Adaptive Stimulus Selection in Estimating Vestibular Model Parameters in the Rod-and-Frame Task

Bachelor Thesis in Artificial Intelligence

Radboud University

Nijmegen, the Netherlands

June 18th, 2018

Author:

A. M. Ernest (Anneloes)

Student number: s4579259

Supervisor:

L. P. J. Selen (Luc)[1]

[1] Donders Institute for Brain, Cognition and Behaviour, Radboud University Nijmegen, the Netherlands.

Abstract

The perception of upright can be biased by the visual context. The extent of the bias is linked to the functioning of the vestibular organ. The rod-in-frame task is an excellent method to test this biasing effect. The response to the task can be fitted by a 4-parameter Bayesian model. In this thesis project, an algorithm that adaptively selects stimuli based on entropy was implemented (in Python) to speed up the convergence of the model's parameters. Tests with a generative agent showed convergence of some of the parameters after only 500 trials (in comparison with ±1600 trials). Experimental tests with real subjects show similar results. Although there are many aspects that could be improved, the results indicate that adaptive stimulus selection can indeed significantly reduce the number of trials needed for convergence of the model's parameters. With further research, there might be a possibility to use the model with adaptive stimulus selection as a clinical tool to help patients with vestibular deficiencies.

Introduction

The vestibular system has two well-known main functions. Firstly, the receptors give information about the position of the body in relation to gravity. Secondly, the receptors signal changes in the direction and speed of head movements. The vestibular system enables our sense of spatial orientation and subserves balance (Kolb, Whishaw, & Teskey, 2014). Other studies have also found evidence for the involvement of the vestibular system in arterial blood pressure (Tanaka, Abe, Awazu, & Morita, 2009) and cerebral blood flow (Serrador, Schlegel, Black, & Wood, 2009).

Many researchers have proven that with age, the functioning of the vestibular organ decreases (Agrawal, Carey, Della Santina, Schubert, & Minor, 2009; Zalewski, 2015). There are multiple therapies available that have proven to be effective in compensating for e.g. the loss in balance due to vestibular deficiencies (Gillespie et al., 2003; Macias, Massingale, & Gerkin, 2005). To help coping with the consequences of the decline in functioning, the patients must first be tested to what extent the functioning of vestibular organ has decreased and particularly for what kind of situations, in order to optimize the help needed.

The testing of the functioning of the vestibular organ can be done with a rod-and-frame task. In this task, the participant is asked whether the direction of a briefly flashed rod was observed to be clockwise (right) from upright. The rod itself is contained within a frame, of which the orientation is altered every trial. The frame is used to bias the observation of the rod. Although every subject will show a biasing effect, the magnitude of the effect depends on the vestibular functioning. When the frame highly biases the observation, it can be deduced that the functioning of the visual-vestibular interaction has weakened. Subsequently, the participant relies more on the visual context (frame) to infer whether the object is upright, rather than using the information from the vestibular organ (Alberts et al., 2016).

In the paper by (Alberts et al., 2016), a Bayesian model is described in which the responses to the rod-and-frame tasks are explained in terms of six parameters. The six parameters consist of noise in the signal of the otoliths ($\sigma_{otoliths}$), prior knowledge of the head-in-space orientation, the rate at which the otoliths' noise is increased by the head-in-space orientation, the visual contextual vertical ($\kappa_{vertical}$), the visual contextual horizontal ($\kappa_{horizontal}$), and the way $\kappa_{vertical}$ and $\kappa_{horizontal}$ change with frame orientation (τ).

Figure 1 shows how the cumulative response density distribution changes with the frame orientation. In the first frame orientation of -¼π, all sides of the frame equally contribute to the probability of seeing an upright rod. The cumulative response density distribution is nicely in the middle of the otoliths' peak and thus can be concluded here that the frame hardly has any effect on the perception of upright. If you look at the graph with frame orientation of -0.1π, we see that the probability density of the frame has shifted more towards the otoliths' peak. There is also a notable difference in the peaks of the frame as either the vertical sides or the horizontal sides now have a larger impact on the perception of upright. The cumulative response density distribution has become steeper and has also slightly shifted towards the first high peak of the frame. When the frame is not altered in orientation, and thus is displayed as a perfect square on top of the rod, the peak of the frame is lined up the peak of the otoliths. We see a nice cumulative response density distribution which is steeper than the cumulative response density distribution of the first frame orientation of -¼π. As the frame orientation is even further increased, the peak of the frame moves away from the peak of the otoliths and the cumulative response density distribution shifts slightly with the move of the peak of the frame.

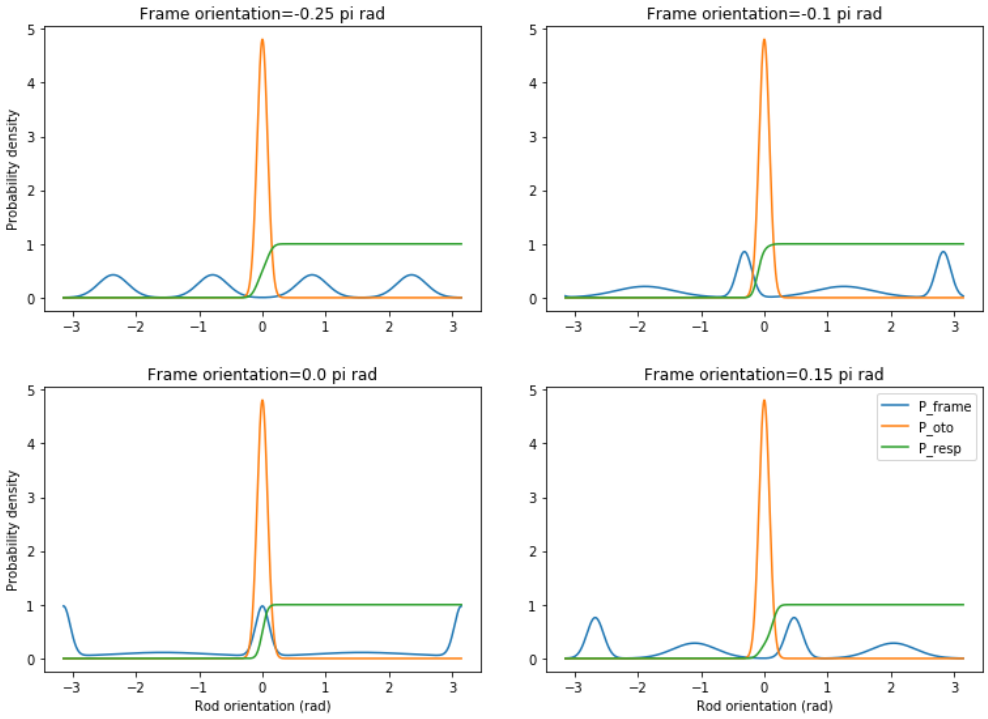

Figure 1: Influence of frame orientation on the clockwise cumulative response density distribution, depicted by the green line. The blue line represents the information from the frame and the orange line represents the information by the otoliths.

In earlier studies, independent psychometric curves were built for every frame orientation (Cadieux, Barnett-Cowan, & Shore, 2010; Lopez, Mercier, Halje, & Blanke, 2011; Magnussen, Landrø, & Johnsen, 1985). This psychometric function expresses the bias and variability of a participant on a rod-and-frame task (Wichmann & Hill, 2003). The responses to different frame-and-rod stimuli are used to update the psychometric function for the corresponding frame.

To accurately estimate parameters for such a model, one would need to test a lot of stimuli (Alberts et al., 2016, used 1620 combinations of rod and frame stimuli per condition). Using this number of stimuli to estimate the functioning of the vestibular organ (as for example in a clinical setting) is not as quick as one would like. There is a need for reducing the number of stimuli, which can be done by adaptively selecting the stimuli. Adaptive sampling for stimulus selection is a means of determining the best next stimuli, based on the current stimulus and response, which will give the most expected information for some parameter(s) (in this case

of the Bayesian model). There have been many applications of adaptive sampling in research, each having a slightly differing algorithm but all based on Bayesian principles: (Kontsevich & Tyler, 1999) propose a Bayesian adaptive method for estimating the slope (variability) and threshold (bias) of a psychometric function; (Kujala & Lukka, 2006) built further on this and adaptively sampled 2D stimuli for a 2D psychometric model; in a slightly other research domain (Pillow & Park, 2016) used adaptive stimulus selection in neurophysiology experiments, namely neuron tuning.

