Self-Calibrating Active Depth Perception Via Motion Parallax

Tanapol Prucksakorn¹, Sungmoon Jeong¹, Jochen Triesch², Hosun Lee¹ and Nak Young Chong¹

Abstract—A hallmark of biological systems is their ability to self-calibrate sensory-motor loops during their development. Understanding the principles of such self-calibration will enable the design of robots with similar autonomous learning abilities. Here we consider the problem of active depth perception based on motion parallax. When an observer moves sideways while looking at an object with a single eye, the eye rotation necessary to keep the object at the center of gaze provides information about the object’s distance. Based on the recently proposed active efficient coding (AEC) approach, we present a self-calibrating system which autonomously learns to represent image motion and perform compensatory eye rotations to keep the object fixated during side-to-side movements — thereby learning to actively estimate the object’s distance. A neural network is used to provide a calibrated depth estimate. We evaluate the system’s performance in simulation and in a hardware implementation.

Index Terms—Autonomous Learning, Motion Parallax, Active Depth Perception, Self-Calibration, Active Efficient Coding

I. INTRODUCTION

Depth perception is one of the most fundamental problems for biological and artificial vision systems. Humans use a number of different cues to infer the depth layout of a scene or estimate the distance of individual objects. Usually, depth perception in humans is an active process involving different kinds of eye and/or body movements. During active binocular vision, when an object is fixated with both eyes such that the optical axes of the two eyes intersect at a point on the object’s surface, the vergence angle between the two eyes provides an estimate of the object’s distance (Fig. 1a). When the observer moves sideways by a known distance as in Fig. 1b, the eye rotations necessary to keep the object at the centers of gaze, the so-called motion parallax, also provide information about the object’s distance. Finally, when the observer approaches the object with a known velocity, the changing optic flow pattern created by the movement also provides information about the object’s distance (Fig. 1c). Note that while active depth perception based on vergence eye movements obviously requires at least two eyes, depth perception based on motion parallax or optic flow requires only a single eye.

During the first few months of their post-natal development, humans and other animals autonomously learn how to use such depth perception cues. Thereafter, they continue to adapt and re-calibrate their vision when necessary, e.g., to compensate for growth of the eye, head, and body, but the underlying neural mechanisms are still largely unknown. Providing robots with similar abilities to autonomously learn and self-calibrate sensori-motor loops for active perception would make them more autonomous and robust. Here we present a system for the autonomous self-calibration of active depth perception based on motion parallax using a single moving camera.

Our system is based on the recently proposed active efficient coding (AEC) framework for the autonomous self-calibration of active perception. AEC is rooted in classic theories of efficient sensory coding in Neuroscience [1], [2], [3], which state that biological sensory systems encode sensory signals in an efficient manner by exploiting the statistical properties of the sensory signals. In particular, a specific goal of biological sensory systems is hypothesized to be the removal of redundancies from sensory signals to allow a maximally efficient encoding. AEC extends these classic theories of efficient sensory coding to the case of active perception, acknowledging the fact that the statistics of sensory signals are shaped by the behavior of the sensory system. Therefore, AEC not only attempts to learn an efficient encoding of the sensory signals but it also learns how to move the sense organs to improve the system’s coding efficiency [4], [5], [6], [7]. This has been successfully demonstrated for the case of active binocular vision, where a representation of binocular disparity and the control of vergence eye movements need to be learned [4], [5], as well as for active motion vision, where a representation of optic flow and pursuit movements of the eyes need to be learned [6], [7]. A model integrating the learning of active stereo and active motion vision has also been presented [8]. Interestingly, these models are not explicitly “trained”

![Fig. 1. Three different methods for active depth perception](image)

The authors are with the School of Information Science, Japan Advanced Institute of Science and Technology, I-1 Asahidai, Nomi, Ishikawa 923-1292, Japan. tanapol.pr@jaist.ac.jp, jeongsm@jaist.ac.jp, hosun.lee@jaist.ac.jp, nakyoung@jaist.ac.jp

