

# Grasping with Flexible Viewing-Direction with a Learned Coordinate Transformation Network\*

Cornelius Weber, Konstantinos Karantzis and Stefan Wermter

*Hybrid Intelligent Systems, School of Computing and Technology*

*University of Sunderland, Sunderland, SR60DD, UK*

*Web: [www.his.sunderland.ac.uk](http://www.his.sunderland.ac.uk)*

*{[cornelius.weber](mailto:cornelius.weber@sunderland.ac.uk), [stefan.wermter](mailto:stefan.wermter@sunderland.ac.uk)}@sunderland.ac.uk, [konstantinos.karantzis@gmail.com](mailto:konstantinos.karantzis@gmail.com)*

**Abstract**— We present a neurally implemented control system where a robot grasps an object while being guided by the visually perceived position of the object. The system consists of three parts operating in a series: (i) A simplified visual system with a what-where pathway localizes the target object in the visual field. (ii) A coordinate transformation network considers the visually perceived object position and the camera pan-tilt angle to compute the target position in a body-centered frame of reference, as needed for motor action. (iii) This body-centered position is then used by a reinforcement-trained network which docks the robot at a table so that it can grasp the object. The novel coordinate transformation network which we describe in detail here allows for a complicated body geometry in which an agent’s sensors such as a camera can be moved with respect to the body, just like the human head and eyes can. The network is trained, allowing a wide range of transformations that need not be implemented by geometrical calculations.

**Index Terms**—Neural Networks, Frame of Reference Transformations

## I. INTRODUCTION

The control of the human body is a complex task due to the complexity of its geometry and the difficulty to extract information from the world by sensors like vision and to transform it into a motor-relevant representation. For the simple task of grasping an object, for example, we need to (i) visually localize an object, (ii) infer its position in body-centered coordinates which are relevant for control of the arm and hand and (iii) activate the relevant muscles to perform the grasping. Here we present a neural model which consists of three systems, (i) a visual, (ii) a coordinate transform and (iii) a motor system, which performs such a task on a simulated PeopleBot robot. The complexity of the human body is addressed by the anthropomorphic robot’s camera which can pan-tilt its gaze direction during the task. This parallels eyes and/or the head moving with respect to the body, which makes it necessary to transform a visually identified location into a body-centered location as is relevant for the motor control.

In [14] we have implemented a visually controlled robotic grasping (docking) action that was learned by reinforcement

learning. The assumption that the robot camera was fixed to the body allowed a direct match from pixel coordinates to body-centered coordinates. Because of the fixed camera, objects had to be in a confined space so that they were visible. Such a grasping maneuver is demonstrated at the following URL: [www.his.sunderland.ac.uk/robotimages/Cap0001.mpg](http://www.his.sunderland.ac.uk/robotimages/Cap0001.mpg). In [9] the camera is allowed to move, but it is assumed that it is already fixating the target when computing the direction of the object. Hence a reaching map can be defined using only the camera posture. When a robot or the object is moving, however, a camera can hardly fixate the object, in particular when using commercially available, slow pan-tilt camera mounts.

### A. Coordinate Transformations

The robotic problem of coordinate systems has its example in humans. Our sensory system provides us with information about objects of interest in sensory frames of references (coordinate systems) which are not directly applicable to the motor system. For example, a bright object will activate certain photo-receptors on the retina, but this information cannot be directly used to control the joint angles of an arm to reach the object. Additional information such as the eye- or head position needs to be integrated. In humanoid robots such coordinate transformations also need to be accounted for, specifically to allow movement of their pan-tilt camera.

1) *Geometry*: For simplicity we will in the following assume that eye- and head position are not distinguished, which accounts for a pivoting camera mounted on a robot body. Fig. 1 visualizes the geometry of our setup with a simple robot, a PeopleBot. It shares with humanoid robots its pan-tilt-able camera, while an arm reaching movement is substituted for by a whole body movement of the PeopleBot.

