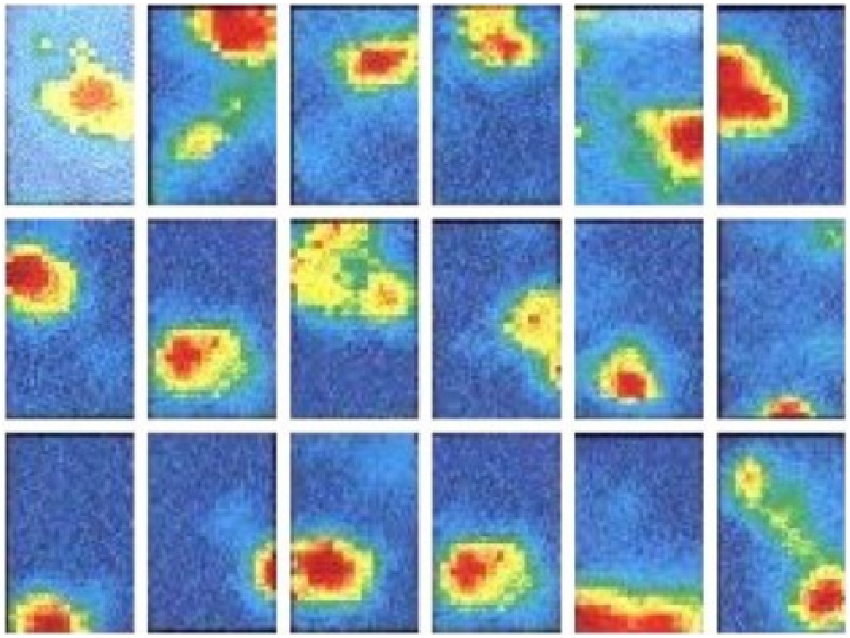
# Approaching the goal


Colour thresholding is performed followed by edge detection to leave outline of landmark:



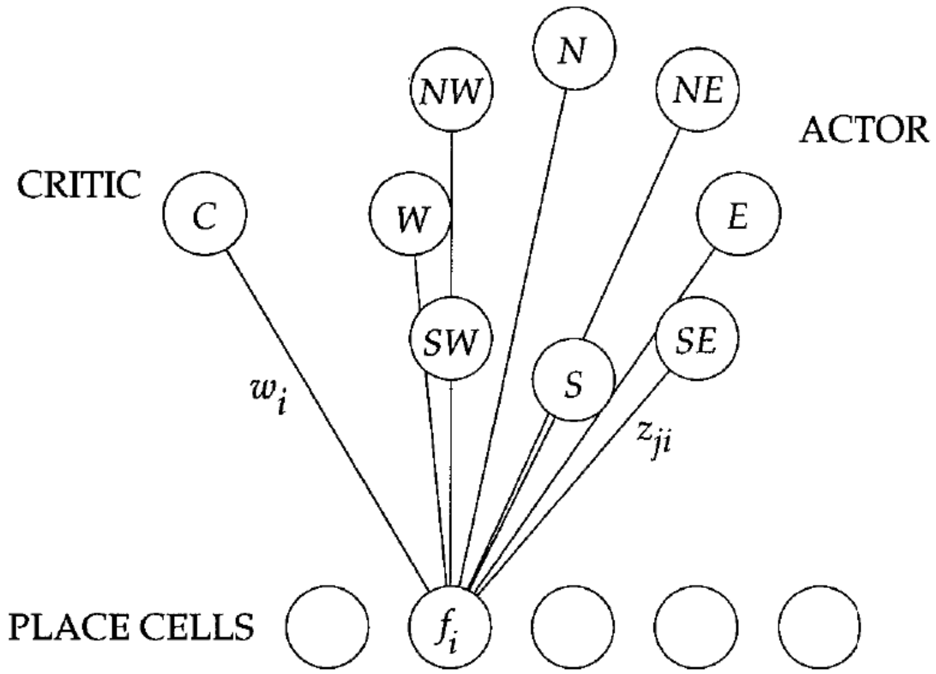
# Neural Actor Critic Learning

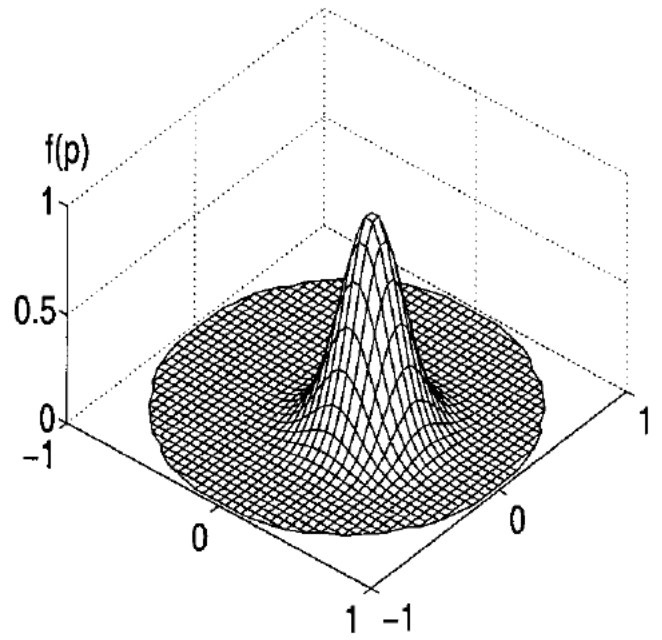
- How can ***place cells*** (neurons) in the hippocampus be used for navigation: they fire if the rodent is at a particular place



Place cell receptive fields (Wilson & McNaughton)

# Neural Actor Critic Learning (Foster et al)

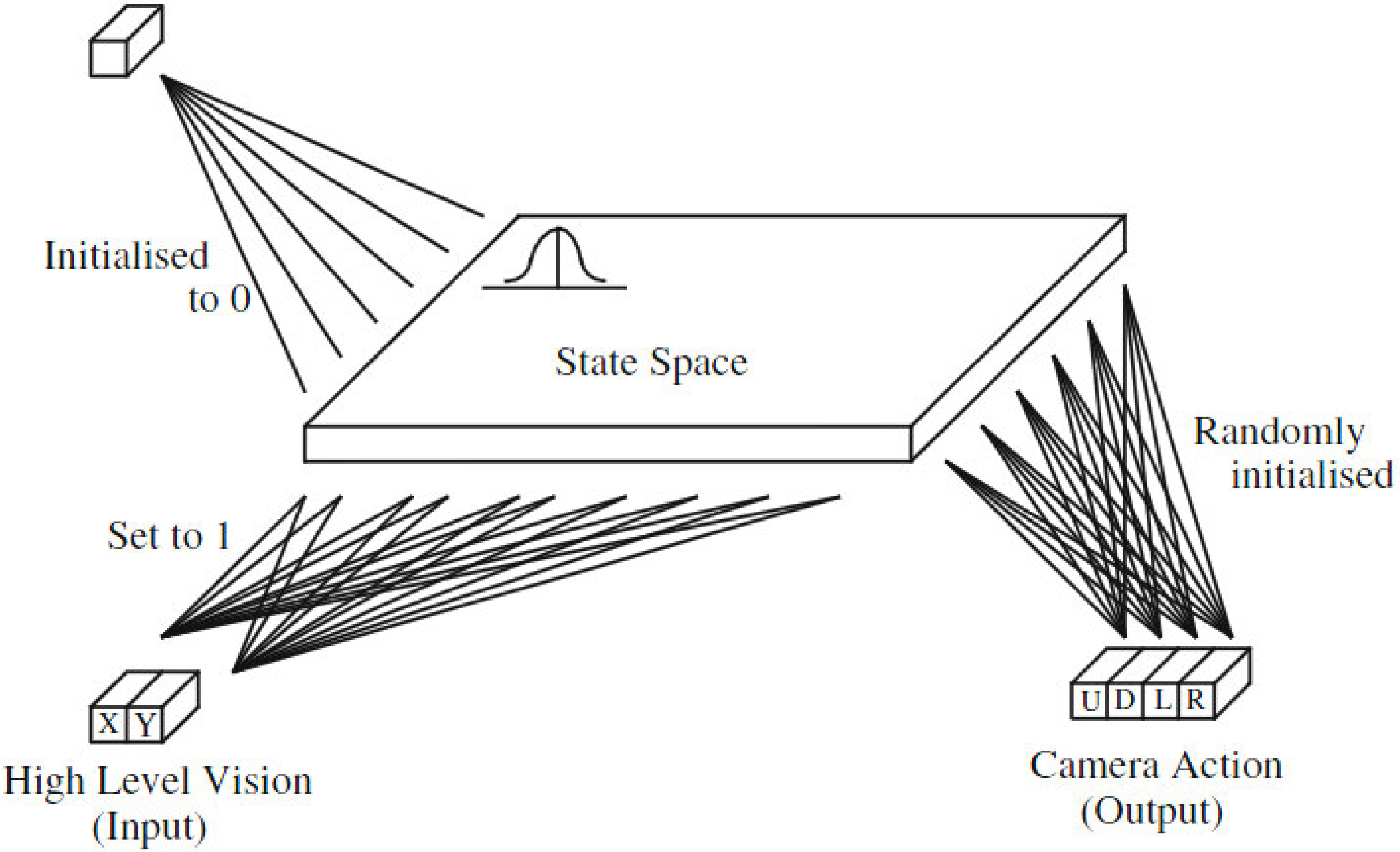


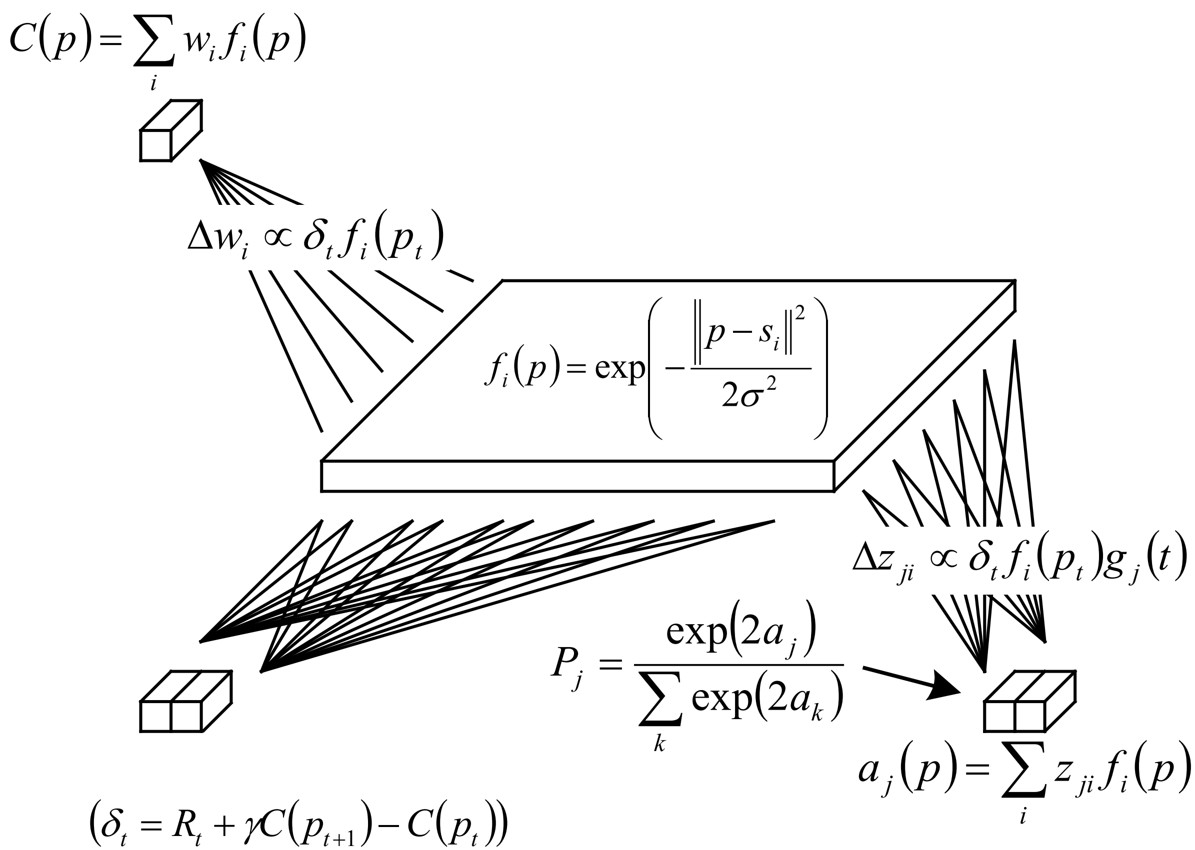

- Input layer of place cells projects to the critic cell, C.
- Output of C is used to evaluate behavior.
- Place cells also project to eight action cells for directions.