The author is with Frankfurt Institute for Advanced Studies, Ruth-Moufang-Str. 1, 60438 Frankfurt am Main, Germany. triesch@fias.uni-frankfurt.de

¹The authors are with the School of Information Science, Japan Advanced Institute of Science and Technology, I-1 Asahidai, Nomi, Ishikawa 923-1292, Japan. tanapol.pr@jaist.ac.jp, jeongsm@jaist.ac.jp, hosun.lee@jaist.ac.jp, nakyoung@jaist.ac.jp

²The author is with Frankfurt Institute for Advanced Studies, Ruth-Moufang-Str. 1, 60438 Frankfurt am Main, Germany. triesch@fias.uni-frankfurt.de
II. METHODS

In this section, we explain our approach to the problem and give details of our simulation and experiment setup. The proposed model is shown in Fig. 2.

The system uses image inputs from a single camera which is moved laterally. A first image is captured from the original position of the camera. The second one is taken after the camera is moved laterally and panned.

A. Perception - Sensory Coding Model

We crop 128 by 128 pixels and 80 by 80 pixels in the center of the images to represent coarse scale and fine scale respectively. Fine scale images represent a foveal region in our eyes, as we can get more detail from the center of vision. The coarse scale represents a parafoveal area. The importance of having two scales has been investigated in [5] in the context of vergence control. Briefly, only having access to a coarse scale limits the accuracy of the system. Only having access to a fine scale may prevent the system from learning at all.

After we crop the images, we convert them to grayscale. Then, we extract 8 by 8 pixel patches from the grayscale images whose locations are generated by 1 pixel and 4 pixels shifts horizontally and vertically, for coarse scale and fine scale, respectively. The patches are sub-sampled using a Gaussian pyramid algorithm by a factor of 8 for the coarse scale, and 2 for the fine scale. The patches are converted to one-dimensional vectors. Then, they are rescaled to have zero mean and unit norm. Corresponding vectors \( x^C(t) \) from the first and the second images are then combined into a single vector, \( x^F(t) \) for both coarse scale and fine scale. Here \( i \) is the index of the patch, and \( j \in \{C, F\} \). \( C \) stands for coarse scale and \( F \) stands for fine scale. The first 64 elements of the vector are from the first image and the remaining are from the second image. This results in both vector \( x^C(t) \) and \( x^F(t) \) having \( P = 128 \) elements.

The patches are encoded using a sparse coding technique that represents each patch as a linear combination of basis functions drawn from an over-complete dictionary \( \phi^j(t) = \{\phi^j_n(t)\}_{n=1}^{N} \). The number of basis functions used in this experiment is \( N = 288 \). We create two separate dictionaries for coarse scale and fine scale. The dictionaries are randomly initialized and normalized. We use the matching pursuit algorithm [10] to estimate and find the sparse representation of the input vector by the weighted sum as follows:

\[
x^j_i(t) \approx \hat{x}^j_i(t) = \sum_{n=1}^{N} a^j_{i,n}(t)\phi^j_n(t) .
\]

We use the matching pursuit algorithm to estimate \( x^j_i(t) \), because it can limit the number of coefficients used, which reduces complexity. We set the maximum number of non-zero scalar coefficients \( a^j_{i,n}(t) \) to be 10 elements. This is to create sparseness for efficient encoding. The coefficients generated by the matching pursuit algorithm will then be used in the behavior learning part. We pool the squared activities of neurons representing the same feature type at different patch locations as follows:

\[
f^j_n(t) = \sum_{i=1}^{P} a^j_{i,n}(t)^2 .
\]

By estimating vector \( x^j(t) \), we can calculate the reconstruction error which is defined as:

\[
c^j(t) = \frac{1}{P} \sum_{i=1}^{P} \left\| x^j_i(t) - \sum_{n=1}^{N} a^j_{i,n}(t)\phi_n(t) \right\|^2 .
\]
In order to reduce the reconstruction error, we use gradient descent to update the dictionary. The dictionary elements are then normalized. This update improves the representation.