In this paper, I will describe whether it is possible to reduce the number of presented stimuli to estimate a subject's parameters of the Bayesian model using adaptive sampling in the rod-and-frame task. I will build further on the algorithm described by (Kontsevich & Tyler, 1999), which will be described in depth, and see whether we can extend this particular algorithm to estimate four parameters of the Bayesian model as described by (Alberts et al., 2016).

This thesis aims to offer a proof of principle: is it possible to adaptively select stimuli in a rod-and-frame task that enables us to fit a Bayesian model explaining the data of the subject within a reasonable time frame?

The corresponding research questions are:

1. *Is it possible to quickly estimate the four parameters of a Bayesian Model that fit the responses of a subject on a Rod-and-frame task where the stimuli are adaptively selected?*

2. *And specifically, is this much faster than the classical approaches as used in* (Alberts et al., 2016)*?*

Methods

The Algorithm

Parameters

The Bayesian model by (Alberts et al., 2016) has been slightly modified to meet this thesis' criteria of four parameters. The original model of (Alberts et al., 2016) contained six parameters that were inferred not only on different frame and rod orientations, but also based on different head orientations. In this thesis I reduced the model to four parameters, thereby excluding the effects of head orientation. The parameters that remain are $\kappa_{vertical}$, $\kappa_{horizontal}$, $\sigma_{otoliths}$. and τ. The model already uses a von Mises distribution instead of a Gauss distribution for the visual contextual vertical and horizontal. This is why I decided to also transform $\sigma_{otoliths}$ to a value that corresponds to the same standard deviation in the von Mises distribution, $\kappa_{otoliths}$. Therefore, an approximation formula is used that transforms the σ-values of a Gauss distribution to the κ-values of a von Mises distribution. Using a von Mises distribution instead of a Gauss distribution allows the integration interval to be unchanged while still having the exact same surface underneath the curve with differing standard deviations. The names I will be using for the parameters in the remainder of this thesis will be κ_{ver}, κ_{hor}, τ, κ_{oto}, respectively.

Parameter ranges

For each of the parameters described above, I determined a range in which the values vary based on the average results and standard deviations in (Alberts et al., 2016). The ranges that were used in the different tests are further discussed under *Results*.

The number of frame orientations was set to 11, ranging from -¼π rad (-45°) to ¼π rad (45°). The number of rod orientations was set to 30, ranging from -10° to 10°. These ranges were used to infer the stimulus pairs in the rod-and-frame task.

Lookup

I need to make a table in which we can find the probability of giving a clockwise (CW) response (=1 response) for every parameter set and rod-frame stimulus pair. I decided to do this with multiple for-loops that loop over the multiple parameter ranges. The first thing is determining the κ's: κ_1 and κ_2. The κ's determine the way that κ_{ver} and κ_{hor} change over the frame orientations, influenced by τ (*also see Figure 1*). The next step is to compute the contextual prior provided by each frame. This is done by computing the priors of each side of the frame with a von Mises and taking the average of the resulting four probabilities. Also, I need to compute the probability distribution of the otoliths over the rod orientations, with a von Mises as well. The last crucial step is to compute the cumulative density of all these distributions.

This table becomes very large, very quickly, and takes a lot of time making. Therefore the table is only made once for every parameter range configuration. The table is then saved and loaded into the code, which makes running orders of magnitude faster.

Priors

The probability of a particular parameter set to be the true set is called the prior probability. Initially, these prior probabilities, or priors as I shall continue to call them, are evenly distributed over all possible parameter sets. After each trial, the priors are updated in order to determine which parameter set is most likely to be the true parameter based on the current responses to the presented stimuli.

Generative agent

A generative agent is made in order to give responses based on a predefined parameter set. Using a generative agent gives the opportunity to analyse the accuracy of the algorithm's

estimates since the to-be-estimated parameters are known. The responses of this agent are used to update the prior. The agent is built using the model described by (Alberts et al., 2016). The agent is built once, and has an inbuilt probability distribution of giving a CW response for every rod-and-frame orientation. During each trial, the agent is asked to give a response based on the current rod-and-frame stimulus. This probability is taken out of the predefined probability distribution and used to give a response. For checking the accuracy of the estimates by the algorithm, I initialized multiple agents with differing parameters. This will give a better idea about the performance of the algorithm. The parameter sets of the agents are further discussed under *Results*.

Algorithm for selecting stimuli

The algorithm by (Kontsevich & Tyler, 1999) is an algorithm that chooses the stimulus to be presented on the next trial. This particular stimulus is chosen such that it maximizes the gain of information about the Bayesian model parameters. The gain of information is a measure by means of entropy, i.e. the algorithm tries to find the stimulus that minimizes the expected entropy. There is a total of eight steps that have to be done during each trial. Note: All of these steps are done for both getting a clockwise (CW) response and counter-clockwise (CCW) response.

The first step is to calculate the probability of getting a particular response (r) after presenting a particular stimulus pair ($x = (frame, rod)$) at the next trial. Here, we calculate the conditional probability at a certain trial (t) of response r given that stimulus x was shown. This is done by computing the conditional probability over all possible parameter sets (λ) and weighing each of those conditional probabilities by the probability of that parameter set λ to be the true one. We calculate this probability for every response and stimulus combination.

$$p_t(r|x) = \sum_{\lambda} p(r|\lambda, x) \cdot p_t(\lambda)$$

The second step is to estimate the posterior probabilities of each parameter set λ, given that at the next trial the participant will give response r after presenting stimulus pair x. Here, we calculate the conditional probability of parameter set λ given that stimulus pair x gave response r at trial t.

$$p_t(\lambda|x, r) = \frac{p(r|\lambda, x) \cdot p_t(\lambda)}{\sum_{\lambda} p(r|\lambda, x) \cdot p_t(\lambda)}$$

This calculation is done with an Einstein summation over the prior probabilities and the lookup table. These values are then standardized by 1/the sum of the resulted Einstein sum. The Einstein summation convention transforms all quantities in the expression into scalars which allows easy computation.

The third step is to estimate the entropy of the posterior probabilities, given that at the next trial t stimulus x will give response r.

$$H_t(x, r) = -\sum_{\lambda} p_t(\lambda|x, r) \cdot \log(p_t(\lambda|x, r))$$

The fourth step is to estimate the expected entropy for each stimulus x.

$$E\big(H_t(x)\big) = H_t(x, success) \cdot p_t(success|x) + H_t(x, failure) \cdot p_t(failure|x)$$

The fifth step is to find stimulus x that minimizes the expected entropy.

$$x_{t+1} = argmin_x E[H_t]$$

The sixth step is to run the next trial with the stimulus that was selected at *step five* (x_{t+1}). This trial will give response r_{t+1}.

The seventh step is to update the prior probability distribution. If the response of the subject is CW, the priors are updated by the CW table and if the response of the subject is CCW, the priors are updated by the CCW table.

$$p_{t+1}(\lambda) = p_t(\lambda|x_{t+1}, r_{t+1})$$

Finally, in the eighth step, the algorithm determines the new estimate of the model parameters based on the new priors. This estimate is the weighted average of all parameter probabilities, also known as the expected value estimate. In the formula below, λ_{t+1} denotes the new expected value estimate and $p_{t+1}(\lambda)$ denotes the updated prior probability (*see step seven*) of the parameter sets λ.

$$\lambda_{t+1} = \lambda \cdot p_{t+1}(\lambda)$$

All of these steps are executed for a fixed number of trials, in most cases 500 trials. This number is based on the convergence of some of the parameters.

The final estimate of the parameter set could either be based on the expected value estimate or the maximum a priori (MAP) estimate. As described above (*see step eight*), the expected value estimate is the weighted average of the parameter probabilities. The MAP estimate is the parameter value with the highest probability.

Testing the Algorithm

Initially, the algorithm is tested with a generative agent. During each trial, the stimuli and their response is saved. Having only this information, the algorithm is able to recreate the estimates that it had calculated before. Since each test was conducted 10 times in a row, I also

kept the estimated parameters from the last trial in each test such that I was able to plot the results after the 10-fold run rapidly. Next, experimental tests were done with two subjects. Here, we conducted the rod-in-frame task for 500 trials where each stimulus pair was selected adaptively using the algorithm described above. In these experiments, the real values of the parameters are not known.

Results

Estimates with the Generative Agents

Beneath in *Table 1* is listed which parameters the different agents had. The parameter set of agent 3 and 4 is based on the exact averages of the subject data as found in (Alberts et al., 2016). The parameter sets of agent 1 and 2 are based on their ranges, where the parameters of agent 1 are in the middle of the range and the parameters of agent 2 deviate a bit from the middle to show the accuracy of the algorithm when the to-be-estimated parameters are not in the middle of the range.