Example of a Gaussian place field (x and y axes represent location, z axis represents firing rate)

# Neural actor critic for approaching



# Guided Actor-Critic reinforcement learning algorithm




Rules for guidance:

If no reward was received from the critic:

- If the agent moved closer to the goal state the Actor is rewarded
- Otherwise the Actor is punished

[Foster et al 2000]

# Actor critic equations I

$$f_i(p) = \exp\left(-\frac{\left\|p - s_i\right\|^2}{2\sigma^2}\right)$$

Firing rate of place cell, p perceived location, si location where neuron i has maximal firing rate, σ radius of Gaussian

$$C(p) = \sum_i w_i f_i(p)$$

Critic firing rate, weighted sum of all of the firing rates

$$\delta_t = R_t + \gamma C(p_{t+1}) - C(p_t)$$

Calculated prediction error, Rt is1 when robot at goal location; then Cpt+1 is 0

$$\Delta w_i \propto \delta_t f_i(p_t)$$

Critic weight update proportional to firing rate and error

[Foster et al 2000]

# Actor Critic Equations II

$$a_j(p) = \sum_i z_{ji} f_i(p)$$

Actor firing rate, weighted sum of activations of surrounding place cell to the current location, z is weight between hidden unit and action

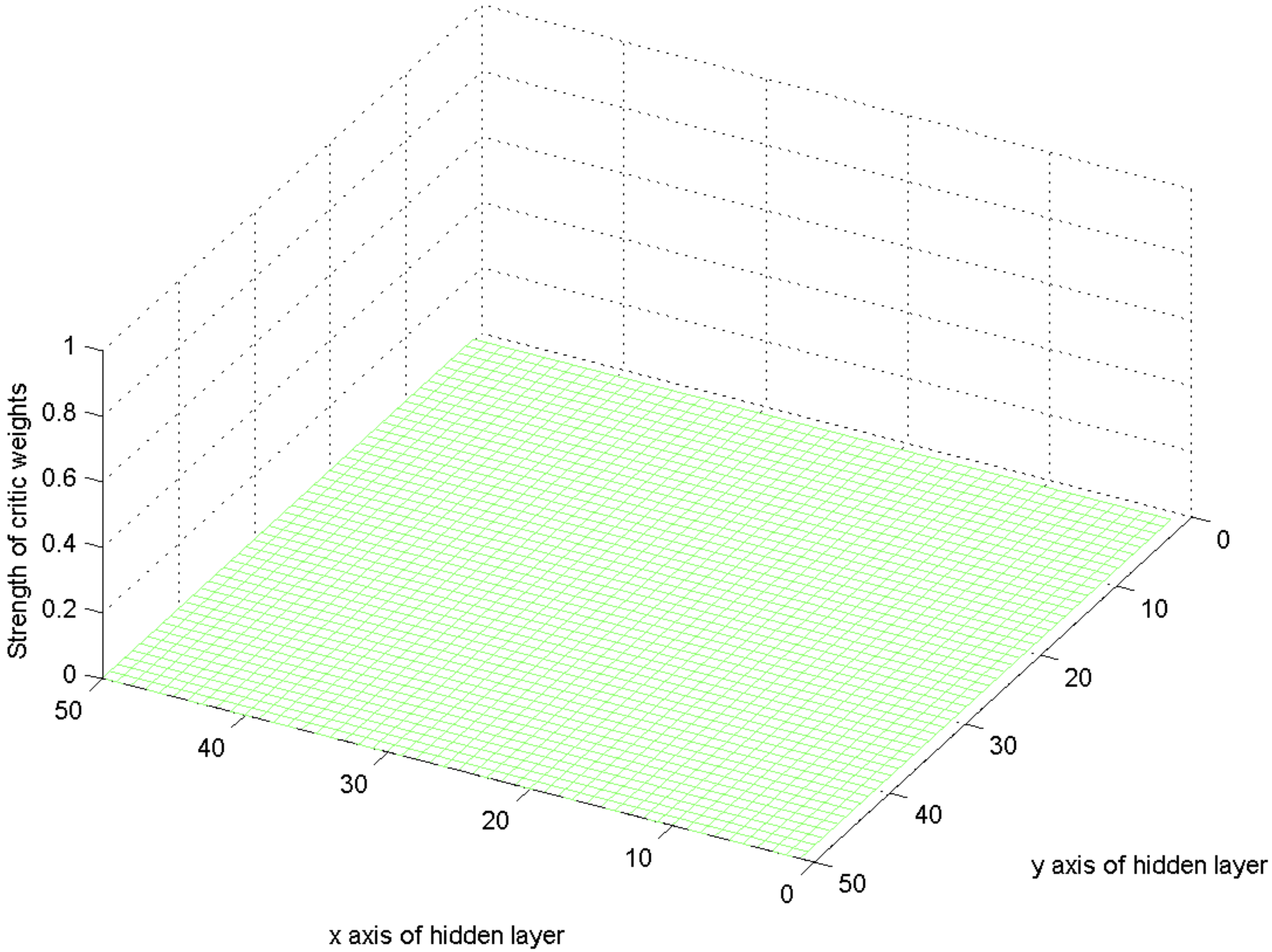
$$P_j = \frac{\exp(2a_j)}{\sum_k \exp(2a_k)}$$

Probability of any given action, firing rate of that actor neuron divided by sum of firing rate of all actor neurons

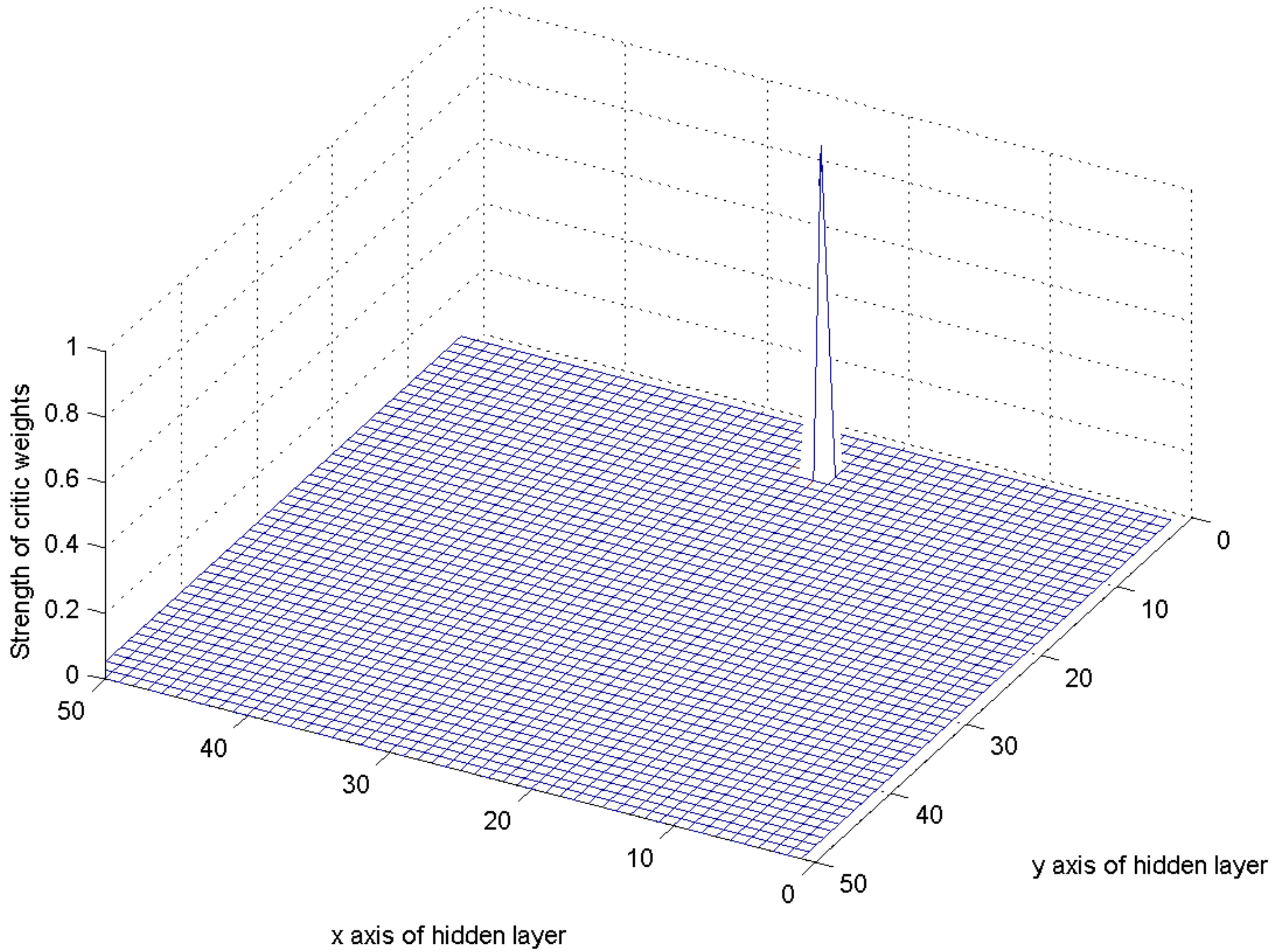
$$\Delta z_{ji} \propto \delta_t f_i(p_t) g_j(t)$$