If the robot is to grasp the fruit object on the table, then the following coordinates are important for controlling the motors: the distance  $d$  of the robot to the target object and the angle  $\theta$  at which the object is to the left or the right of the robot body. The angle  $\varphi$  which defines the robot rotation with respect to the table border will later be used for the motor coordination, but is not important for now.

\*This work is part of the MirrorBot project supported by EU Grant IST-2001-35282

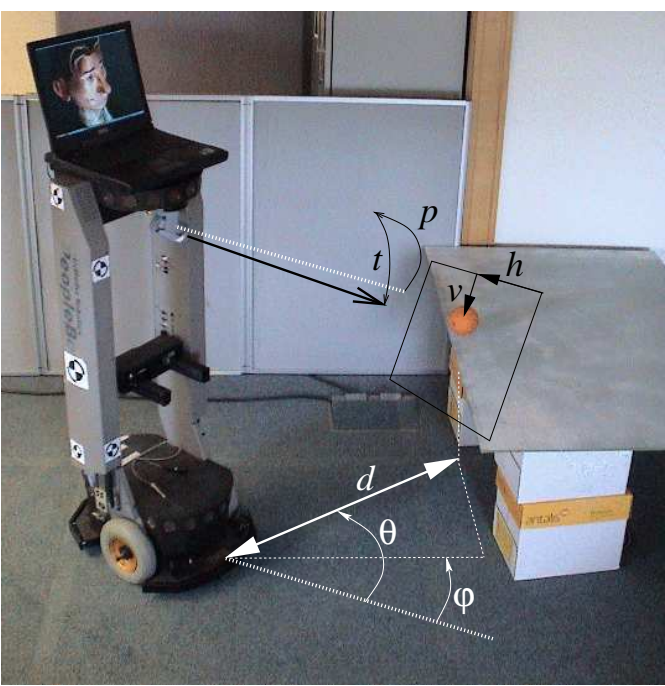


Fig. 1. The PeopleBot robot in its environment. The relevant motor coordinates (in a body centered frame of reference) are the distance  $d$  between the robot and the target, and the angle  $\theta$  of the target away from the forward direction (white dotted lines) of the robot. Known to the robot are the pan angle  $p$  and tilt angle  $t$  of the camera (which points into the direction of the long black arrow) and the horizontal and vertical image pixel coordinates  $h$  and  $v$  of the perceived target object (the black frame indicates the image seen by the robot camera). The values  $d, \theta$  have to be computed from  $h, v$  and  $p, t$  via the coordinate transformation network. The motor network will later receive additional information  $\varphi$ , which is the robot orientation angle w.r.t. the axis perpendicular to the table edge.

While  $d$  and  $\theta$  are important for motor control, the robot sensory system represents the object only visually, delivering the perceived horizontal and vertical positions  $h$  and  $v$  of the object in the visual field. Knowing also the camera pan- and tilt angles  $p$  and  $t$ , it is possible to compute  $d$  and  $\theta$ . An assumption we make is a constant elevation of the camera over the target object which allows that the distance of the object can somehow be estimated from how low it is perceived. This compensates for not using a stereo camera.

In summary,  $(d, \theta)$  are a function of  $(h, v, p, t)$  which the coordinate transformation network has to learn. It would be possible, even with complicated human geometries, to compute this function using deterministic vectorial transformations. Humans, however, learn this function allowing for adaptations during evolution and ontogenesis.

2) *Biology*: In the mammalian cortex transitions are made between visual representations and motor representations in the posterior parietal cortex (PPC). It is situated at a strategic position between the visual cortex and the sensory cortex and consists mainly of Brodmann's areas 5 and 7. PPC neurons

are modulated by the direction of hand movement, as well as by visual object position, eye position and limb position signals [2]. These multi-modal responses allow the PPC to carry out computations which transform the location of targets from one frame of reference to another [3], [4].

3) *Previous Models*: Models of neural coordinate transformations originally dealt with “static” sensory-motor mappings, in which, for example, Cartesian coordinates  $(x_1, x_2)$  of an object (e.g. as seen on the retina) are neurally transformed into joint angles  $(\theta_1, \theta_2)$  of an arm required to reach the target [7]. However, this model is static in the sense that it does not account for the influence of another variable, such as the rotation of the head. In order to account for such a modulating influence we need dynamic, adjustable mappings.