Table 1: Parameters of the generative agents.

	κ_{ver} (°)	κ_{hor} (°)	τ	κ_{oto} (°)
Agent 1	4	40	0.8	2.2
Agent 2	5	35	0.85	2.3
Agent 3	4.87	52.26	0.80	2.21
Agent 4	4.87	52.26	0.80	2.21

In *Tables 2-4*, the used parameter ranges for the different agents is described. Agent 1 and agent 2 have a smaller parameter range that is loosely based on the averages and standard deviations of the findings in (Alberts et al., 2016). The parameter ranges for agent 3 and 4 are a bit larger and offer more room for the convergence of the parameters. The parameter range

for agent 4 differs from the parameter range of agent 3. Here, the range of κ_{hor} is set to a smaller range to see whether that affects the convergence of τ.

Table 2: Parameter ranges for both agent 1 and agent 2. These ranges are strictly based on the averages and standard deviations of the findings by (Alberts et al., 2016).

	κ_{ver} (°)	κ_{hor} (°)	τ	κ_{oto} (°)
start	2	30	0.70	1.95
end	7	75	0.95	2.50
nr of samples	15	15	10	10

Table 3: Parameter ranges for agent 3. These ranges are larger than the ones for agent 1 and 2.

	κ_{ver} (°)	κ_{hor} (°)	τ	κ_{oto} (°)
start	2.5	22	0.6	1.4
end	7.5	80	1.0	3.0
nr of samples	15	15	10	10

Table 4: Parameter ranges for agent 4. These ranges are the same as for agent 3, except for the range of κ_{hor}.

	κ_{ver} (°)	κ_{hor} (°)	τ	κ_{oto} (°)
start	2.5	30	0.6	1.4
end	7.5	75	1.0	3.0
nr of samples	15	15	10	10

Convergence

In *Figure 2*, the convergence of two out of four parameters is shown. Already after 100 trials (*c*), the probabilities for both κ_{ver} and τ are closing in on a more specific value. The ideal convergence of a parameter would be one peak at the actual parameter value, with all others at zero. This would mean that this one parameter is the true one, with high confidence.

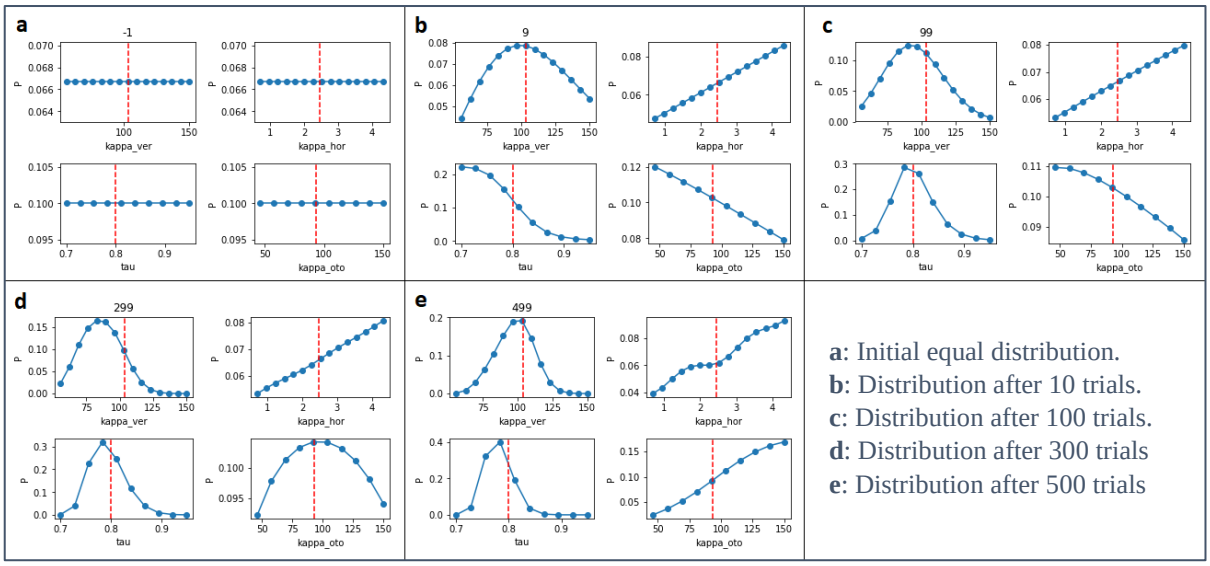

Figure 2: This is what the marginal prior distributions (the blue line with dots) look like during certain trials. The red dotted lines indicate the value of the actual parameters. You can clearly see convergence for both κ_{ver} and τ.

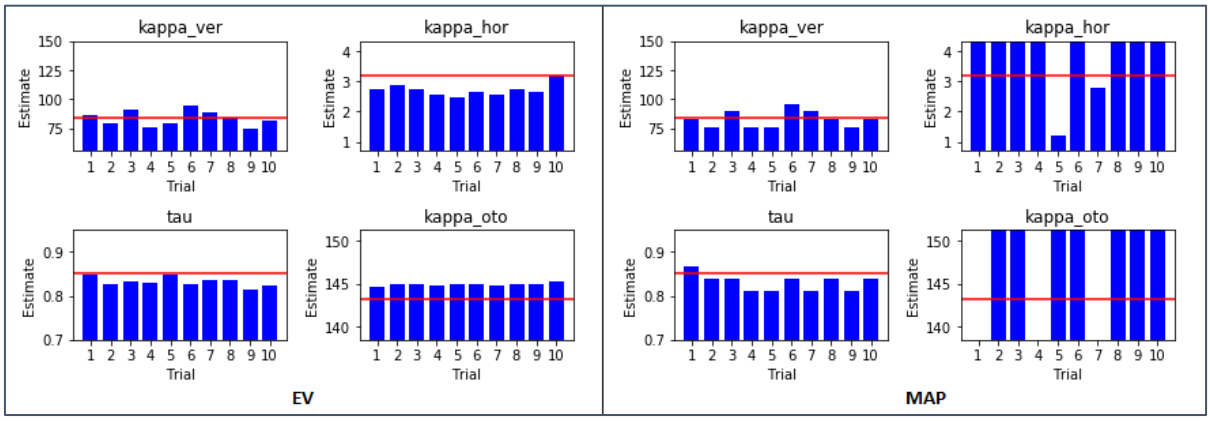

Figure 3: Estimates of the parameters of agent 1. The left panels show the expected value (EV) estimate, and the right panels show the maximum a priori (MAP) estimate.

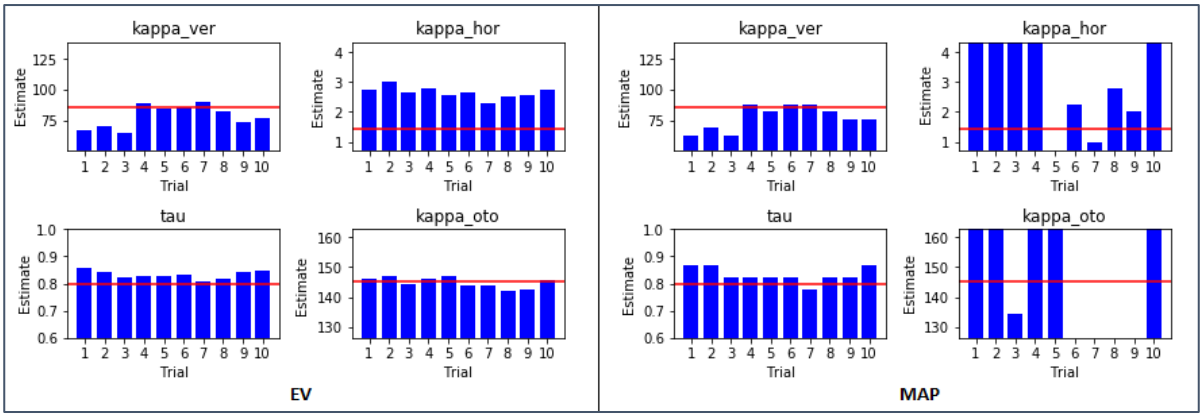

Figure 4: Estimates of the parameters of agent 2.

Figures 3-6 show the estimates of each test and agent. Each picture contains estimates from 10 tests of a single agent. The estimates are indicated by the height of the blue bars. The red line indicates the agent's true parameter value. In every figure, the graphs on the left side are the expected value estimates and the graphs on the right side are the MAP estimates. The values on the y-axis are limited to the respective parameter ranges that were used. This means that the tests with no bars in the graph have an estimated parameter value that is equal to the lowest bound of the range.