**B. Behavior - Reinforcement Learning**

For fine scale and coarse scale, the pooled activity is represented as:

$$f^i(t) = \begin{bmatrix} f_1^i(t) \\ f_2^i(t) \\ \vdots \\ f_{p}^i(t) \end{bmatrix}.$$ \hspace{1cm} (4)

In this framework, the state representation of the reinforcement learner is simply the combined pooled activity of coarse scale and fine scale:

$$f(t) = \begin{bmatrix} f^C(t) \\ f^F(t) \end{bmatrix}.$$ \hspace{1cm} (5)

The action is the command for rotating the camera. Specifically, the actions are the amount of degrees that the camera will rotate. We use a natural actor critic reinforcement learning algorithm, proposed in [11], to allow the system to learn how to choose an action from a set of $A$ actions. We use a softmax action selection for probabilistically choosing an action as follows:

$$\pi(f(t), a_t) = \frac{e^{z_a}}{\sum_{a \in A} e^{z_a}}.$$ \hspace{1cm} (6)

The activation $z_a$ for each action is given by:

$$z_a = \sum_{n=1}^{N} \theta_n(t) f_n(t),$$ \hspace{1cm} (7)

where $\theta_n(t)$ is a vector of weights from the state $f(t)$ to action $a$.

The reward that is given to the reinforcement learner is a sum of the negative reconstruction errors of coarse scale and fine scale:

$$r_t = -(e^C(t) + e^F(t)).$$ \hspace{1cm} (8)

**C. Depth Estimation**

When the eye movements have become accurate, we record two pan positions of the camera corresponding to the leftmost and rightmost position in vector $\vec{q}_t$ together with the depth $d_t$ in a data matrix $D$:

$$D = \begin{bmatrix} \vec{q}_1 & \vec{q}_2 & \vec{q}_3 & \cdots \\ d_1 & d_2 & d_3 & \cdots \end{bmatrix}. $$ \hspace{1cm} (9)

We use a simple two layer feed-forward neural network to map between the current pan position of the camera and the object’s distance. $q_t$ serves as the input to the neural network and the physical depth $d_t$ as the target output. The number of neurons in the hidden layer is 10 and we use a sigmoid transfer function in the hidden layer and a linear transfer function in the output layer. The Levenberg-Marquardt method [12] is used to train the connection weights between each layer. 70-percent of the data is reserved for training, 15-percent is validation and another 15-percent is for testing.

**III. SIMULATION**

**A. Simulation Setup**

We test the framework in simulation first. We use the virtual experiment platform V-REP for simulating the environment. The framework is implemented in MATLAB. The simulation environment is shown in Figure 3. The scene comprises a HOAP3 robot, an object, and background.

The lateral movement of the robot is generated by simply changing the position of the robot in the environment. The initial distance between the object and the robot is 1 meter. Every iteration, the robot moves from the original position to the left by 50 centimeters. The robot pans the left eye camera according to the action received from the reinforcement learning part. Then the robot moves back to the original position. At the end of each movement the robot uses the left eye camera to capture the images. The two images will be the input to the sensory coding model. After the two images are processed, we get a camera movement command for panning the camera. We use actions (in degrees) $A = \{-0.2, -0.1, -0.05, 0, 0.05, 0.1, 0.2\}$.

After moving in the left direction, the system has 15 time steps to try to re-fixate the object. After that, the robot is moving to the right and has another 15 time steps to try to re-fixate the object. After that, we randomly change the position of the robot in the environment. The robot pans the left eye camera according to the action received from the reinforcement learning part. Then the robot moves back to the original position. At the end of each movement the robot uses the left eye camera to capture the images. The two images will be the input to the sensory coding model. After the two images are processed, we get a camera movement command for panning the camera. We use actions (in degrees) $A = \{-0.2, -0.1, -0.05, 0, 0.05, 0.1, 0.2\}$.