Actor weight update

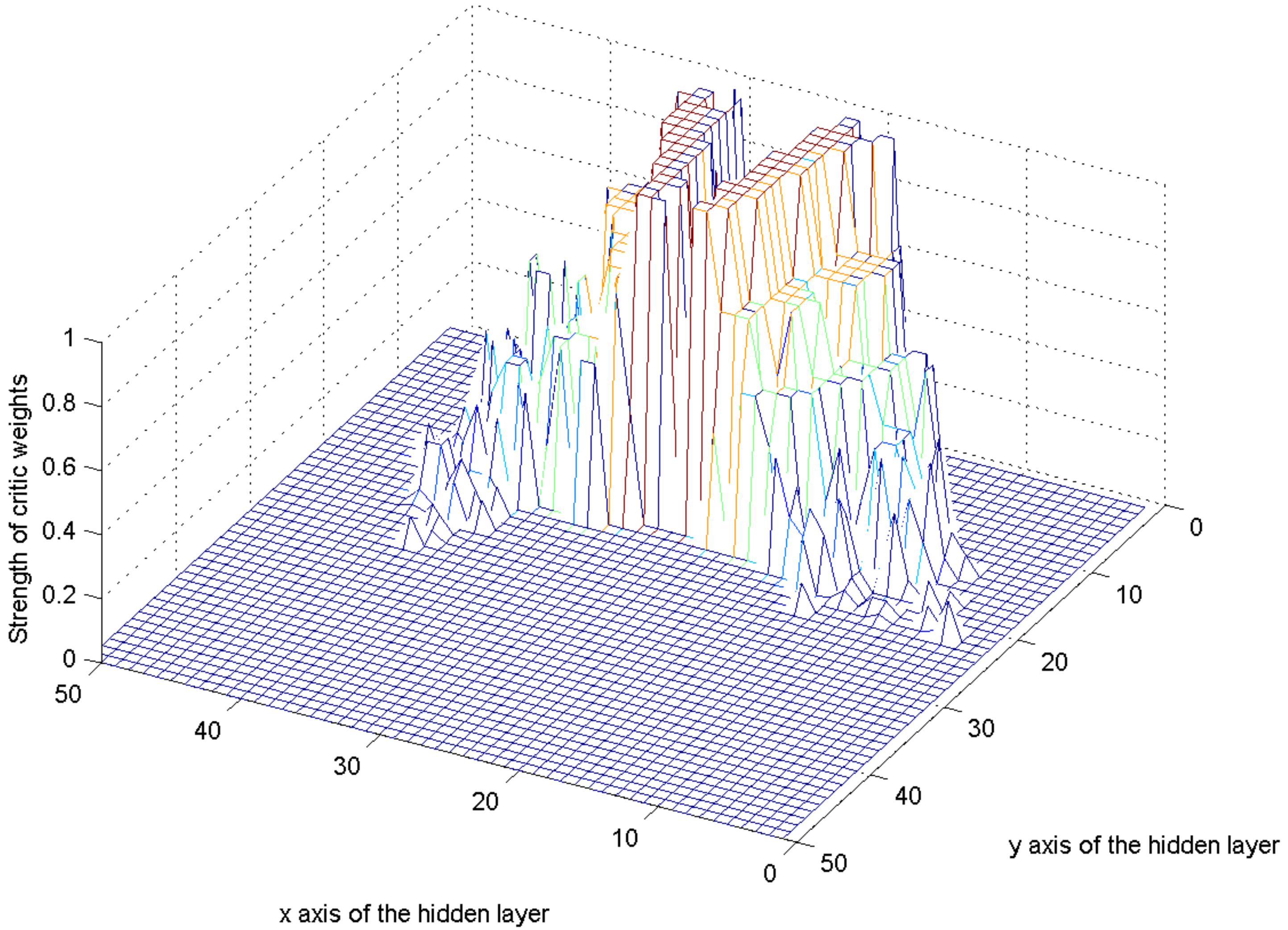
# Training of the critic (0 samples)