The expected value estimates of the first agent are very promising (*see Figure 3*). There is hardly any difference with the actual parameters of the agent. However, if one would look to the MAP estimates for the same agent, we see that the estimates for only κ_{ver} and τ are accurate. The MAP estimates of κ_{hor} and κ_{oto} are off. Although the MAP estimate of κ_{hor} does indicate that there are some probability peaks in the convergence of the estimate which are not at the bounds of the range, there is no such evidence for the parameter κ_{oto}. The MAP estimate of κ_{oto} is either at the lowest or the highest bound of the parameter range. If the expected value estimate would also be at the bound, this would indicate convergence at the bound. If the expected value estimate is not at the bound, it can be deduced that there is no convergence at all.

By adjusting the to-be-estimated parameters slightly from the middle in agent 2, it can be seen that the estimates for both κ_ver and τ change with the adjustments in the parameters (see Figure 4). There does seem to be a bit more variability in the estimates. The parameter estimates of κ_hor and κ_oto do not alter with the adjustments in the agent's parameter set. This confirms the previously mentioned notion that κ_hor and κ_oto are difficult to estimate.

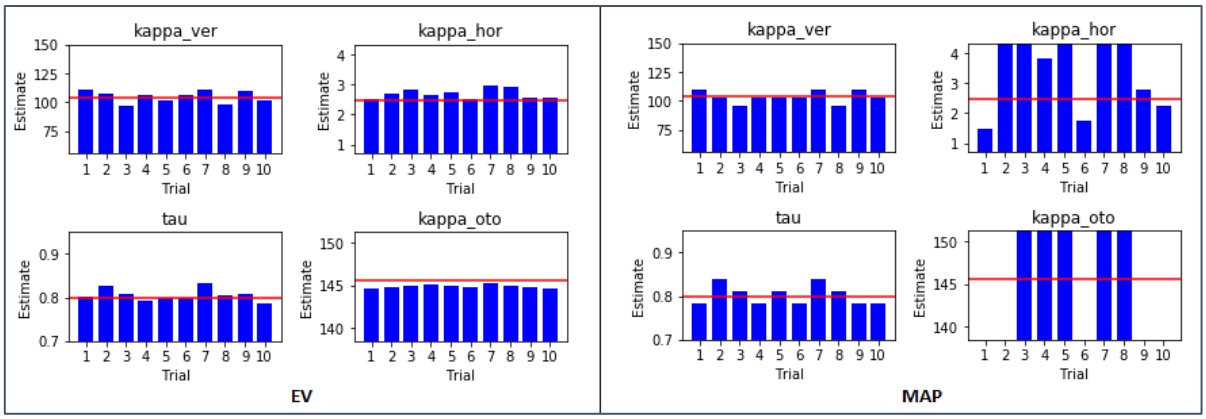

Figure 5: Estimates of the parameters of agent 3.

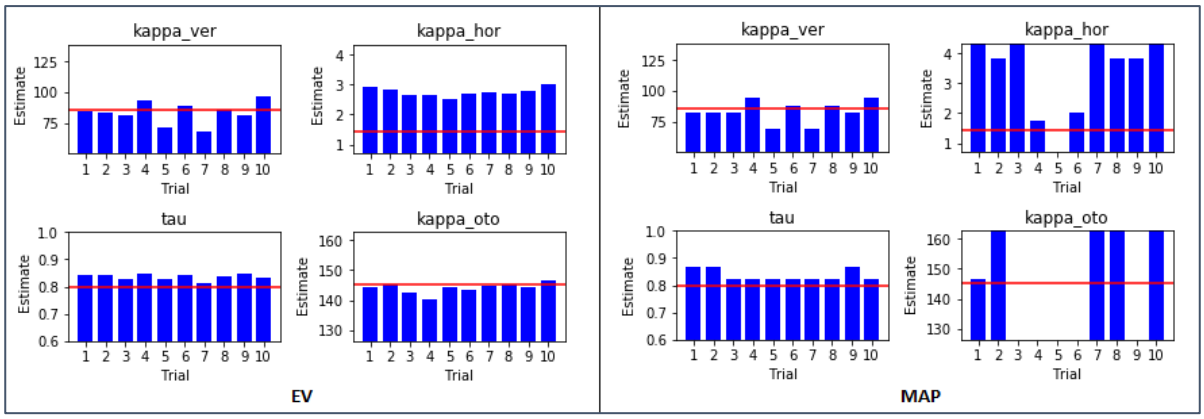

Figure 6: Estimates of the parameters of agent 4.

For the estimates of the parameters of agent 3 and agent 4 (*see Figure 5 and Figure 6 respectively*), there seems to be more difficulty with accurately estimating κ_{ver}. However, both the expected value estimate as the MAP estimate give almost identical values, proving convergence of κ_{ver}. The parameter τ seems to be overestimated in both agents and also by both estimate measures. There is great overestimation of κ_{hor} too, but it is already known from the previous two agents that this parameter was more difficult to estimate. Since the estimates for κ_{hor} are mostly around the middle of the range, and the MAP values are either very high or low, it can be concluded that there is no convergence of κ_{hor}. The same goes for the parameter κ_{oto}. There was only one trial in total that had a good MAP estimate of κ_{oto} (agent 3, trial 1, *Figure 5*).

Furthermore, the parameters κ_{ver} and τ already showed an obvious convergence shape after only 40 trials during tests with the generative agents.

Estimates with Real Subjects

Since the true parameters of the subjects in the experimental setting are unknown, the parameter ranges need to be chosen as wide as possible. Since there is some information about the ranges in which the parameters occur based on the results in (Alberts et al., 2016), and it is desired to have some comparison with the generative agents, it was decided that the parameter ranges for agent 3 (*see Table 3*) will be used for the experiment. These ranges will offer the most flexibility within reasonable bounds.

Figure 7 shows that for both subjects there is convergence for some of the parameters. We can clearly see convergence of κ_{ver} and τ for both subjects since the EV and MAP estimates of each parameter are similar. In subject 1, there is also some evidence for the convergence of κ_{hor}. The estimates of κ_{oto} for both subjects did not show convergence as the EV estimates are averaged out by the rest of the parameter probabilities.

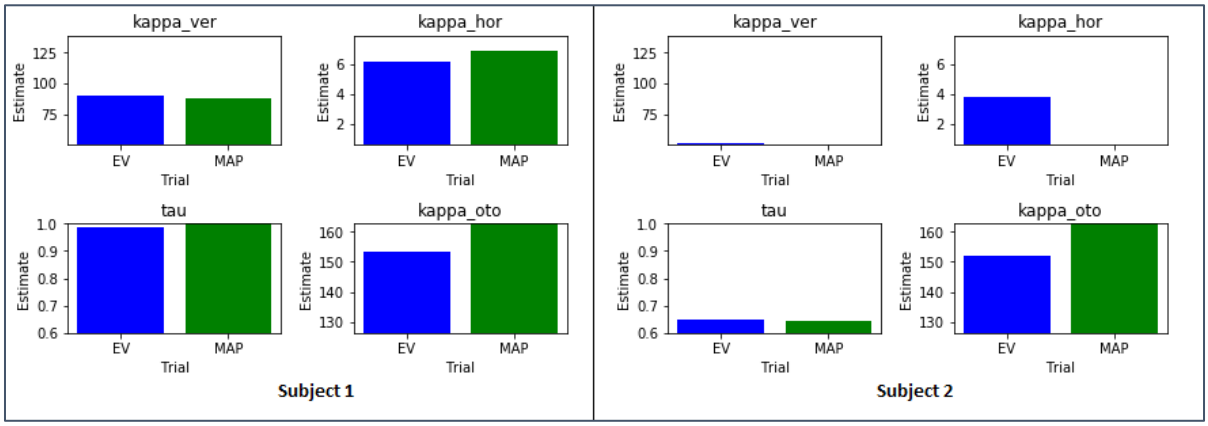

Figure 7: Estimates of the trials with subjects. The blue bars show the expected value estimates and the green bars show the MAP estimates. The estimates of subject 1 and subject 2 are shown on the left and right respectively.

Stimuli Selection

The stimulus pair selections during the two subjects' tests and two of the agents' tests are shown in *Figure 8*. From the agents' tests, I chose to show the stimulus pair selections for agent 2 and agent 3 during run number 2. A test with agent 2 was chosen because the true parameter set is not in the middle of the range. Therefore this agent's parameter set will probably resemble the unknown parameter set of the subjects more since these are not likely to be in the middle either. In particular, a test with agent 3 was chosen because the parameter ranges for this agent are the exact same as for the subjects. *Figure 8* shows which frame and rod orientation was presented during each trial of a single test for the two subjects and two agents described above. The frame orientations are shown in the graphs with the blue dots, and the rod orientations are shown in the graphs with the red dots.