A standard way to achieve such a mapping is to feed two inputs into the hidden layer such as Cartesian coordinates  $c$  and head rotation  $r$ . These inputs use population codes  $x^c$  and  $x^r$  where the location of an approximately Gaussian-shaped activation hill encodes the value. Both inputs are used in a symmetric way. The working principle of the use of the hidden layer is described for the case of one-dimensional input layers as [11]: “One first creates a 2-dimensional layer with an activity equal to the [outer] product of the population activity in  $x^c$  and  $x^r$ . Next, a projection from this layer to an output layer implements the output function  $z = f(x^c, x^r)$ .”

Such a network with two one-dimensional input layers, a one-dimensional output layer and a two-dimensional hidden layer has been termed a basis function network [5]. Because of its structure, the output layer is symmetric with the input layers, so the network can be used in any direction. Lateral weights within each layer allow for a template fitting procedure during which attractor network activations generate approximately Gaussian-shaped hills of activations. In a “cue integration” mode the network can receive noisy input at all three visible layers in which case it will produce the correct, consistent hills with maximum likelihood.

The gain field architecture [10] adds a second hidden layer which subtracts certain inputs to remove unwanted terms from the solution on the first hidden layer. This allows it to encode the position of a hill of activation and also its amplitude. Since this breaks the symmetry between the input layers and the output layer, the network is used only in one direction.

The use of the hidden layer as the outer product of input layers has the advantage that the hidden code or weights can easily be constructed using algebraic transformations. Therefore a learning algorithm for all the weights is not given with these models. A specific disadvantage is the large dimensionality of the hidden layer: if both input layers are two-dimensional, then the hidden layer would have to be represented as a four-dimensional hyper-cube.

4) *Proposed Model*: Here we propose a network which learns the transformation. Every unit is connected with all other units by connection weights which are trained by an

TABLE I

THE SIMULATED AREAS AND THEIR SIZES.

Area	Size	Comments
Visual network		
retina	$3 \times 48 \times 32$	red, green, blue layers
what	$64 \times 48$	connected only with "what"
where (retinal)	$48 \times 32$	
Coordinate transformation network		
where (retinal)	$24 \times 16$	connected only with "eye-pos" and "where (body)"
eye-pos	$20 \times 20$	
where (body)	$20 \times 20$	
Action network		
eye-mov	2	pan, tilt
state	$20 \times 20 \times 20$	
critic	1	forward, back, turn left, right
actor	4	

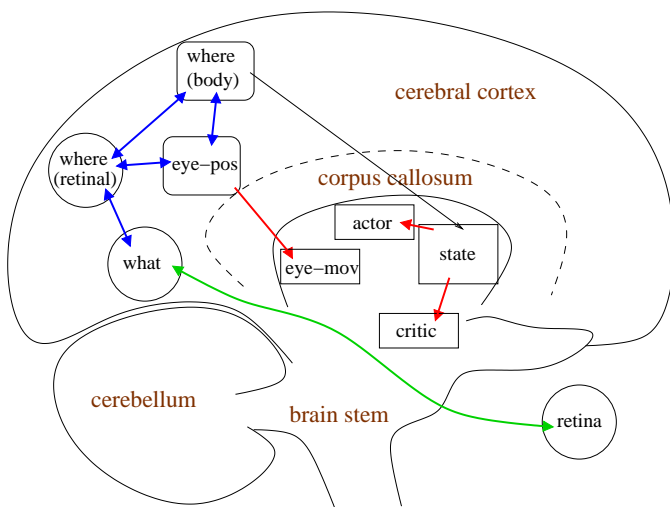


Fig. 2. The full model overlaid on a sketch of the brain. The boxes denote the implemented areas, or layers, of neurons; round if in the visual system, oval-boxes if in the coordinate transform system and rectangular if in the motor system. Thick arrows represent trained connections, partially in both connections as the arrow heads suggest, a thin arrow denotes mere copying. Green arrow color denotes self-organized learning as neural activations on one of the sides was not given during training. Blue denotes associative learning where all activations were given for training. Red arrows denote error-driven or reinforcement learning of connections.

associative learning rule [1]. The learning rule is biologically plausible using only local, Hebbian and anti-Hebbian learning, and furthermore allows to include any number of additional hidden units in order to make the network more powerful in generating a complex data distribution. The hidden code would self-organize during learning without requiring the network designer to construct it.