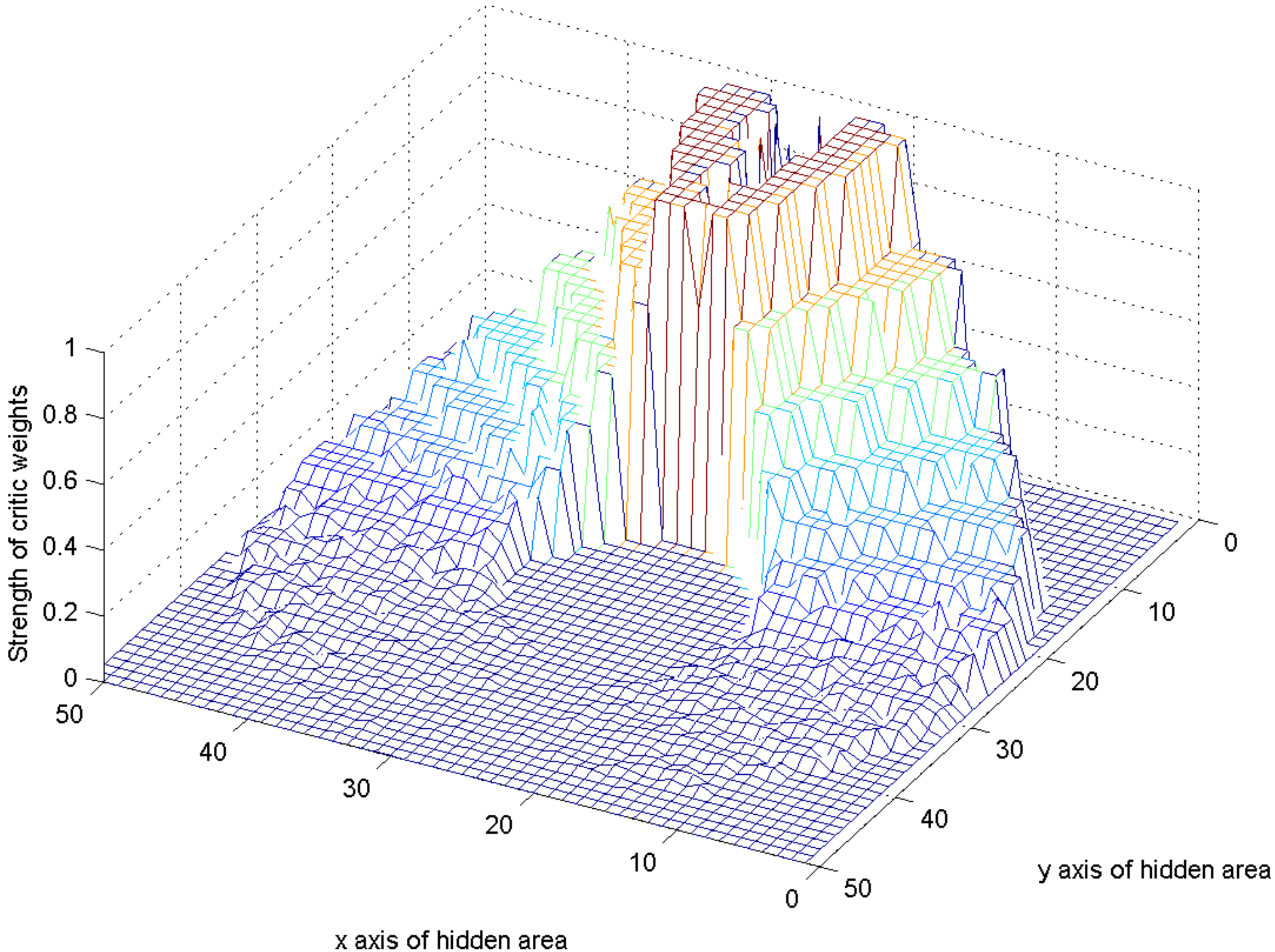
# Critic after 1 sample



# Critic after 1 epoch of 500 samples



# Critic after 50 epochs

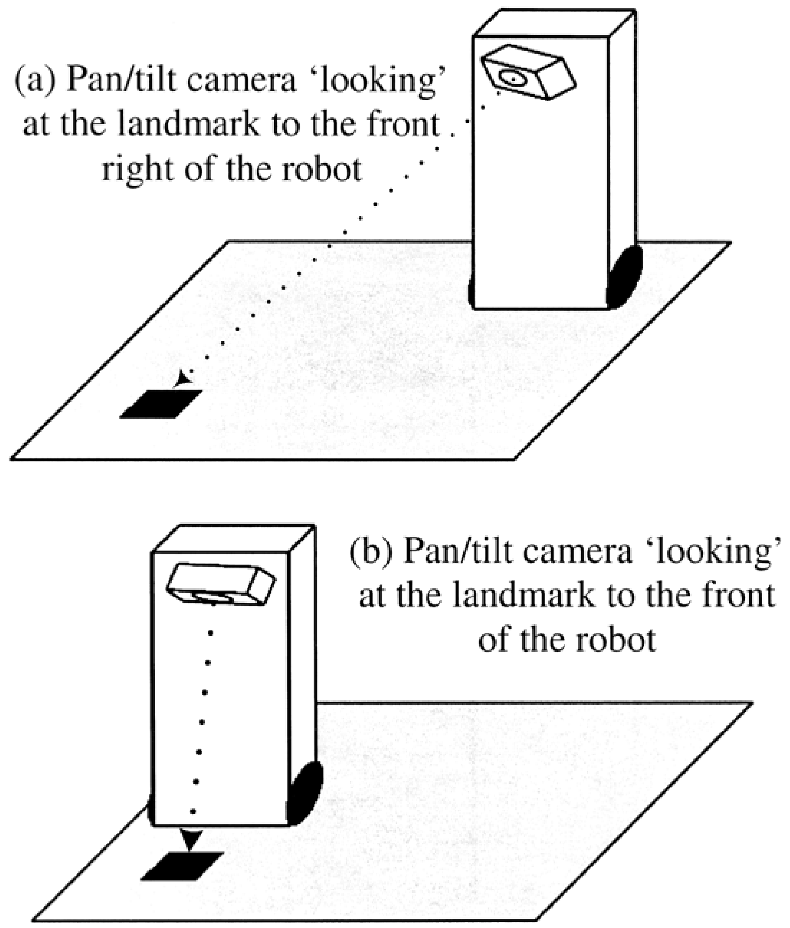


# Robot view

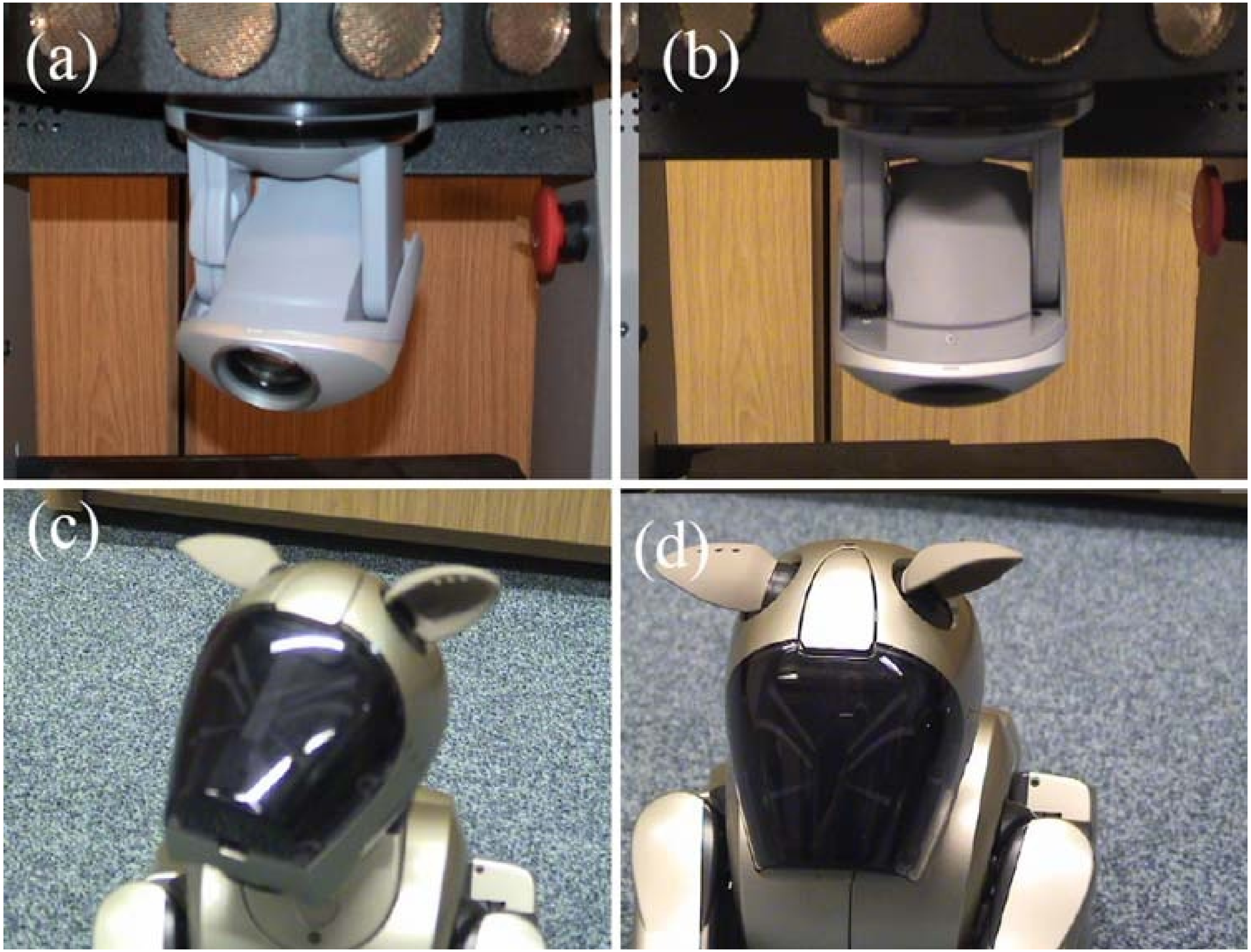
# Extension and example: Modular actor critic architecture motivation

- Is it possible to develop a ***platform-independent*** and ***dynamically coupled*** neural architecture based on the Actor-Critic learning algorithm, to control different robotic platforms equipped with a pan / tilt camera and movement capabilities?

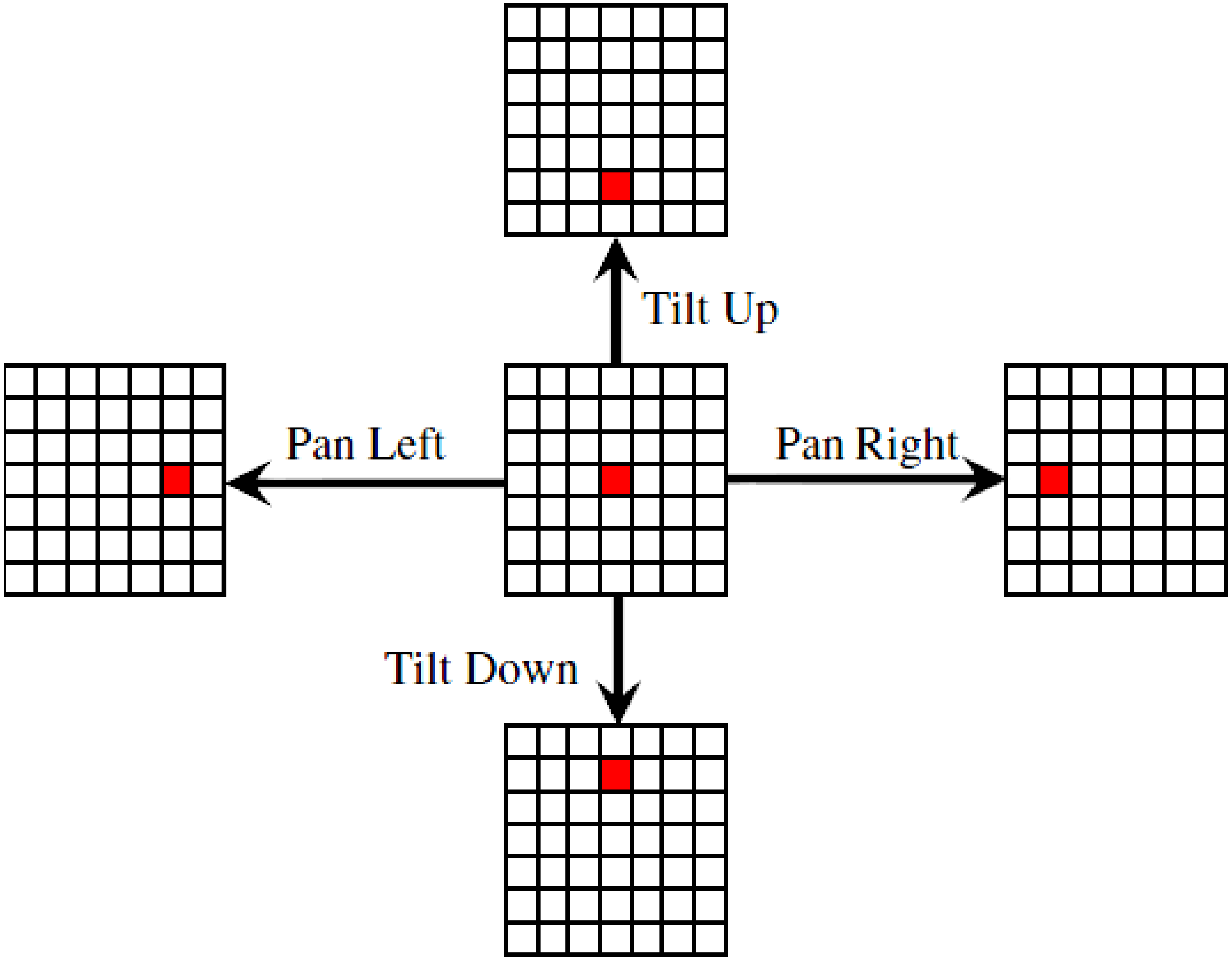
# Pan/Tilt vision control



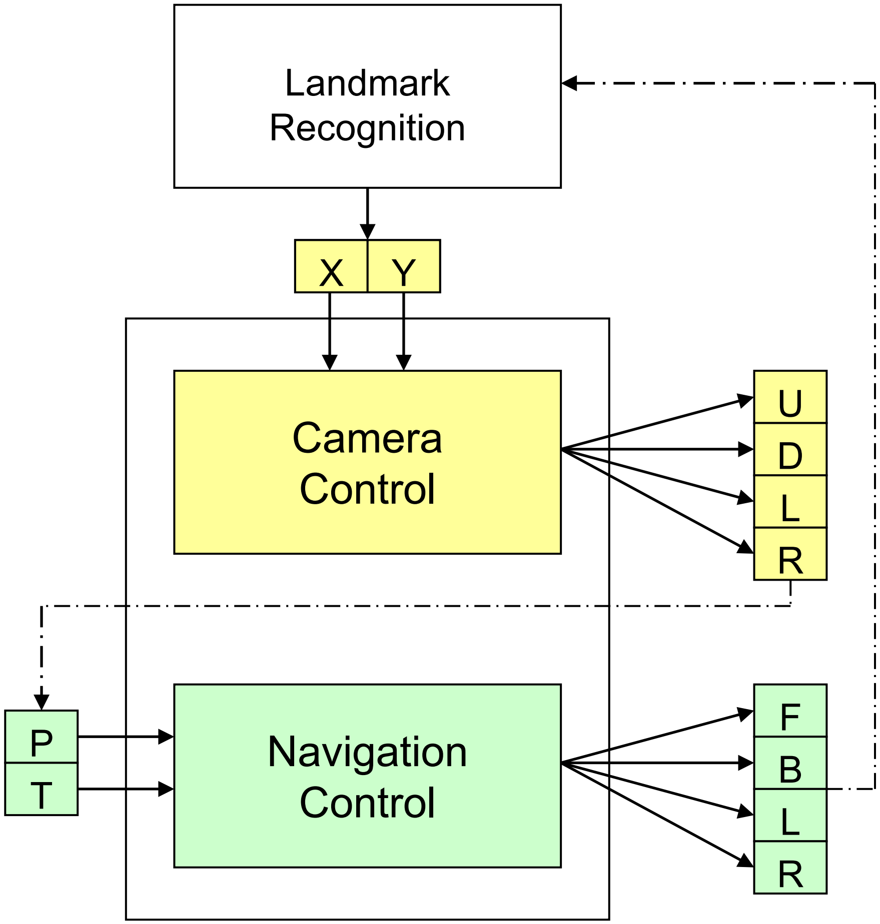
# Camera alignments for two robot types



# Effects of camera movement on landmark location



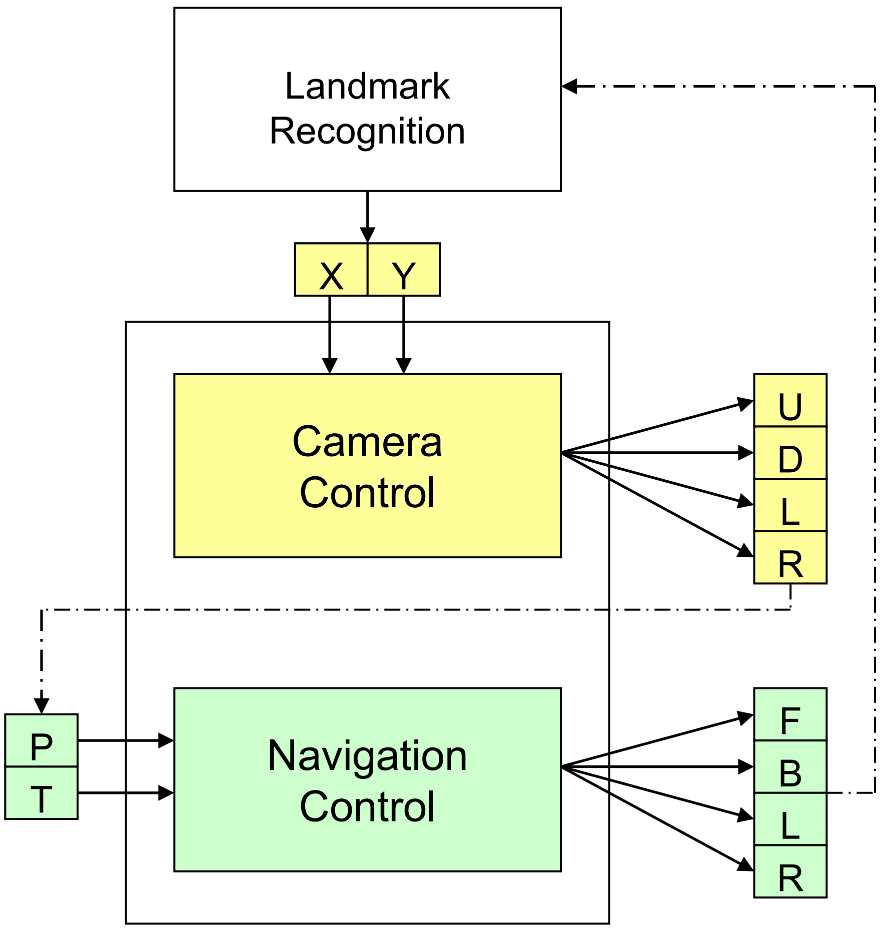
# Development of the MAC Architecture (Muse et al.)