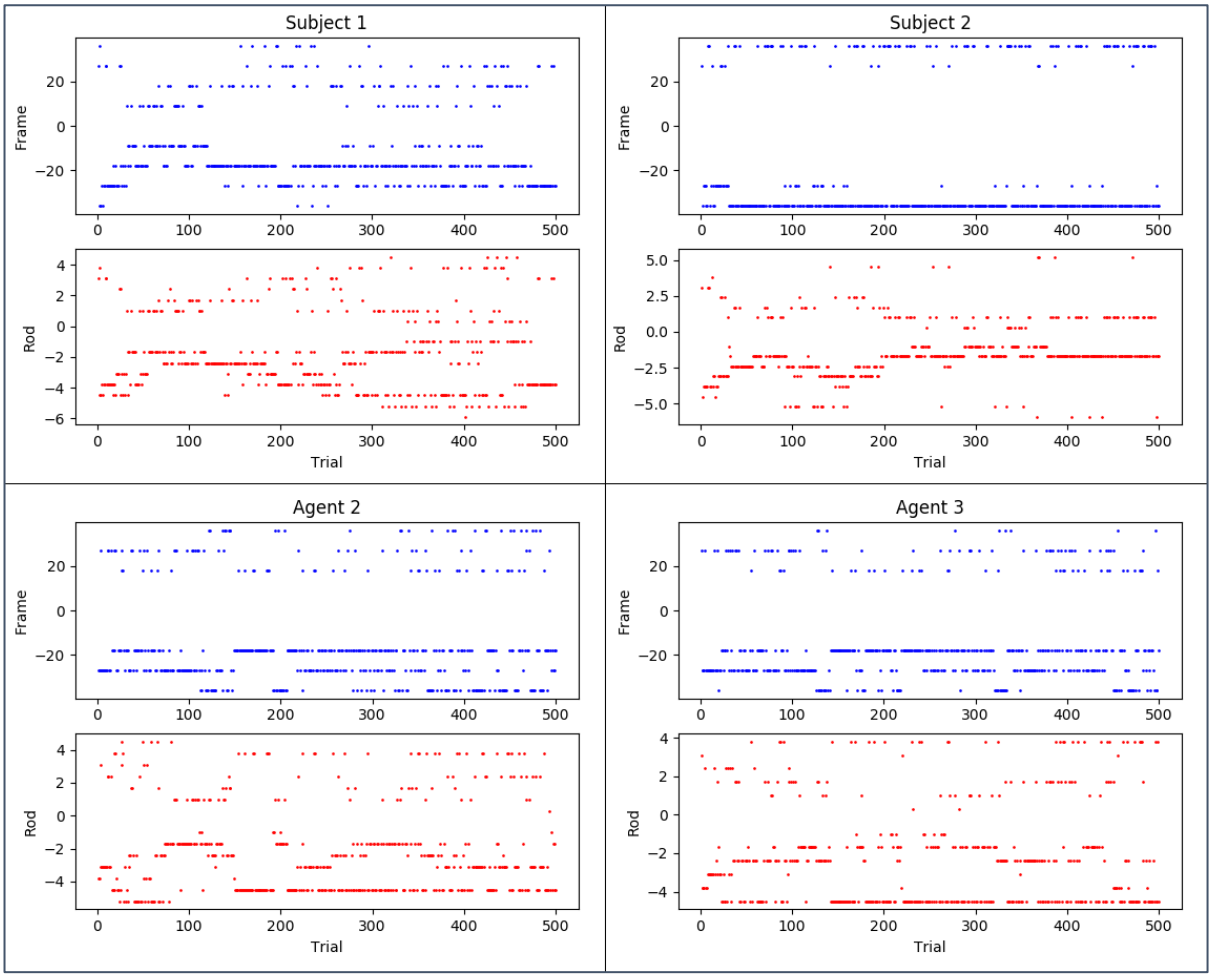

Figure 8: Examples of stimuli selections by the algorithm for both subjects and two generative agents' tests.

Comparing the graphs of the subjects in *Figure 8*, it is clear that both the frame and rod orientations for subject 1 were much more alternating than those for subject 2. It is especially notable that subject 2 mostly got very large negative frame orientations with very small negative rod orientations.

The stimulus pair selections for both the subjects and the agents are mostly negative. This is explained by the fact that the Bayesian model by (Alberts et al., 2016) is symmetric. This means that it is very likely for the negative and positive stimulus pair to have the same expected entropy (as calculated in *step five* of the stimulus selection algorithm). If both of these stimulus pairs have the highest entropy, the algorithm will select the first occurrence, being the negative one.

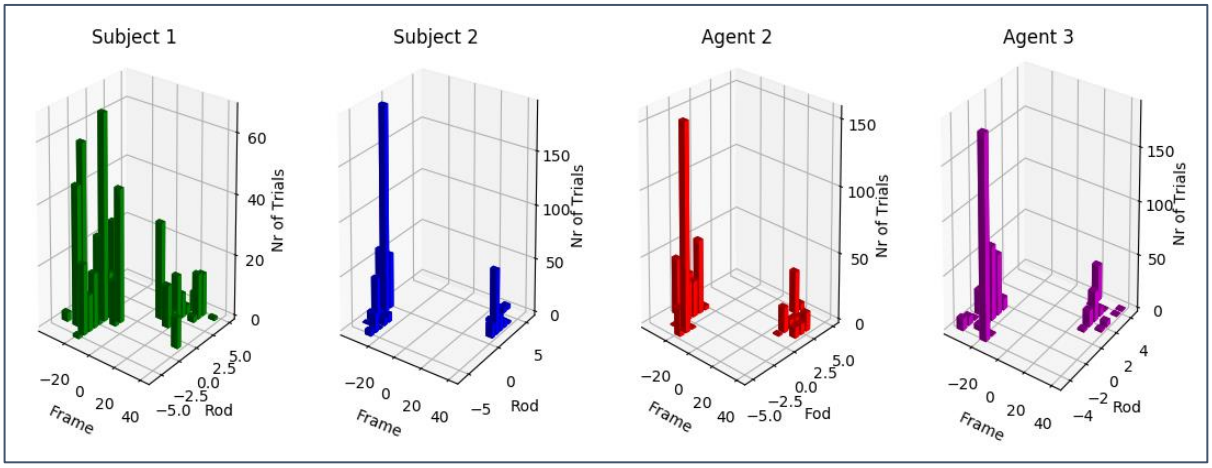

Figure 9: Number of stimulus pair selections in each test. These frequencies correspond to the stimuli selections depicted in the figure above, Figure 8.

To give a better overview of which stimulus pairs were chosen during the entire test, all unique pairs were selected and the frequency of each was calculated. The frequency graphs in *Figure 9*, based on the stimuli selections depicted in *Figure 8*, show that for most of the chosen stimulus pair selections the rod and frame orientations are either both positive or both negative. There are not many cases where one of the stimuli is positive and the other negative, except for the stimulus pair selections for subject 1.

The two stimulus pairs with the highest frequency for both of the depicted agents were exactly the same, only the number of occurrences are slightly different. This shows that, although their ranges and true parameter set were very different, there is some consistency to the responses and therefore the stimulus pair selections.

Discussion

Estimates

The results are very promising: for two out of four parameters there is convergence. κ_{ver} and τ have proven to be easier to estimate than κ_{hor} and κ_{oto}. The convergence of κ_{hor} and κ_{oto} seem to be much more difficult. During some of the trials within tests, there was a rough convergence shape that shifted into more of a diagonal line later on during certain tests. To answer the research questions, thus far it is possible to use adaptive sampling in the rod-and-frame task to estimate two out of four parameters accurately. And the number of trials needed for convergence can definitely be reduced significantly in comparison to 1620 trials as reported by (Alberts et al., 2016).

Dependencies

The results indicated that when the parameter range of κ_{hor} was extended too far (from 22° instead of from 30°), τ was no longer easy to estimate. It in fact shown that the second part of the curve of τ would stay up instead of going down. However, that phenomenon was not always visible in every test with the larger range. There were many tests where the second part of the curve of τ would go down, but much slower in comparison with the smaller range. This indicates a dependency between τ and κ_{hor}.