During learning, all three areas receive their respective coordinate as training data in a symmetric fashion, while after learning, missing information in any area can be recovered based on the principle of pattern completion. Using two-dimensional input areas coding a visually perceived object position and the camera pan-tilt position, we show that this auto-associative attractor network is able to generate the corresponding body-centered object position which is required for the robot motor control. This is done without an additional hidden layer, thereby significantly reducing the network complexity w.r.t. the previous methods.

## II. METHODS AND RESULTS

The overall system is shown in Fig. 2, suggestively overlaid on a sketch of the brain. It is a starting point to implement a fully neurally controlled agent, consisting of a visual system to localize the target object, a coordinate transform system accounting for the complexity of the agent's body and a motor system for action execution.

The model not only covers a small portion of cortical areas, but also the simulated areas would cover only a small part of

each area (e.g. approximately four hypercolumns in the visual cortex), thus, areas are displayed too large in Fig. 2. The sizes of the simulated areas are given in Table I.

### A. Visual Network

The visual system consists of three areas. The "retina" receives the color image of the robot camera, a "what" area extracts features from the image and based on the activated features, a "where" area localizes the object of interest. Its size matches the size of the "retina" (see Table I), and a hill of neural activation represents the perceived object location at the corresponding "retinal" location. Fig. 3 shows an example of how these areas are being activated.

To refer to biology, the "retina" area rather represents the LGN and the "what" area represents the primary visual cortex V1, since unsupervised learned recurrent connections between these areas establish edge detector-like neuron responses on the "what" area [12]. The "where" area received the object location during learning, so was effectively trained in a supervised manner. Details are described in [13].

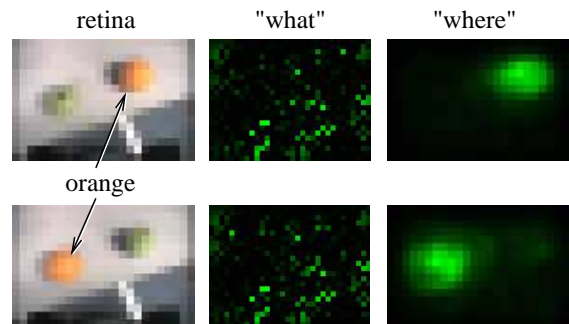


Fig. 3. The visual network localizing an orange object correctly in two situations in the presence of a distractor apple object. Active units in the "what" and "where" areas are depicted in lighter color.

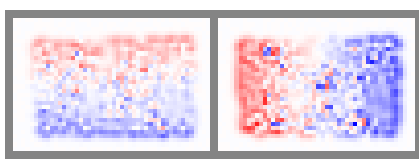


Fig. 4. The connection weights from the visual “where” area to the two motor units controlling tilt (left) and pan (right) of the camera. Red indicates negative connections which cause a tilt upward (and the object to be perceived more downward) or a pan to the left. Blue indicates positive weights, causing movement into the opposite directions.

*Eye Movement:* In order to keep the target object constantly in view, two neural units controlled the camera to make a “saccade” toward the target object after every evaluation of the image. These have been trained based on their error after causing a “saccade”. This error was the distance in vertical direction  $\delta^v$  and in horizontal direction  $\delta^h$  of the perceived target object after movement to the middle of the retina. First, the units’ activations  $a^{v,h}$  were computed from the “where” unit activations  $\bar{a}^w$  according to:

$$a^{v,h} = \sum_i w_i^{v,h} a_i^w.$$

A camera movement was made in proportion to these activations, then the error was measured and the weights were changed according to

$$\Delta w_i^{v,h} \approx \delta^{v,h} a_i^w.$$

This scheme is simple and efficient rather than biologically accurate. Resulting weights are shown in Fig. 4. Irregularities originate from some mis-localizations of the “what”-“where” network in small regions during training.