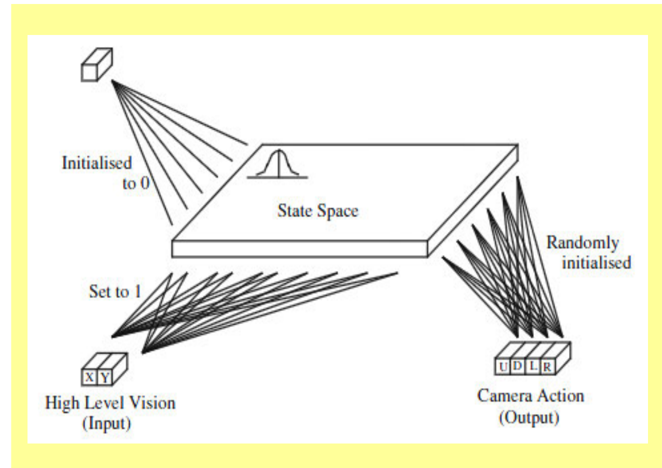


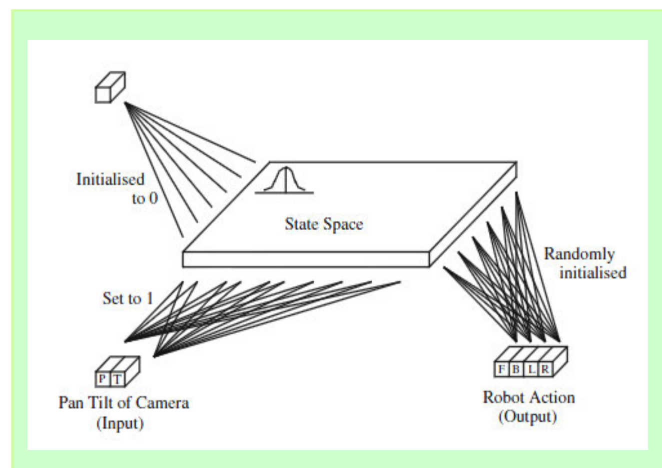

Key:

| | |
|---|---|
| X = | X coordinate of the landmark in the camera image |
| Y = | Y coordinate of the landmark in the camera image |
| U = | Tilts the camera upward |
| D = | Tilts the camera downward |
| PL = | Pans the camera left |
| PR = | Pans the camera right |
| P = | Pans angle of the camera |
| T = | Tilts angle of the camera |
| F = | Moves the robot forward |
| B = | Moves the robot backward |
| RL = | Rotates the robot left |
| RR = | Rotates the robot right |

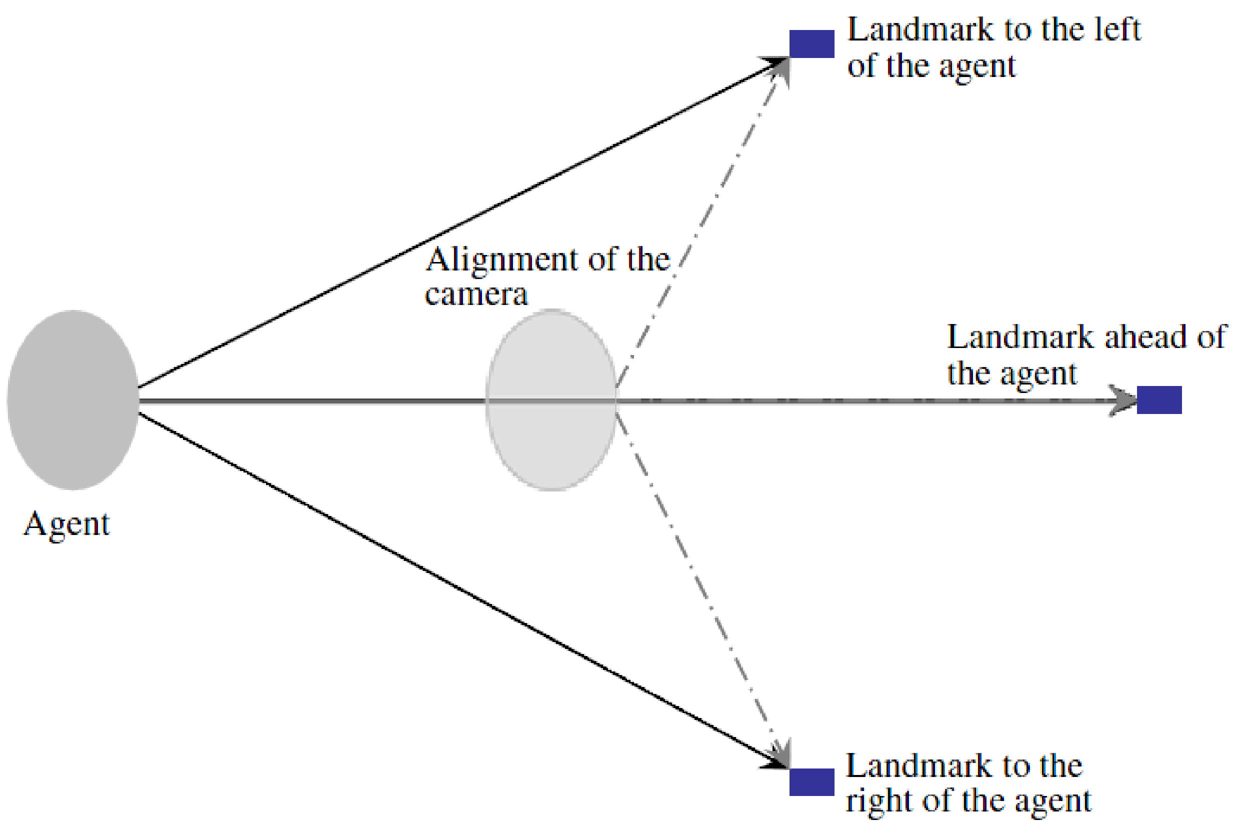
# Development of the MAC architecture: Separate control