Future Directions

Since all of the data is easily saved and replicable later on, there are more ways to find the final estimates if not only on the last trial of a test. The results have already shown that for the harder to estimate parameters there was rough convergence at some point. That fact has great implications for exploiting all estimates in a test, rather than only of the last trial. A search algorithm could be designed which will be able to find these best estimates in the trials of a single test, and maybe combine them for the most optimal one.

Looking at the results of the generative agents, there is clearly a possibility to estimate parameters based on adaptive stimulus selection. Using adaptive sampling, it surely does open up the possibility to use the Bayesian model described in (Alberts et al., 2016) as a clinical tool. However, due to the exponential growth of the lookup table with every addition of another data point in the range of one of the parameters, the results are not as accurate as they could be. There needs be more research in finding a way to make the construction of the lookup table and updating the priors much faster before the algorithm could be used as a clinical tool.

In this paper, it was briefly mentioned that the Bayesian model by (Alberts et al., 2016) assumes the responses to the rod-and frame task to be symmetric, meaning that the response to a negative frame orientation and negative rod orientation is the exact opposite of the response to the same frame and rod orientations but then positive. Using this notion of symmetry, the presented stimulus pairs could alternate a lot more between positive and negative orientations in order to keep the stimulus pairs interesting for the subjects. It would also allow the ranges of the orientations to only contain positive values that can be transformed into a negative one if desired for a certain trial. The range would then contain less values as well which would increase the speed of the algorithm. One could also opt to let the length of the range as it was and let it contain more detailed values than it previously was able to, offering more potential

orientations to be presented and possibly giving a more accurate estimation of the parameter set.

The full Bayesian model by (Alberts et al., 2016), including differing head orientations, could make the parameters involved in the perception of upright better identifiable. Having head orientation as an additional stimulus allows the model to separate the contribution of the otoliths and the head-in-space prior. This head-in-space prior represents the prior knowledge that our head is usually positioned upright in space (Alberts et al., 2016). Thus, extending the stimuli frame orientation and rod orientation with head orientation would offer more precise information about the contribution of the model's parameters in verticality perception.

Conclusion

In this thesis, I have presented a way to adaptively select the rod and frame orientations for the rod-and-frame task. Using the responses to update a Bayesian model that explains the responses in terms of four parameters. The results show convergence of two out of four parameters, namely κ_{ver} and τ. There are many parts that could be significantly improved, however, the aim of this thesis has been achieved. Adaptive stimulus selection offers great possibilities for speeding up the convergence process. The number of trials needed to have convergence of κ_{ver} and τ is very small in comparison to the numbers of trials as stated in the literature.

References

Agrawal, Y., Carey, J. P., Della Santina, C. C., Schubert, M. C., & Minor, L. B. (2009). Disorders of balance and vestibular function in US adults: Data from the National Health and Nutrition Examination Survey, 2001-2004. *Archives of Internal Medicine*, *169*(10),

938–944. http://doi.org/10.1001/archinternmed.2009.66

Alberts, B. B. G. T., de Brouwer, A. J., Selen, L. P. J., & Medendorp, W. P. (2016). A Bayesian Account of Visual-Vestibular Interactions in the Rod-and-Frame Task. *ENeuro*, *3*(5). http://doi.org/10.1523/ENEURO.0093-16.2016

Cadieux, M. L., Barnett-Cowan, M., & Shore, D. I. (2010). Crossing the hands is more confusing for females than males. *Experimental Brain Research*, *204*(3), 431–446. http://doi.org/10.1007/s00221-010-2268-5

Gillespie, L. D., Gillespie, W. J., Robertson, M. C., Lamb, S. E., Cumming, R. G., & Rowe, B. H. (2003). Interventions for preventing falls in elderly people. *Cochrane Database of Systematic Reviews*, (4). http://doi.org/10.1002/14651858.CD000340

Kolb, B., Whishaw, I. Q., & Teskey, G. C. (2014). *An Introduction to Brain and Behavior* (4th ed.). New York: Worth Publishers.

Kontsevich, L. L., & Tyler, C. W. (1999). Bayesian adaptive estimation of psychometric slope and threshold. *Vision Research*, *39*(16), 2729–2737. http://doi.org/10.1016/S0042-6989(98)00285-5

Kujala, J. V, & Lukka, T. J. (2006). Bayesian adaptive estimation : The next dimension. *Journal of Mathematical Psychology*, *50*, 369–389. http://doi.org/10.1016/j.jmp.2005.12.005

Lopez, C., Mercier, M. R., Halje, P., & Blanke, O. (2011). Spatiotemporal dynamics of visual vertical judgments: Early and late brain mechanisms as revealed by high-density electrical neuroimaging. *Neuroscience*, *181*, 134–149. http://doi.org/10.1016/j.neuroscience.2011.02.009

Macias, J. D., Massingale, S., & Gerkin, R. D. (2005). Efficacy of vestibular rehabilitation therapy in reducing falls. *Otolaryngology - Head and Neck Surgery*, *133*(3), 323–325. http://doi.org/10.1016/j.otohns.2005.04.024

Magnussen, S., Landrø, N. I., & Johnsen, T. (1985). Visual half-field symmetry in orientation perception. *Perception*, *14*(3), 265–273. http://doi.org/10.1068/p140265

Pillow, J. W., & Park, M. (2016). Adaptive Bayesian methods for closed-loop neurophysiology. In *Closed Loop Neuroscience* (pp. 1–28).

Serrador, J. M., Schlegel, T. T., Black, F. O., & Wood, S. J. (2009). Vestibular effects on cerebral blood flow. *BMC Neuroscience*, *10*(Article ID 119). http://doi.org/10.1186/1471-2202-10-119

Tanaka, K., Abe, C., Awazu, C., & Morita, H. (2009). Vestibular system plays a significant role in arterial pressure control during head-up tilt in young subjects. *Autonomic Neuroscience: Basic and Clinical*, *148*(1–2), 90–96. http://doi.org/10.1016/j.autneu.2009.03.007

Wichmann, F. A., & Hill, N. J. (2003). The psychometric function: I. Fitting, sampling, and goodness of fit. *Perception & Psychophysics*, *63*(8), 1293–1313.

Zalewski, C. K. (2015). Aging of the Human Vestibular System. *Seminars in Hearing*, *36*(3), 175–196. http://doi.org/10.1055/s-0035-1555120