## B. Coordinate Transformation Network

1) *Encoding of the Data:* The coordinate transformation network has as inputs the visually perceived target object position and the camera position, and as output the target object position in a body-centered frame of reference. All positions are coded as a neural activation code, as shown in Fig. 5. That is, each two-dimensional position is encoded on a two-dimensional sheet of neurons. The neurons maintain an activity pattern that has the shape of one localized hill, and the position of this hill encodes the two-dimensional vector.

a) *Visual “Where” Area:* The hill of activation coding for the visually perceived object location (visual “where” area) is taken from the visual system. However, in order to allow all areas of the coordinate transformation network to have comparable sizes, the visual system’s “where” area was down-scaled to half the size for use here (see Table I). This may alternatively be regarded as allowing only units on every second row and column to make any connections with the other two areas of the coordinate transformation network.

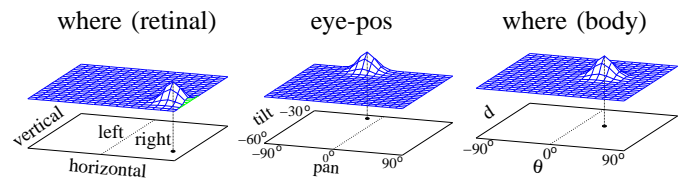


Fig. 5. The three areas involved in the coordinate transformation (cf. Fig. 2). The lower part of each figure denotes the space encoded by the neurons in the upper part where active neurons are displayed as hills. A hill of neural activation carries the information about the corresponding position of each vector. The visual “where” area, left, represents the object position as perceived on the retina; the eye-position area, middle, codes the pan- and tilt values of the robot camera; the body-centered “where” area, right, codes the object position in body centered coordinates: this is encoded by the square root of the distance  $d$ , and the body centered angle of the target  $\theta$  (cf. Fig. 1).

b) *Eye-Position Area:* The “eye-pos” area encodes the robot camera pan- and tilt position which is directly readable from the camera unit interface.

c) *Body-Centered “Where” Area:* The body-centered “where” area encodes the object position in a body-centered frame of reference, which is directly usable for the motor control. There are different ways to parameterize this position, such as using Cartesian coordinates w.r.t. the main, forward axis of the robot. We chose instead as one coordinate the angle  $\theta$  of the object away from the main axis and as the second coordinate the square root  $\sqrt{d}$  of the distance  $d$  between the robot and the object. The angle is naturally important for how the robot must turn, while the distance is particularly important at close ranges, because the robot can bump with its “shoulders” into the table if it turns near the table. The square root function extends near ranges, thus more neurons will be devoted to more critical areas for the grasping action.

2) *Generation of Data:* Instead of theoretically computing the body-centered position from the visual and pan-tilt positions, we use the gazebo robot simulator to generate this data. After training, the neural network can compute the body-centered position of the target.

The data was collected by moving the simulated PeopleBot robot within the area which would be covered later for the docking. This was necessary, because the PeopleBot model in the gazebo simulator does not allow it to be randomly placed at any given position (which would have simplified data collection). The simulated PeopleBot was started at the goal-position and then moved into random positions in the field by a random series of: turn – backward movement – turn – backward or forward movement – turn. Following this, the robot moved approximately to the goal by its learned reinforcement strategy (see next chapter), using absolute position information that can be retrieved by the simulator. Thus the coordinate transformation network was circumvented for data collection. Also the visual network was replaced by a color blob-finding algorithm in order to get a more precise estimate of the position and to be able to sort out invalid data where

the target object was not seen in the visual field.

At the end of a movement sequence, a “reset” flag caused the simulation to restart from the beginning, i.e. the exact goal-position, and another sequence of data sampling was done. Since the distances traveled by the robot are influenced by the load of the computer processor, movement was done in a speed approximating real-time and was not accelerated. The data was collected as a file, and learning of the coordinate transformation network was done off-line with a randomized order in which the data was shown.