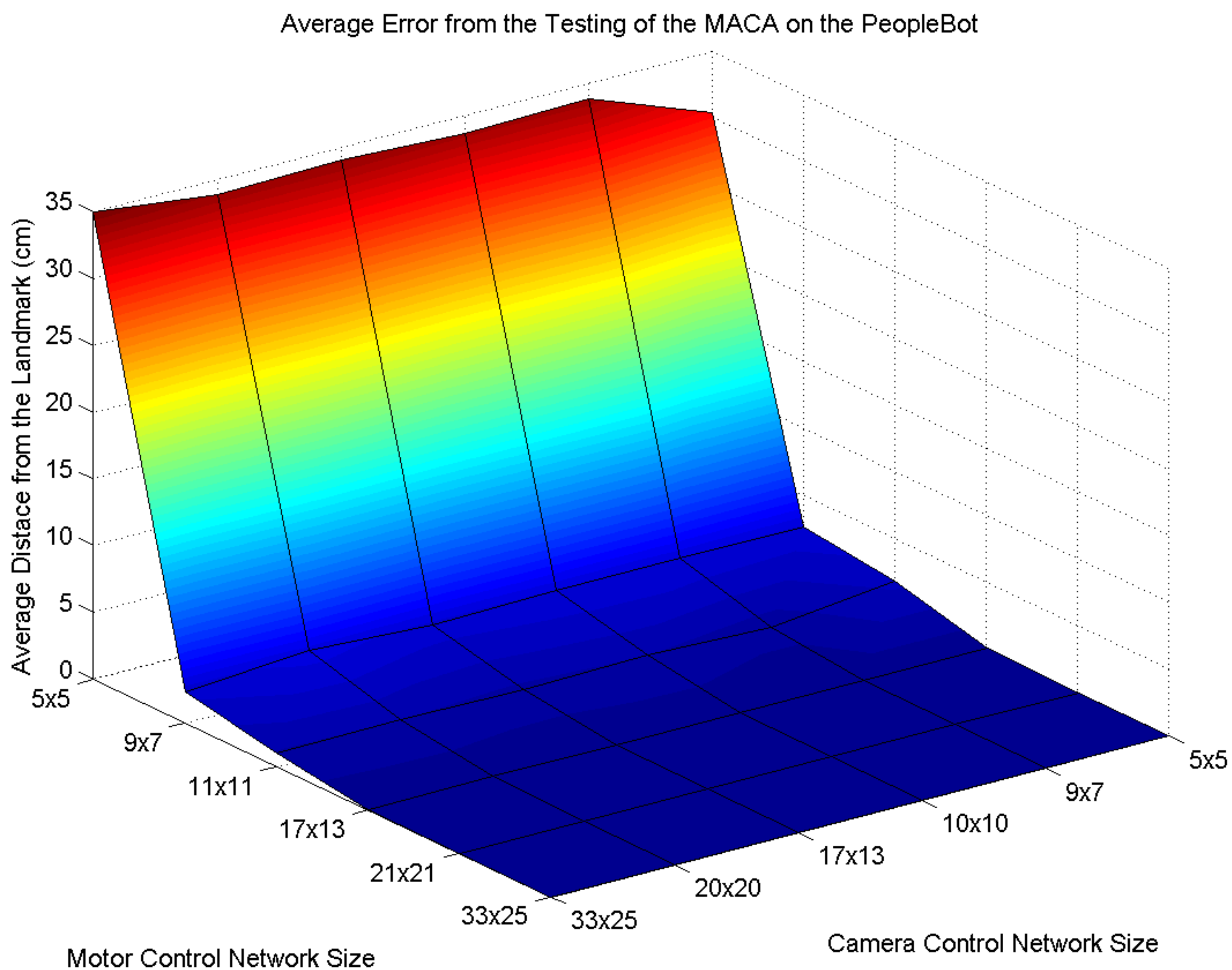



# Effects of camera movement while robot moves forward

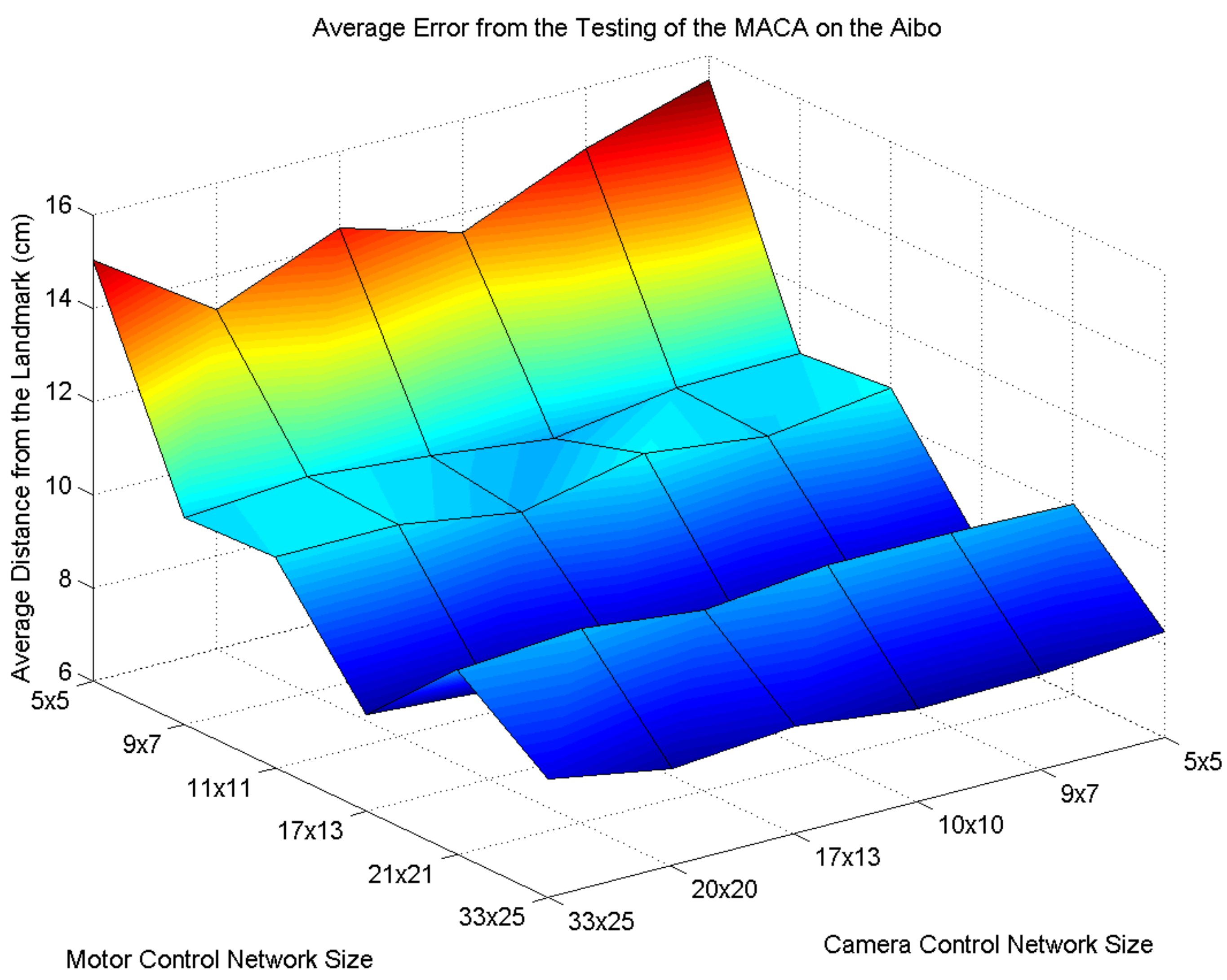



With the landmark to the left, when the robot agent moves forward the camera needs to pan left to keep track of the landmark.

# Results (1)

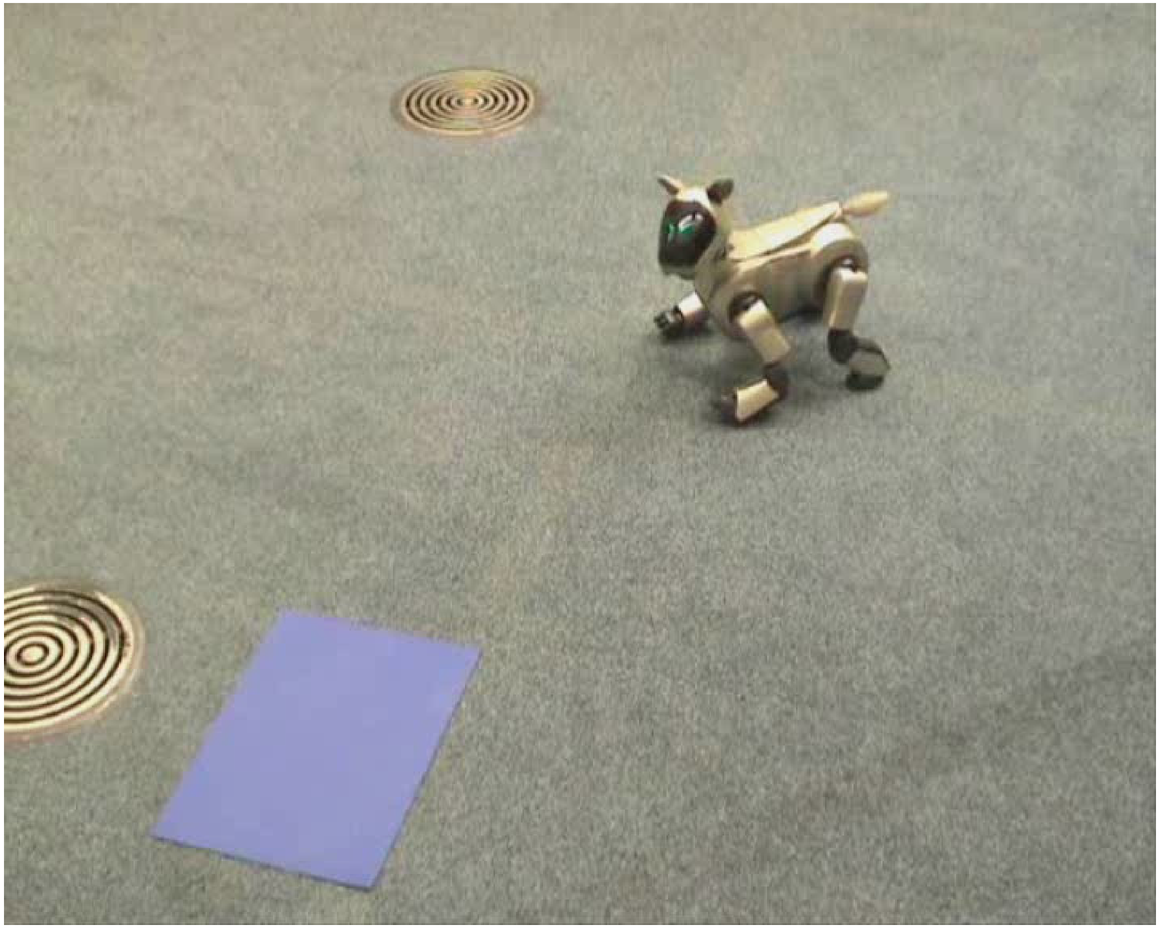

Average Error from the Testing of the MACA on the PeopleBot
Average Distace from the Landmark (cm)
35
30
25
20
15
10
5
0
5x5
9x7
11x11
17x13
21x21
33x25
Motor Control Network Size
5x5
9x7
10x10
17x13
20x20
33x25
Camera Control Network Size

# Results (2)

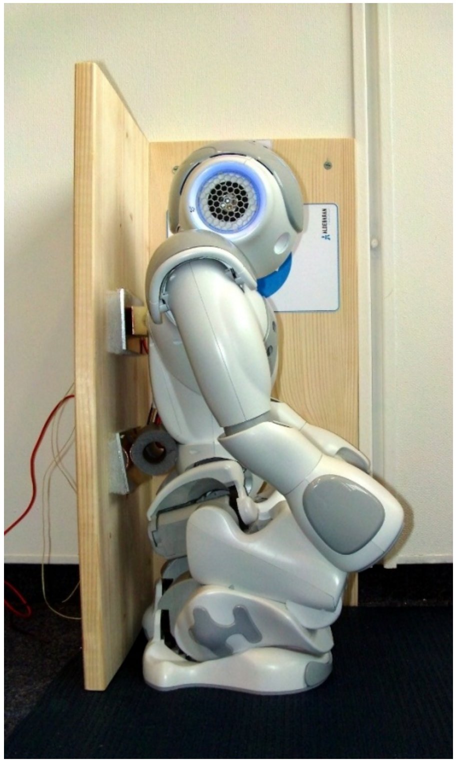

Average Error from the Testing of the MACA on the Aibo
Average Distance from the Landmark (cm)
16
14
12
10
8
6
5x5
9x7
11x11
17x13
21x21
33x25
Motor Control Network Size
5x5
9x7
10x10
17x13
20x20
33x25
Camera Control Network Size

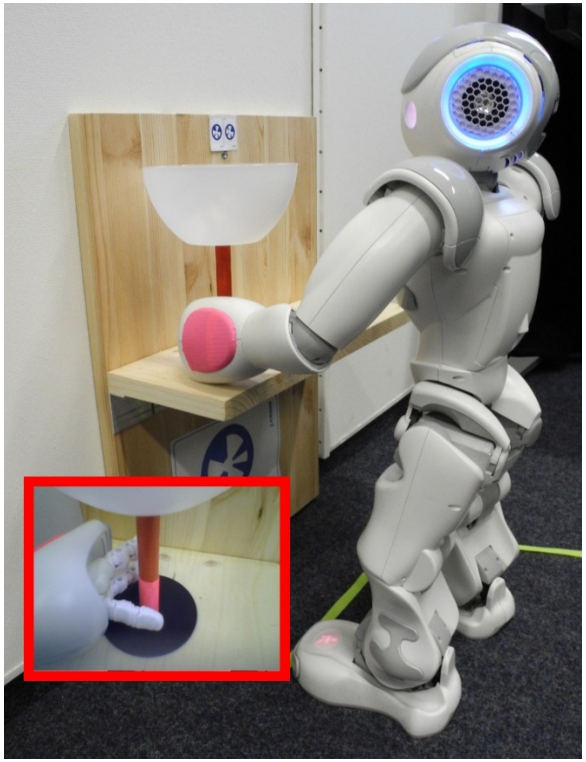
# Landmark approaching on a PeopleBot

# Landmark approaching on a Sony Aibo

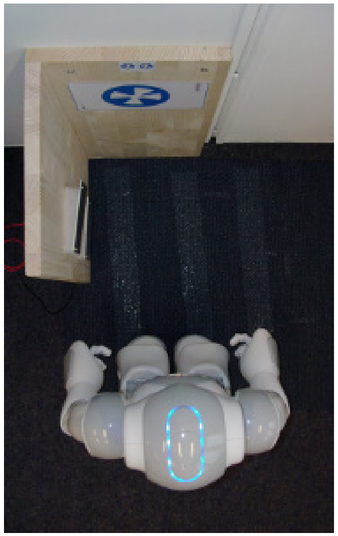


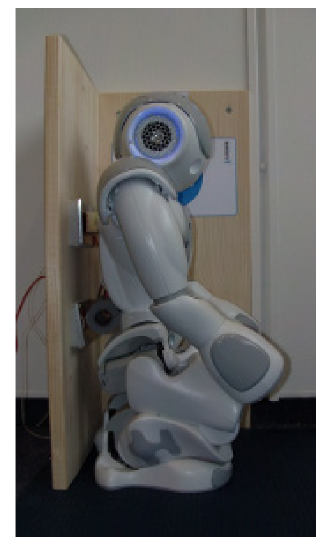
# Real-world reinforcement learning for autonomous humanoid robot docking (Nicolás Navarro-Guerrero)