Appendix

A: PSI class

```
1.  import numpy as np
2.  from scipy.stats import vonmises
3.  from GenerativeAgent import GenerativeAgent
4.
5.
6.  class PSIfor:
7.
8.      def __init__(self, kappa_ver, kappa_hor, tau, kappa_oto, theta_frame, theta_rod):
9.          self.kappa_ver=kappa_ver
10.         self.kappa_hor=kappa_hor
11.         self.tau=tau
12.         self.kappa_oto=kappa_oto
13.         self.theta_frame=theta_frame
14.         self.theta_rod=theta_rod
15.         self.stim1_index=-1
16.         self.stim2_index=-1
17.
18.
19.         #dimensions of 2Dstimulus space
20.         self.nframes=len(self.theta_frame)
21.         self.nrods=len(self.theta_rod)
22.
23.         #dimensions of 2D parameter space
24.         nkappa_ver=len(self.kappa_ver)
25.         nkappa_hor=len(self.kappa_hor)
26.         nkappa_oto=len(self.kappa_oto)
27.         ntau=len(self.tau)
28.
29.         #initialize and compute the two kappas
30.         kappa1=np.zeros((nkappa_ver,nkappa_hor,ntau,self.nframes))
31.         kappa2=np.zeros((nkappa_ver,nkappa_hor,ntau,self.nframes))
32.
```

```
        for kv in range(0,nkappa_ver):
            for kh in range(0,nkappa_hor):
                for t in range(0,ntau):
                    kappa1[kv,kh,t,:] = kappa_ver[kv]-(1-
np.cos(np.abs(2*self.theta_frame)))*tau[t]*(kappa_ver[kv]-kappa_hor[kh])
                    kappa2[kv,kh,t,:] = kappa_hor[kh]+(1-np.cos(np.abs(2*self.theta_frame)))*(1-
tau[t])*(kappa_ver[kv]-kappa_hor[kh])

        #initialize cumulitative distribution for every kappa_ver,kappa_hor,tau,sigma_oto combinati
on per frame and rod orientation.
        cdf=np.zeros((nkappa_ver,nkappa_hor,ntau,nkappa_oto,self.nframes,self.nrods))

        for kv in range(0,nkappa_ver):
            for kh in range(0,nkappa_hor):
                for t in range(0,ntau):
                    # for all frames compute the contextual prior (four von mises), the otolith dis
tribution (and the head-in-space prior)
                    for f in range(0,self.nframes):

                        # the context provided by the frame
                        p_frame1 = vonmises.pdf(self.theta_rod-
self.theta_frame[f],kappa1[kv,kh,t,f])
                        p_frame2 = vonmises.pdf(self.theta_rod-np.pi/2-
self.theta_frame[f],kappa2[kv,kh,t,f])
                        p_frame3 = vonmises.pdf(self.theta_rod-np.pi-
self.theta_frame[f],kappa1[kv,kh,t,f])
                        p_frame4 = vonmises.pdf(self.theta_rod-3*np.pi/2-
self.theta_frame[f],kappa2[kv,kh,t,f])

                        p_frame = (p_frame1+p_frame2+p_frame3+p_frame4)/4.0

                        # the otoliths
                        for so in range(0,nkappa_oto):
                            ko=kappa_oto[so]
                            p_oto = vonmises.pdf(theta_rod,ko)
                            # the upright prior
                            #p_hsp = vonmises.pdf(theta_rod,kappa_hsp)
```

```
62.
63.                             # compute the cumulative density of all distributions convolved
64.                             cdf[kv,kh,t,so,f,:]=np.cumsum(np.multiply(p_oto, p_frame))/np.sum(np.mu
    ltiply(p_oto, p_frame))
65.
66.         cdf=np.nan_to_num(cdf)
67.         cdf[cdf==0]=1e-10
68.         cdf[cdf>1.0]=1.0
69.
70.         self.lookup=np.reshape(cdf,(nkappa_ver*nkappa_hor*nkappa_oto*ntau,self.nframes,self.nrods),
    order="F")
71. #         self.lookup=np.load('lookup.npy')
72.         self.prior=np.ones(nkappa_hor*nkappa_ver*nkappa_oto*ntau)/(nkappa_hor*nkappa_ver*nkappa_oto
    *ntau)
73.
74.         self.makeG2()
75.
76.         self.calcNextStim()
77.
78.     def calcNextStim(self):
79.         # Compute posterior
80.         self.paxs = np.zeros([self.lookup.shape[0], self.lookup.shape[1], self.lookup.shape[2]])
81.         self.paxf = np.zeros([self.lookup.shape[0], self.lookup.shape[1], self.lookup.shape[2]])
82.         #h = np.zeros([self.nframes, self.nrods])
83.
84.         self.paxs = np.einsum('i,ijk->ijk', self.prior, self.lookup)
85.         self.paxf = np.einsum('i,ijk->ijk', self.prior, 1.0 - self.lookup)
86.
87.         ps = np.sum(self.paxs,0)
88.         pf = np.sum(self.paxf,0)
89.
90.         self.paxs = np.einsum('jk,ijk->ijk', 1/ps, self.paxs)
91.         self.paxf = np.einsum('jk,ijk->ijk', 1/pf, self.paxf)
92.
93.         # Compute entropy
94.         hs = np.sum(-self.paxs * np.log(self.paxs + 1e-10),0)
95.         hf = np.sum(-self.paxf * np.log(self.paxf + 1e-10),0)
```

```
96.
97.         # Compute expected entropy
98.         h = hs*ps + hf*pf
99.         h = h.flatten('F')
100.
101.        # Find stimulus with smallest expected entropy
102.        idx=np.argmin(h)
103.
104.        frame_f = np.expand_dims(self.theta_frame,axis=1)
105.        frame_f = np.tile(frame_f,(1,self.nrods))
106.        frame_f = frame_f.flatten('F')
107.        rod_f = np.expand_dims(self.theta_rod,axis=0)
108.        rod_f = np.tile(rod_f,(self.nframes,1))
109.        rod_f = rod_f.flatten('F')
110.
111.        # Find stimulus that minimizes expected entropy
112.        self.stim = ([frame_f[idx],rod_f[idx]])
113.        self.stim1_index = np.argmin(np.abs(self.theta_frame - self.stim[0]))
114.        self.stim2_index = np.argmin(np.abs(self.theta_rod - self.stim[1]))
115.
116.    def addData(self,response):
117.        self.stim=None
118.
119.        # Update prior based on response
120.        if response == 1:
121.            self.prior = self.paxs[:,self.stim1_index,self.stim2_index]
122.        elif response == 0:
123.            self.prior = self.paxf[:,self.stim1_index,self.stim2_index]
124.        else:
125.            self.prior = self.prior
126.
127.        self.theta=np.array([self.kappa_ver_g2.flatten('F'),self.kappa_hor_g2.flatten('F'),self.tau
    _g2.flatten('F'),self.kappa_oto_g2.flatten('F')])
128.        self.params=np.matmul(self.theta,self.prior)
129.
130.        self.calcNextStim()
131.
```

```
        return self.params

    def makeG2(self):
        nkappa_ver=len(self.kappa_ver)
        nkappa_hor=len(self.kappa_hor)
        nkappa_oto=len(self.kappa_oto)
        ntau=len(self.tau)

        kappa_ver_g2 = np.expand_dims(self.kappa_ver,axis=1)
        kappa_ver_g2 = np.expand_dims(kappa_ver_g2,axis=2)
        kappa_ver_g2 = np.expand_dims(kappa_ver_g2,axis=3)
        self.kappa_ver_g2 = np.tile(kappa_ver_g2,(1,nkappa_hor,ntau,nkappa_oto))

        kappa_hor_g2 = np.expand_dims(self.kappa_hor,axis=0)
        kappa_hor_g2 = np.expand_dims(kappa_hor_g2,axis=2)
        kappa_hor_g2 = np.expand_dims(kappa_hor_g2,axis=3)
        self.kappa_hor_g2 = np.tile(kappa_hor_g2,(nkappa_ver,1,ntau,nkappa_oto))

        tau_g2 = np.expand_dims(self.tau,axis=0)
        tau_g2 = np.expand_dims(tau_g2,axis=1)
        tau_g2 = np.expand_dims(tau_g2,axis=3)
        self.tau_g2 = np.tile(tau_g2,(nkappa_ver,nkappa_hor,1,nkappa_oto))

        kappa_oto_g2 = np.expand_dims(self.kappa_oto,axis=0)
        kappa_oto_g2 = np.expand_dims(kappa_oto_g2,axis=1)
        kappa_oto_g2 = np.expand_dims(kappa_oto_g2,axis=2)
            self.kappa_oto_g2 = np.tile(kappa_oto_g2,(nkappa_ver,nkappa_hor,ntau,1))
```

B: Generative Agent

```
import numpy as np
from scipy.stats import vonmises
from random import random


class GenerativeAgent:
```

```
7.      def __init__(self, kappa_ver, kappa_hor, tau, kappa_oto, theta_frame, theta_rod):
8.          self.kappa_ver=kappa_ver
9.          self.kappa_hor=kappa_hor
10.         self.tau=tau
11.         self.kappa_oto=kappa_oto
12.         self.theta_frame=theta_frame
13.         self.theta_rod=theta_rod
14.
15.         self.makeProbTable()
16.
17.
18.     def makeProbTable(self):
19.         nFrames = np.size(self.theta_frame,0)
20.         nRods = np.size(self.theta_rod,0)
21.         cdf=np.zeros((nFrames,nRods))
22.
23.         #compute kappas
24.         kappa1 = self.kappa_ver-(1-np.cos(np.abs(2*self.theta_frame)))*self.tau*(self.kappa_ver-
    self.kappa_hor)
25.         kappa2 = self.kappa_hor+(1-np.cos(np.abs(2*self.theta_frame)))*(1-self.tau)*(self.kappa_ver-
    self.kappa_hor)
26.
27.         #for every frame orientation, compute:
28.         for i in range(0,np.size(self.theta_frame,0)):
29.
30.             # the context provided by the frame
31.             P_frame1 = vonmises.pdf(self.theta_rod-self.theta_frame[i],kappa1[i])
32.             P_frame2 = vonmises.pdf(self.theta_rod-np.pi/2-self.theta_frame[i],kappa2[i])
33.             P_frame3 = vonmises.pdf(self.theta_rod-np.pi-self.theta_frame[i],kappa1[i])
34.             P_frame4 = vonmises.pdf(self.theta_rod-3*np.pi/2-self.theta_frame[i],kappa2[i])
35.
36.             P_frame = (P_frame1+P_frame2+P_frame3+P_frame4)/4
37.
38.             # the otoliths
39.             P_oto = vonmises.pdf(self.theta_rod,self.kappa_oto)
40.
41.             # cumulatitve response distribution per frame
```

```
            cdf[i,:]=np.cumsum(np.multiply(P_oto, P_frame))/np.sum(np.multiply(P_oto, P_frame))

        #save cdf as lookup table
        self.prob_table=cdf

    #Determine the response of agent on particular frame and rod combination
    def getResponse(self,stim_frame,stim_rod):

        #Find index of stimulus
        idx_frame=np.where(self.theta_frame==stim_frame)[0]
        idx_rod=np.where(self.theta_rod==stim_rod)[0]

        #lookup probability of responding 1
        PCW=self.prob_table[idx_frame, idx_rod][0]

        #Determine response
        if random()<=PCW:
            return 1
        else:
            return 0
```