Inspection of the data showed inconsistencies, in that given visual and pan-tilt coordinates  $(h, v, p, t)$  were not always paired with the same body-centered coordinates  $(d, \theta)$ . We have two explanations for these. First, the data were generally taken while the robot and pan-tilt camera were moving, thus, reading the sensor and odometry values one after another, with them possibly being buffered by the simulator, means that the values originate from differing times. This will cause relatively small mismatches. Secondly, we have found large mismatches at small distances  $d \approx 0$  where at different values of  $\theta$  the perceived input values  $(h, v, p, t)$  can be the same.

3) *Architecture*: The coordinate transformation network consists of three fully connected areas which represent the respective coordinates as neural population codes, as visualized in Fig. 5. The three areas together constitute one attractor network, so if information is absent on one area, then it will be filled-in by a process of pattern completion. By allowing a full connectivity and by not specifying the direction of information flow (what is input and what is output), we set only very general constraints that are compatible with associative information processing in the cortex.

4) *Training*: The interrelations of the coordinates on the three areas in terms of statistical correlations are learned by the Boltzmann machine framework. With its binary stochastic units it is powerful in learning a given data distribution. The learning rule uses randomized unit update rather than structured information flow, so it lends itself to highly interconnected networks and allows to introduce additional hidden units if performance needs to be enhanced.

Let us term the whole network activation vector  $\mathbf{x} = (\mathbf{x}^{retinal}, \mathbf{x}^{eye-pos}, \mathbf{x}^{body})$ , consisting of the concatenated activations of all areas. The Boltzmann machine learning rule distinguishes two running modes: in the *clamped phase* the data distribution is forced upon the visible units of the network (in our case, all units are visible). This distribution is termed  $P_{\mathbf{x}}^+$  with the upper index “+” denoting the *clamped phase*. The other running mode is the *free running phase* in which the distribution  $P_{\mathbf{x}}^-$  over the network states arises from the stochasticity of the units and is determined by the network parameters, such as weights and thresholds. The upper index “-” denotes the *free running phase*.

The goal of learning is that the distribution  $P_{\mathbf{x}}^-$  generated by the network approximates the data driven distribution  $P_{\mathbf{x}}^+$

which is given.  $P_{\mathbf{x}}^- \approx e^{-E(\mathbf{x})}$  is a Boltzmann distribution which depends on the network energy  $E(\mathbf{x}) = \sum_{i,j} w_{ij} x_i x_j$  where  $w_{ij}$  denotes the connection weight from neuron  $j$  to neuron  $i$ . Therefore  $P_{\mathbf{x}}^-$  can be molded by training the network parameters. Derivation of the Kullback-Leibler divergence between  $P_{\mathbf{x}}^-$  and  $P_{\mathbf{x}}^+$  w.r.t. the network parameters leads to the learning rule (see e.g. [8]):

$$\Delta w_{ij} = \epsilon \left( \sum_{\{\mathbf{x}\}} P_{\mathbf{x}}^+ x_i x_j - \sum_{\{\mathbf{x}\}} P_{\mathbf{x}}^- x_i x_j \right) \quad (1)$$

with learning step size  $\epsilon = 0.001$ . Computing the left term corresponds to the *clamped phase* of the learning procedure, the right term to the *free running phase*. Without hidden units, the left term in Eq. (1) can be re-written as  $\sum_{\mu}^{data} x_i^{\mu} x_j^{\mu}$  where  $\mu$  is the index of a data point. Without hidden units thus the *clamped phase* does not involve relaxation.

The right term of Eq. (1) can be approximated by sampling from the Boltzmann distribution. This is done by recurrent relaxation of the network in the *free running phase*. The stochastic transfer function

$$P(x_i(t+1)=1) = \frac{1}{1 + e^{-\sum_j w_{ij} x_j(t)}} \quad (2)$$

computes the binary output  $x_i \in \{0, 1\}$  of neuron  $i$  at time step  $t + 1$ . Repeated relaxation approximates a Boltzmann distribution of the activation states.