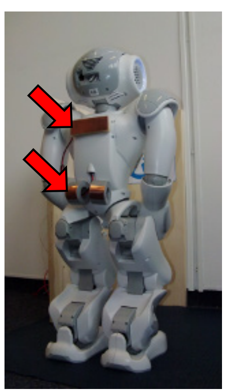




# Hybrid approaching for charging (1)

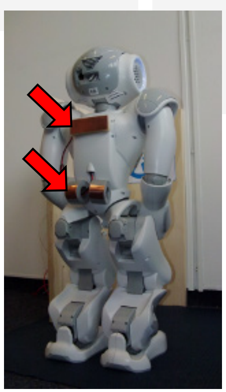






# Hybrid approaching for charging (2)

- Phase I – Hard-coded algorithm: search and approach landmarks. Placed the robot 40 cm away from landmarks



- Phase II – Hard-coded algorithm: Places the robot (approx.) parallel to the wall looking at the landmarks

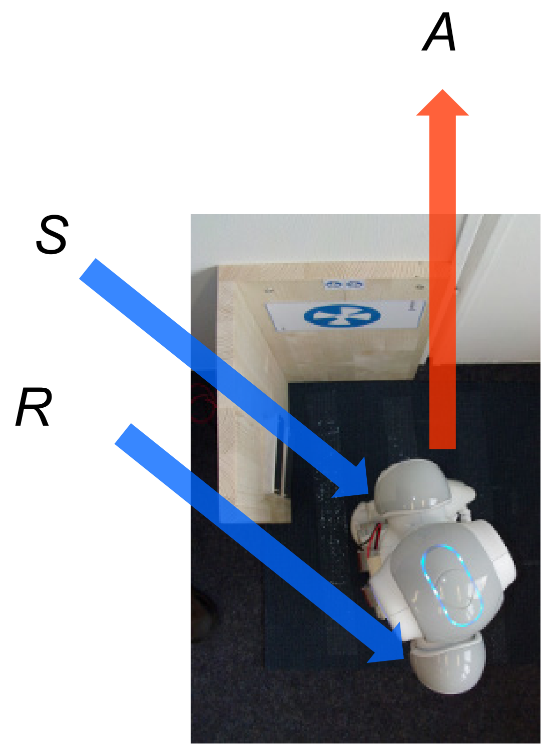
- Phase III – Neural Docking: Reinforcement learning (SARSA) algorithm. After learning the robot senses position and orientation and manoeuvers towards goal

- Phase IV – Hard-coded algorithm: check sensors. If false positive is detected, correct pose or go to Phase III. Else move the robot to a crouch pose

# Agent-environment interaction

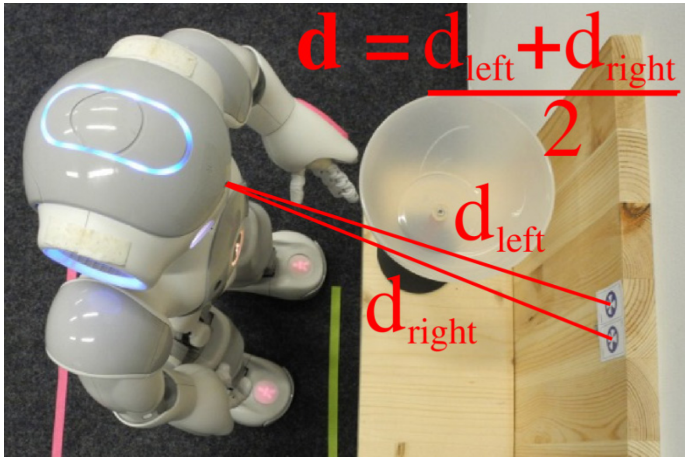
Markov Decision Process (MDP)

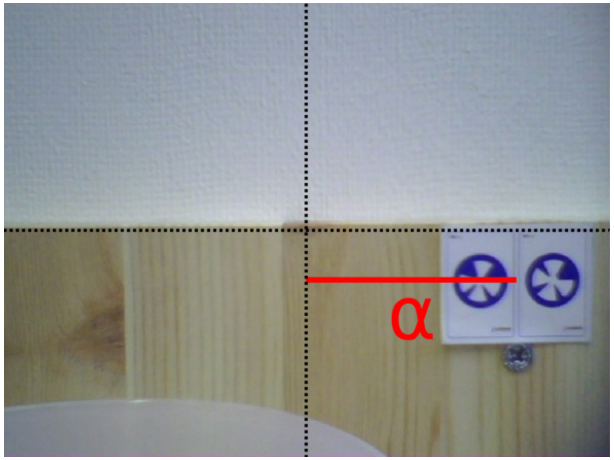
- fixed transition probabilities
- next move not depending on history
- fixed reward probability

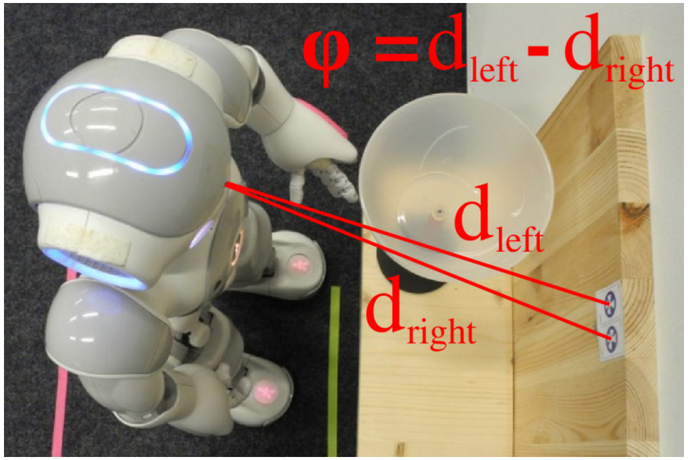



- sense state **S**
- select and perform an action **A**
- occasionally, receive reward **R**
- sense state **S'**
- select an perform an action **A'**

# Example of state space definition for forward docking



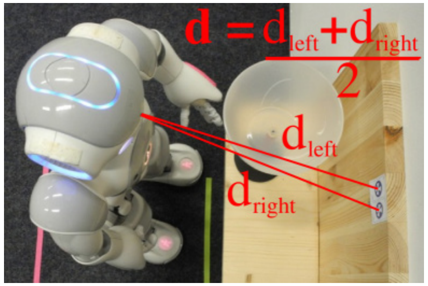



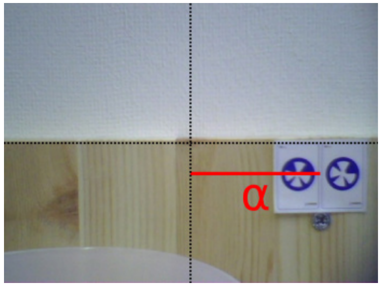



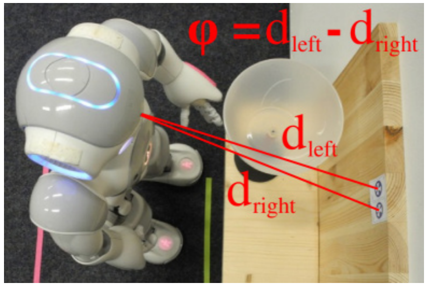

A set of measurable variables to form a finite and discrete state space.

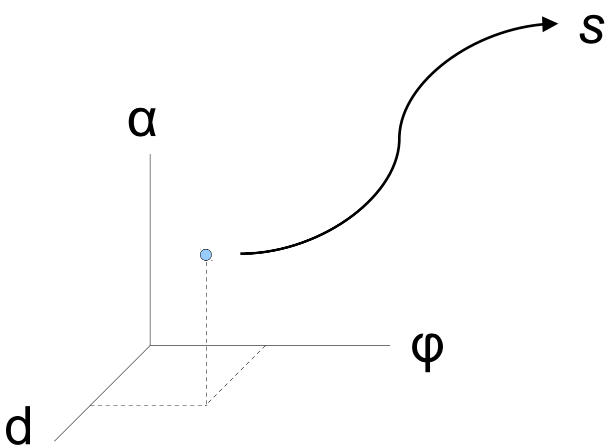
# Reinforcement learning – SARSA (1)

(1) Check where the agent is, i.e. determine active state *s* and check whether any feedback *r* has been received (reward or punishment)



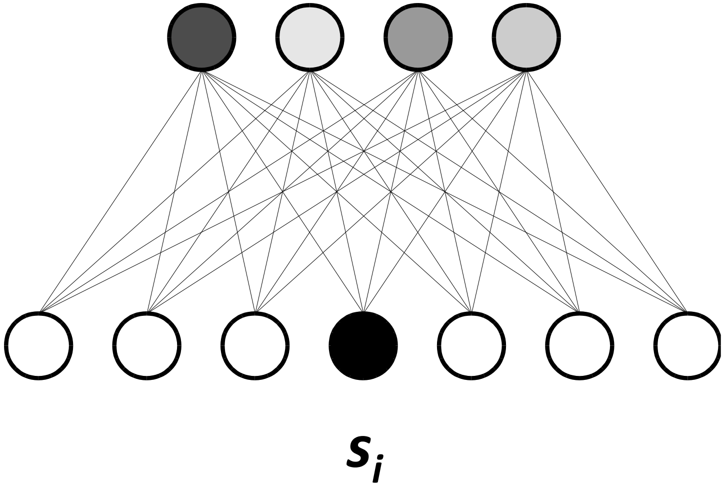




The selected variables are encoded into a unique state value. Other algorithms can directly use the measured variables.

# Reinforcement learning – SARSA (2)

(2) Compute action strength

$$h_i = \sum_l W_{il} s_l$$

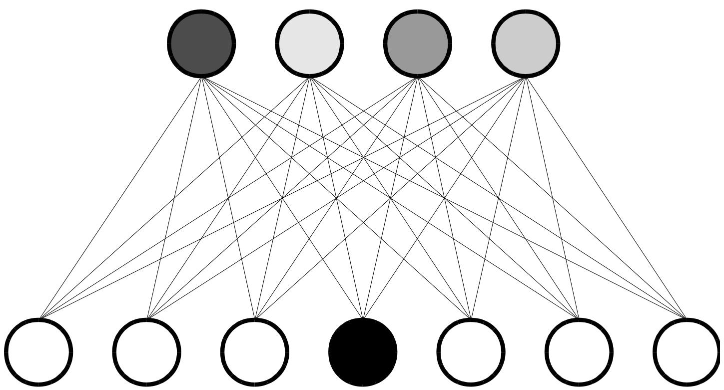



With the knowledge acquired so far the network's output indicates which action leads to higher reward (or lower punishment).