C: Rod-and-Frame task experiment

```
# Anouk de Brouwer & Bart Alberts, March 2015. Adapted by Anneloes Ernest, June 2018.
# Sensorimotorlab, Nijmegen
# Rod-and-frame experiment

import numpy, os
from psychopy import visual, core, data, event, gui, sys
from rusocsci import buttonbox

from PSIRiF import PSIfor

boolButtonBox = False

cwd = os.getcwd()

```

```
15. if boolButtonBox:
16.     bb = buttonbox.Buttonbox(port='COM2')
17. 
18. def waitResponse():
19.     global response
20.     key = ['']
21.     while key[0] not in ['z', 'm','escape']:
22.         key = event.waitKeys()
23.         if key[0] == 'z':
24.             response = 0
25.         elif key[0] == 'm':
26.             response = 1
27.         elif key[0] == 'escape':
28.             exit()
29. 
30. # transforms sigma values into kappa values
31. def sig2kap(sig): #in degrees
32.     sig2=numpy.square(sig)
33.     return 3.9945e3/(sig2+0.0226e3)
34. 
35. 
36. # experiment info
37. print('Possible conditions: practice, frame \nPossible locations: v(isionLab), a(noukTest)')
38. #expInfo = {'dayofbirth':'','expDate':data.getDateStr(),'subjectNr':'','nRepetitions':''}
39. expInfo = {'Subject Code':'','Year of Birth':''}
40. 
41. expInfoDlg = gui.DlgFromDict(dictionary=expInfo, title='Experiment details', fixed='expDate')
42. for key in expInfo.keys():
43.     assert not expInfo[key]=='', "Forgot to enter %s!"% key
44. 
45. 
46. # setup info
47. setupInfo = {'monitorResolution':[1920,1080],'monitorFrameRate':60,'monitorSize_cm':[122,67.5],'vi
    ewDistance_cm':57,'screenNr':1}
48. fileDir = cwd
49. fileName = 'RIF_{!s}_{!s}.txt'.format(expInfo['Year of Birth'],expInfo['Subject Code'])
50. 
```

```
51. # open a textfile and write the experiment and setup details to it
52. dataFile = open('{}'.format(fileName),'w')
53. dataFile.write('Rod-and-
    frame experiment by Anneloes Ernest and Luc Selen, Sensorimotorlab Nijmegen \n')
54. dataFile.write('Rod-and-frame task: a rod-and-
    frame stimulus is presented, and the participant has to indicate whether the rod is tilted counter
    clockwise (left arrow key, response=-
    1) or clockwise (right arrow key, response=1) with respect to the gravitational vertical.\n\n')
55.
56. for key,value in expInfo.iteritems():
57.     dataFile.write('{} = {};\n'.format(key,value))
58.
59. for key,value in setupInfo.iteritems():
60.     dataFile.write('{} = {};\n'.format(key,value))
61.
62. dataFile.write('\n') # new line
63. dataFile.write('frameOri rodOri response reactionTime \n')
64. dataFile.close()
65.
66. # create the window to draw in
67. resolution = setupInfo['monitorResolution']
68.
69. #win = visual.Window(monitor="Philips", fullscr=True, units="cm", winType='pyglet', screen=1, colo
    r = 'black')
70. win=visual.Window(monitor="testMonitor", fullscr=True, units="cm", winType='pyglet', screen=0, col
    or = 'black')
71.
72. # stimulus colors
73. frameColor = (-.8,-.8,-.8)
74. rodColor = (-.8,-.8,-.8)
75.
76. #frameColor = (0.0,0.0,0.0)
77. #rodColor = (0.0,0.0,0.0)
78.
79.
80. frame_size = 15
81. line_length = 6
```

```
82.
83.  # create a rod-and-frame stimulus
84.  frame = visual.Rect(win,width=frame_size,height=frame_size,lineColor=frameColor,lineColorSpace='rg
     b',lineWidth=1,units = 'cm')
85.  rod = visual.Line(win,start=(0, -
     line_length),end=(0,line_length),lineColor=rodColor,lineColorSpace='rgb',lineWidth=1,units = 'cm')


86.
87.  vert_center = -2
88.  horz_center = +15.0
89.
90.
91.  frame.pos = (horz_center, vert_center)
92.  rod.pos = (horz_center, vert_center)
93.
94.  # stimulus orientations
95.  # frameOri = range(-45,45,5)
96.  frameOri = numpy.linspace(-45,45,11)
97.  rodOri = numpy.linspace(-10,10,30)
98.
99.
100. # wait for a key press to start the experiment
101.
102. startText1 = visual.TextStim(win,text='Press the left arrow if you believe that\n the line is tilt
     ed counterclockwise.\nPress the right arrow if you believe the that\n the line is tilted clockwise
     .',pos=(horz_center,vert_center+2),alignHoriz='center',color=rodColor,wrapWidth=25, height =1)
103. startText2 = visual.TextStim(win,text='Ready?\n Press a button to start.',alignHoriz ='center',pos
     =(horz_center,vert_center-5),color=rodColor, height = 1)
104.
105.
106. startText1.draw()
107. startText2.draw()
108. win.flip()
109.
110.
111. if boolButtonBox:
112.     b = bb.waitButtons()
```

```
113. else:
114.     event.waitKeys(maxWait=60)
115. core.wait(1.0)
116.
117. # experiment: present stimulus and wait for keyboard response
118. n = 0
119. t0=core.getTime
120.
121. #init parameter ranges
122. #lookup9
123. kappa_oto = numpy.linspace(sig2kap(1.4),sig2kap(3.0),10)
124. kappa_ver = numpy.linspace(sig2kap(2.5),sig2kap(7.5),15)
125. kappa_hor = numpy.linspace(sig2kap(22),sig2kap(80),15)
126. tau = numpy.linspace(0.6,1.0,10);
127.
128. #init algorithm
129. psi=PSIfor(kappa_ver,kappa_hor,tau,kappa_oto,frameOri,rodOri)
130.
131. for trial in range(0,500):
132.     while psi.stim == None:
133.         pass
134.
135.     # set rod and frame orientations
136.     stim_frame=psi.stim[0]
137.     stim_rod=psi.stim[1]
138.
139.     frame.setOri(stim_frame)
140.     rod.setOri(stim_rod)
141.
142.     # draw and flip only the frame
143.     frame.draw()
144.     win.flip()
145.     core.wait(.25) # time that the frame is visible
146.
147.     # add the rod for 1 frame
148.     frame.draw()
149.     rod.draw()
```

```
150.     win.flip()
151.     # add the rod for a 2nd frame
152.     frame.draw()
153.     rod.draw()
154.     win.flip()
155.
156.     # draw and flip only the frame
157.     frame.draw()
158.
159.     win.flip()
160.     timer1 = core.getTime()
161.
162.     # wait for a key press and then remove the frame
163.     if boolButtonBox:
164.         b = bb.waitButtons()
165.         key = b[0]
166.     else:
167.         waitResponse()
168.
169.     #roughRT = keypress[0][1] - timer1
170.     if 'escape' in event.getKeys():
171.         exit()
172.     else:
173.         # get response from buttonbox
174.         if boolButtonBox:
175.             if 'A' == key:
176.                 response = 0
177.             elif 'B' == key:
178.                 response = 1
179.             else:
180.                 response = 99
181.         roughRT = core.getTime()-timer1
182.         #write data to text file
183.         with open('{}'.format(fileName),'a') as dataFile:
184.             dataFile.write('{:1.1f} {:1.1f} {:1.1f} {:1.2f}\n'.format(stim_frame,stim_rod,response
    ,roughRT))
185.         win.flip()
```

```
186.         #update priors
187.         params = psi.addData(response)
188.         print trial, stim_frame, stim_rod, response
189.         psi.print_expected_value()
190.         core.wait(0.2) # intertrial interval, black screen
191.
192.
193. ExpDur = core.getTime()-t0
194. event.waitKeys(maxWait=60)
195. # close
196. win.close()
197. core.quit()
```