During training, the two phases are computed alternately. One randomly chosen data point (similar to Fig. 5) accounts for the *clamped phase*. Its continuous values are allowed in Eq. 2. Then a relatively short relaxation of the network, consisting of updating all units for 15 iterations using Eq. (2) accounts for the *free running phase*. Units are initialized in this phase by activating every unit the first iteration with a probability of 0.1, regardless of its position.

Self-connections were omitted and a threshold  $\theta_i$  was added to each unit. This is treated as a weight that is connected to an external unit with a constant activation of  $-1$ .

5) *Performance*: We initialize the network activations with a hill of activation on each of the visual “where” area and the eye-position area, taken from the training data. After initializing the body-centered “where” area with zero activation, we are interested in whether the network dynamics according to Eq. 2 produce a hill of activation at the correct location. Fig. 6 shows the average deviations between the correct location and the generated hill. The average distance is approximately one unit, with some outliers at a few positions.

### C. Action Network

The state space is made from the representation in the body-centered “where” area along two dimensions and a representation of the robot rotation angle  $\varphi$  along the third dimension. A narrow Gaussian represents a position in this state space, as visualized in Fig. 7 a).

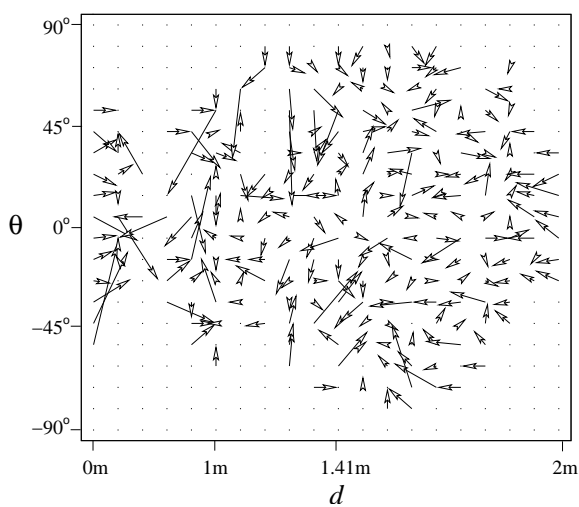


Fig. 6. Errors by the coordinate transformation network on the body-centered “where” area. Arrows show from the correct position in space of  $(\theta, \sqrt{d})$  to the position predicted by the network. An average over 2000 data samples is taken. Note that not the entire space was covered by the robot movements.

In [14] we have described the actor-critic reinforcement learning algorithm [6] applied to this docking action in a close range to the target with fixed camera and thus without involving the coordinate transformation network. A video of the robot performing this docking action can be seen at: <http://www.his.sunderland.ac.uk/robotimages/Cap0001.mpg>.

The result of learning is a complex weight structure from the state space to the critic and to the four motor units (actor), as shown in Fig. 7 b) and c), respectively.

### III. DISCUSSION

The model is shown controlling the PeopleBot robot in the gazebo simulator at the following web address: <http://www.his.sunderland.ac.uk/supplements/humanoids05/>

In summary, we have presented a neural network which controls a visually guided grasping / robotic docking action while allowing movement of its camera w.r.t. its body. It integrates essential processing steps of the brain, such as vision, coordinate transformations and movement control.

As the first model known to us that *learns* such a dynamic coordinate transformation, it learns in a supervised fashion in that the body-centered “where” location was retrieved from the simulator. Next, we will address its self-organization from the inputs. This would be in particular useful as the data from the simulator were erroneous when retrieved during robot and camera pan-tilt movement, while self-organized data from vision and pan-tilt readings are expected to be consistent. Since the reinforcement learning scheme does not require a specific organization of the data within the state space, it allows for such self-organization and also *grounding* the robot behavior in the real world. Our fully learned network is a basis for this, and is as well directly applicable to humanoid shaped robots.