After moving in the left direction, the system has 15 time steps to try to re-fixate the object. After that, the robot is moving to the right and has another 15 time steps to try to re-fixate the object. After that, we randomly change the position of the robot in the environment. The robot pans the left eye camera according to the action received from the reinforcement learning part. Then the robot moves back to the original position. At the end of each movement the robot uses the left eye camera to capture the images. The two images will be the input to the sensory coding model. After the two images are processed, we get a camera movement command for panning the camera. We use actions (in degrees) $A = \{-0.2, -0.1, -0.05, 0, 0.05, 0.1, 0.2\}$.

**B. Simulation Results**

Examples of object fixating of the framework are shown in Fig. 4. To evaluate how well the framework can fixate the object, we define the mean absolute error (MAE)

$$\text{MAE}(t) = \frac{1}{1000} \sum_{k=0}^{999} |\phi(t + 14 + 15k) - \phi^*(t + 14 + 15k)|,$$ \hspace{1cm} (10)
Fig. 4. Fixating an object. Two images from before and after the movement are superimposed as anaglyphs. After training, the system holds fixation accurately.

$$\phi(t)$$ is the rotation angle of the eye at time $$t$$ and $$\phi^*(t)$$ is the optimal rotation angle at time $$t$$. The MAE shows the average error throughout the learning process. It is shown in Fig. 5. We can see that at 540,000 iterations the MAE is around 0.2 degrees.

After the training is finished, we test the framework for 1000 sets of simulations. The system selects the action that has the highest probability from softmax activation function, i.e., greedy action selection. The depth of the object is varied from 1 meter to 2 meters in steps of 10 centimeters in the same way as in the training period. The results and errors are shown in Table I. We can see that the system can generate accurate eye movements, and it can extract depth information from the eye movement. The error histograms are shown in Fig. 6a and Fig. 6b for eye movement and depth estimation, respectively.

### IV. REAL WORLD EXPERIMENT

#### A. Hardware Setup

For the real world experiments, a camera is attached on a linear actuator facing a target object on the floor. As shown in Fig. 7, the motion parallax images are generated by moving the camera (blue eye symbol) laterally in the same way as in the simulation. We manually varied the depth between the camera and the object.

The whole framework is implemented in MATLAB. A micro-controller, Arduino, is used to control the linear actuator. The control and acquisition flow of the system is the same as that in the simulation.

For the training data, we capture all the images generated from lateral movements with various distances to the target object; from 1 meter to 2 meters (each step has 10 centimeters interval). Then we use the set of images that we have gathered to train and test the framework. The process and action list are the same as in the simulation.

#### B. Experiment Results

An example of object fixating of the framework is shown in Fig. 8. The MAE of the experiment is shown in Fig. 9. After training, we test the framework for 1000 sets of experiments. The actions are selected by the greedy strategy. The results are shown in Table II. Similar to the simulation, we use the eye movement to estimate the depth of the object. The error histograms of eye movement and depth estimation are shown in Fig. 10a and Fig. 10b, respectively.

From the simulation and the experiment results, the MAE of the real world experiment tends to be smaller than the one from simulation, because there are 100 different objects to be