# Reinforcement learning – SARSA (3)

(2) Compute action strength

$$h_i = \sum_l W_{il} s_l$$



$s_i$

(3) Select action (soft-max)

$$P_{(a_i=1)} = \frac{e^{\beta h_i}}{\sum_k e^{\beta h_k}}$$

$a$

Many action selection strategies exist:

- winner-takes-all
- soft-max
- random
- …

# Reinforcement learning – SARSA (4)

(4) Current estimate (action-value function)

$$Q_{(s,a)} = \sum_{k,l} W_{kl} a_k s_l$$

(5) Prediction error

$$\delta = \begin{cases} \gamma Q(s', a') - Q(s, a), & \text{if } r = 0, \\ r - Q(s, a), & \text{if } r = 1, \end{cases}$$

When the executed action places the agent in the terminal state or a state known to lead to the terminal state, the network is updated towards this action

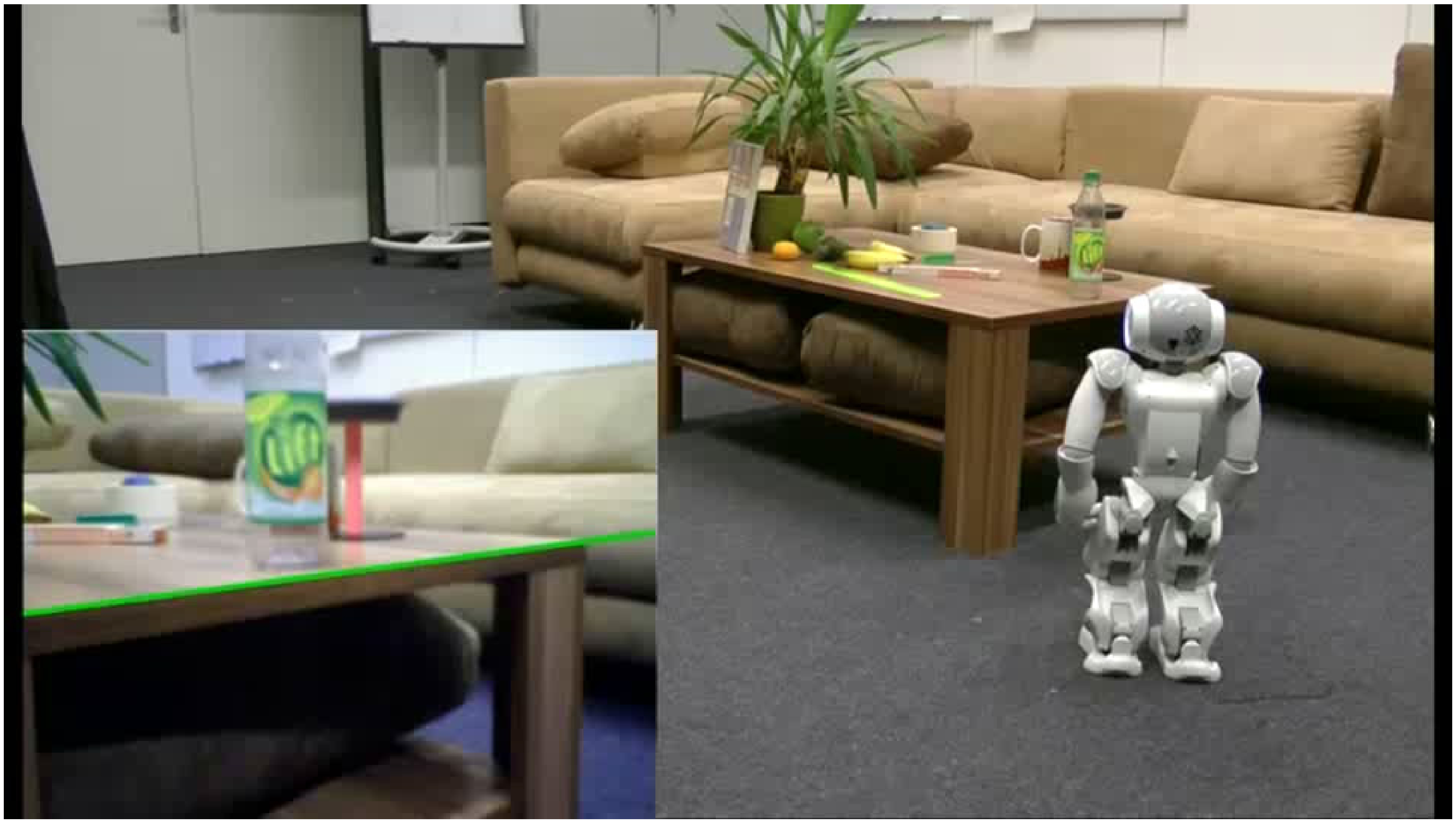
(6) Weight update

$$\Delta W_{ij} = \epsilon \delta a_i s_j$$

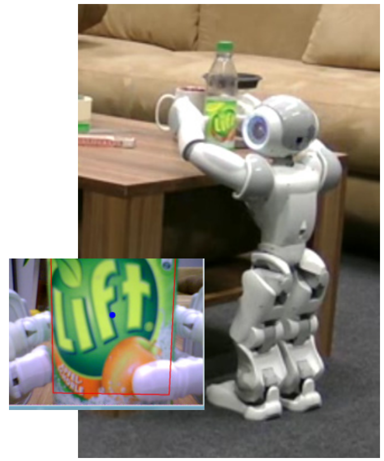
Weights are updated and a new learning step can be started.

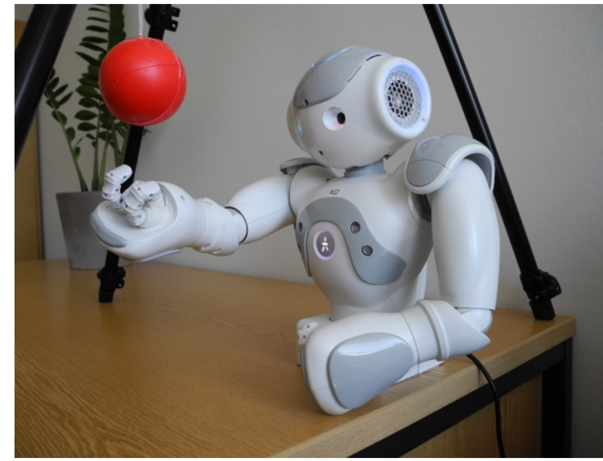
# Docking and recharging

# Forward docking and grasping



# Reinforcement learning and punishment



- Objectives:
  - Learn arm’s inverse kinematics with RL (only one arm)
  - Study the effect of pain-like signals on robot learning
    - Does improve learning speed?
    - What is a good dynamical model of joint pain?
    - How much pain decrease performance?

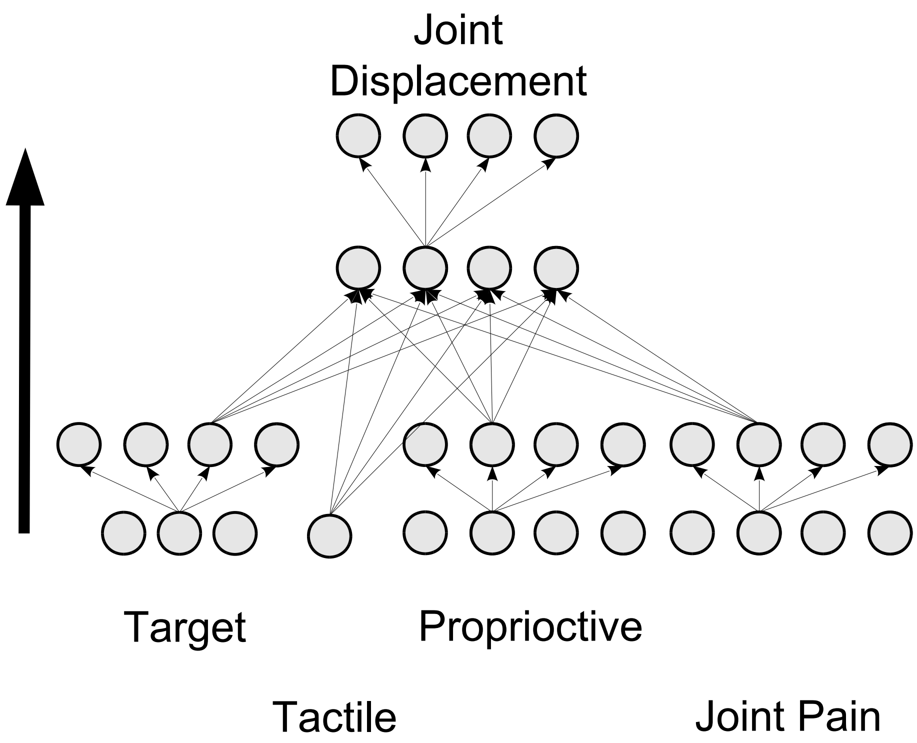
# Reinforcement learning - Continuous State and Action Spaces (CACLA)

**Algorithm 7** Cacla

Given $\gamma$, an initial state distribution $I$ and an MDP to act on.<br>
Initialize $\vec{\theta}$, $\vec{\psi}$, $s \sim I$.<br>
**repeat**<br>
    Choose $a \sim \pi(s, \vec{\psi})$<br>
    Perform $a$, observe $r$ and $s'$<br>
    $\delta = r + \gamma V(s') - V(s)$<br>
    $\vec{\theta}^T = \vec{\theta}^T + \beta\delta\nabla_\theta V(s)$<br>
    **if** $\delta > 0$ **then**<br>
        $\vec{\psi}^T = \vec{\psi}^T + \alpha(a - Ac(s, \vec{\psi}))\nabla_\psi Ac(s, \vec{\psi})$<br>
    **end if**<br>
    **if** $s'$ is terminal **then**<br>
        $s \sim I$<br>
    **else**<br>
        $s = s'$<br>
    **end if**<br>
**until** end

H. van Hasselt and M. A. Wiering, (2007) "Reinforcement learning in continuous action spaces," in Proceedings of the IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL). IEEE. pp. 272-279.

# Neural architecture




- 3D Target coordinates respect to the arm's base (shoulder)
- A binary input signal activated when the hand and ball are in contact
- Proprioceptive input: angular position of every joint
- Joint Pain: exponential pain signal

# Experimental setup

- Reinforcement
  - Reward proportional to the distance from hand to ball
  - Bonus reward when the ball is in contact with the ball
  - Punishment when joint position is in maximal position
- Open Questions and Extensions:
  - Learn to stop when target is reached
  - Are both pain and punishment necessary?
  - What would be the best dynamical curve for joint pain?
  - Include other embodied pain sources, e.g. joints' velocity, torque, temperature
  - Collision detection

# Summary and reading

- Symbolic Strips like planners versus RL planners:
  Most promising hybrid planners: hierarchical RL, interactive RL, probabilistic Strips etc...


- A Model of Hippocampally Dependent Navigation, Using the Temporal Difference Learning Rule, D.J. Foster, R.G.M. Morris, and P. Dayan, *HIPPOCAMPUS 10:1–16 (2000)*

- Muse, D., Wermter, S. Actor-Critic Learning for Platform-Independent Robot Navigation. Cognitive Computation, Volume 1, Springer New York, pp. 203-220, 2009