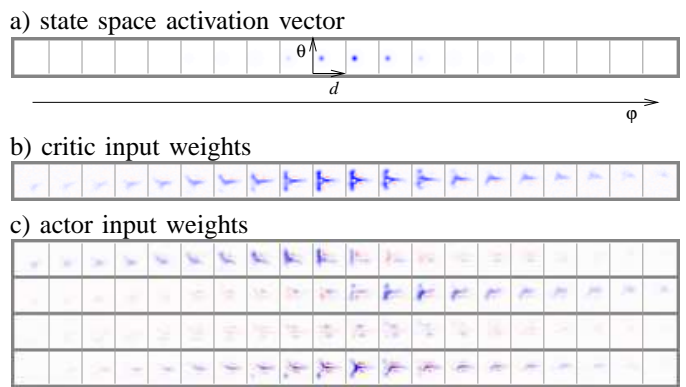


Fig. 7. Representations in the state space. **a)** shows the coordinates of the state space. Each small rectangle represents the body-centered “where” space (cf. Figs. 5 and 6). The robot rotation angle  $\varphi$  is then represented by multiplexing the “where” space 20-fold. It ranges from  $-90^\circ$ , left to  $90^\circ$ , right. The blobs that can be seen represent an example activation vector which has the shape of a Gaussian in this  $20 \times 20 \times 20$ -dimensional cube. **b)** shows the critic unit’s weights from the state space after learning. Large weights, i.e. high fitness values, are assigned near the target, i.e. at small distances  $d$  and around zero values of the angles  $\theta$  and  $\varphi$ . **c)** shows weights (blue denotes positive, red negative weights) from the state space to the actor which consists of the four motor units (from top to bottom: right, left, back, forward).

### REFERENCES

- [1] D. Ackley, G. Hinton, and T. Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive Science*, 9:147–69, 1985.
- [2] C.A. Buneo, M.R. Jarvis, A.P. Batista, and R.A. Andersen. Direct visuomotor transformations for reaching. *Nature*, 416:632–6, 2002.
- [3] Y.E. Cohen and R.A. Andersen. A common reference frame for movement plans in the posterior parietal cortex. *Nature Review Neuroscience*, 3:553–62, 2002.
- [4] J.D. Crawford, W.P. Medendorp, and J.J. Marotta. Spatial transformations for eye-hand coordination. *J. Neurophysiol.*, 92:10–9, 2004.
- [5] S. Deneve, P.E. Latham, and A. Pouget. Efficient computation and cue integration with noisy population codes. *Nature Neurosci.*, 4(8):826–31, 2001.
- [6] D.J. Foster, R.G.M. Morris, and P. Dayan. A model of hippocampally dependent navigation, using the temporal difference learning rule. *Hippocampus*, 10:1–16, 2000.
- [7] Z. Ghahramani, D.M. Wolpert, and M.I. Jordan. Generalization to local remappings of the visuomotor coordinate transformation. *J. Neurosci.*, 16(21):7085–96, 1996.
- [8] S. Haykin. *Neural Networks. A Comprehensive Foundation*. MacMillan College Publishing Company, 1994.
- [9] L. Natale, G. Metta, and G. Sandini. A developmental approach to grasping. In *Developmental Robotics AAAI Spring Symposium*, 2005.
- [10] E. Sauser and A. Billard. Three dimensional frames of references transformations using recurrent populations of neurons. *Neurocomputing*, 64:5–24, 2005.
- [11] A. van Rossum and A. Renart. Computation with populations codes in layered networks of integrate-and-fire neurons. *Neurocomputing*, 58-60:265–70, 2004.
- [12] C. Weber. Self-organization of orientation maps, lateral connections, and dynamic receptive fields in the primary visual cortex. In G. Dorffner, H. Bischof, and K. Hornik, editors, *Proc. ICANN*, pages 1147–52. Springer-Verlag Berlin Heidelberg, 2001.
- [13] C. Weber and S. Wermter. Object localization using laterally connected “what” and “where” associator networks. In *Proc. ICANN/ICONIP*, pages 813–20. Springer-Verlag Berlin Heidelberg, 2003.
- [14] C. Weber, S. Wermter, and A. Zochios. Robot docking with neural vision and reinforcement. *Knowledge-Based Systems*, 17(2-4):165–72, 2004.