<table>
<thead>
<tr>
<th>Real Depth (m)</th>
<th>Desired Eye Movement (degrees)</th>
<th>Average (STD) Output Eye Movement (degrees)</th>
<th>Average (STD) Output Depth (m)</th>
<th>Depth Estimation Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>1.60</td>
<td>1.35 (1.25)</td>
<td>1.15 (0.10)</td>
<td>15.57%</td>
</tr>
<tr>
<td>1.10</td>
<td>1.46</td>
<td>1.12 (1.16)</td>
<td>1.23 (0.13)</td>
<td>12.10%</td>
</tr>
<tr>
<td>1.20</td>
<td>1.34</td>
<td>1.14 (1.07)</td>
<td>1.28 (0.11)</td>
<td>6.71%</td>
</tr>
<tr>
<td>1.30</td>
<td>1.23</td>
<td>1.21 (0.67)</td>
<td>1.35 (0.12)</td>
<td>4.32%</td>
</tr>
<tr>
<td>1.40</td>
<td>1.15</td>
<td>1.09 (1.05)</td>
<td>1.49 (0.14)</td>
<td>6.94%</td>
</tr>
<tr>
<td>1.50</td>
<td>1.07</td>
<td>1.01 (0.87)</td>
<td>1.58 (0.12)</td>
<td>5.99%</td>
</tr>
<tr>
<td>1.60</td>
<td>1.01</td>
<td>1.08 (0.66)</td>
<td>1.61 (0.11)</td>
<td>1.16%</td>
</tr>
<tr>
<td>1.70</td>
<td>0.94</td>
<td>1.01 (0.45)</td>
<td>1.69 (0.10)</td>
<td>0.07%</td>
</tr>
<tr>
<td>1.80</td>
<td>0.89</td>
<td>0.96 (0.42)</td>
<td>1.78 (0.10)</td>
<td>0.95%</td>
</tr>
<tr>
<td>1.90</td>
<td>0.84</td>
<td>0.89 (0.54)</td>
<td>1.80 (0.10)</td>
<td>4.78%</td>
</tr>
<tr>
<td>2.00</td>
<td>0.80</td>
<td>0.87 (0.58)</td>
<td>1.80 (0.14)</td>
<td>9.67%</td>
</tr>
</tbody>
</table>
trained in the simulation. While in the real world experiment we have only one texture for training.

According to the results, the framework can estimate the depths of the object. However, there are some eye movement errors as we can see in the histograms. For some depths, the framework outputs the same result. This is because there are still eye movement errors. Also, the image resolution is not enough to represent some depths. So, for the depths that are close together, there is a chance that the generated eye movement is the same.

This problem could be solved by increasing the size of the image patch and resolution of input images or reducing the sub-sampling factor. Thus, we could get more information to represent depths with more accuracy. However, the computation time will be increased.

V. ROBUSTNESS TEST

The simulation and experiment show that our system is able to learn to generate eye movements to stabilize the object in the image center. To demonstrate that the system has an ability of developmental learning, we apply a perturbation to the system. We rotate the camera clockwise by 20 degrees and keep training the system. The rotation of the camera for the real world experiment is simulated by rotating the input images. As shown in Fig. 11, a noticeable increase in error occurs when the perturbation is applied. The figure also shows that the system can recover from the perturbation, i.e. the system is able to learn to adapt to the changes in configuration.

VI. DISCUSSION

This research extends the recent works [6], [7] on self-calibration of active motion vision. We applied it to create a model that learns to keep fixating an object when the camera

![Fig. 7. Linear actuator and the object](image)

![Fig. 8. Fixating an object in the real world experiment, analogous to Fig. 4](image)

![Fig. 9. MAE of real world experiment](image)

![Fig. 10. Error histogram of the experiment](image)

![TABLE II](image)
is moving laterally. We showed that we can utilize the eye movements for estimating the depth of an object by using a neural network. The difference from their works is that we consider self-induced motion parallax, which helps the system to extract depth information.

According to the simulation results, the proposed framework can successfully estimate depth and generate eye movements to keep the object at the center of gaze. Both action and perception learning are trained by the same reconstruction error function. The framework can simultaneously learn to choose actions and create visual representations to understand the motion parallax effect. Moreover, the proposed model can be applied with any single camera system, because it does not depend on details of the hardware and its configuration.

However, there is still much room of improvement for this research. The depth estimation using the neural network relies on supervised training. It would be more interesting if the entire model is unsupervised. There is also interesting work on active vision of head and eye movements [13]. They also consider the motion parallax effect as a method to estimate depth. However, their approach is utilizing the head and eye rotation to geometrically solve for depths without lateral body movements. Having the ability to learn to estimate depth by head and eye rotation would be superior for robots that are difficult to move laterally. Also we could consider velocity control for the eye rotation rather than position control to estimate depth based on the rotation speed of the camera that is required to compensate for a lateral movement speed. These improvements should be considered in future work on the autonomous learning of active depth perception.

ACKNOWLEDGEMENT

This research is supported by the project Autonomous Learning of Active Depth Perception: from Neural Models to Humanoid Robots from Japan Agency for Medical Research and development, AMED and the German BMBF.

REFERENCES