Effect of gaze-contingent windows on visual sampling and using visual sampling models to predict gaze behavior

Student name: Ahmed Bakay
Student number: 4534409
Supervisors: Yke Bauke Eisma, Joost de Winter

Study: M.Sc. Robotics
Report Type: M.Sc. Thesis
Period: 10-2021 until 08-2022

Effect of gaze-contingent windows on visual sampling and using visual sampling models to predict gaze behavior

Abstract

Human operators who are tasked with monitoring automation systems may experience a high visual demand to process the information streams from these systems. The visual sampling behavior of human operators can be described using mathematical models. These models can help designers improve environments where multiple signals are present for human operators to monitor, to a configuration that can be processed properly.

This study consisted of two parts. The first part investigated how peripheral vision plays a role in visual sampling behavior and task performance, specifically in the experimental eye-tracking setup presented in Eisma et al. (2018). In this setup, participants were instructed to monitor a bank of six dials, of which each dial pointer had a threshold indicator, and press a response key whenever a dial pointer crossed the threshold indicator. In the second part, the sampling models as presented by Senders (1983) are implemented to predict sampling trajectories. The sampling characteristics that resulted from the predictions were then evaluated.

The results of the experiments show that peripheral vision plays a role in visual sampling and task performance. More specifically, sampling behavior is more evenly distributed among dials, and task performance is lower when peripheral vision is absent. The main attractor in the peripheral vision is shown to be the pointer speed. Moreover, the learning effect presented in Eisma et al. (2018) is not apparent when peripheral vision is absent.

The results of the predictions showcase the sampling behavior characteristics, some of which show similarities with the results from the experimental data.

Keywords

Human sampling behavior, human operators, dial characteristics, Senders models, general prediction (distribution), prediction in the time domain, visual attention, gaze distribution, sampling, prediction, Senders, contingency, contingent

Introduction

Relevance

With the continuous development of new technologies and their implementations in our lives, the increase in information streams for humans is unavoidable. Some of those developments are accessible internet-of-things (IoT) infrastructures (Alam, 2021) and wireless cameras that can be placed at any desired place (Spachos et al., 2019). These new developments may provide valuable information to human operators, who are sometimes tasked with monitoring them. This is especially true for human operators in industrial settings such as aviation, nuclear reactors, and production plants, where monitoring information streams from systems that are in operation can be crucial at times (Peterson, 2009). In these settings, the human operator is tasked with monitoring multiple information sources that will vary in value or state depending on the real-world equivalent that they are representing (Zhang et al., 2020).

However, with these new sources of information, the human operator must also process the information visually and cognitively and act in case anything goes wrong. This can lead to a phenomenon called information overload (Dadashi et al., 2016), in which case the human operator cannot process all the information sources at the same time, such that the human operator can take corrective measures in case something goes wrong (Perrow, 1981). To combat information overload in settings where multiple information streams are present, it is important to understand how the sampling behavior of human operators is determined (Sharma et al., 2016). With the understanding of how human operators sample, models can be formulated (Wickens et al., 2001). Such models can help assess sampling behavior and can help to create optimal circumstances for the operator to get an accurate picture of all the information streams (Senders, 1983).

Background

In his doctoral thesis titled "Visual Sampling Processes" (Senders, 1983), Senders presented various models and findings on the relationship between information stream characteristics and sampling behavior. There are multiple models described which are used to predict sampling behavior variables such as fixation duration, fixation frequency, sampling probabilities, attentional demand, and the interval between fixations. These variables are calculated based on different strategies a human operator is assumed to follow. Three of those models are the Periodic Sampling Model (PSM), Random Constrained Sampling Model (RCM), and Conditional Sampling Model (CSM).

- PSM assumes a strategy in which the operator tries to sample periodically and at a frequency of twice the frequency bandwidth of the pointer signal.
- RCM assumes a strategy in which the operator samples randomly based on a certain probability, an interval time between fixations, and a sampling duration.
- CSM assumes a strategy of the operator keeping a memory of pointer angles since the last time they were sampled and estimating when they will reach their threshold value.

These models have been used by Senders (1983) to predict the sampling behavior of human operators and to evaluate those results in four experiments in which a group of five high school students were assigned the task of viewing a bank of six dials with each a different bandwidth (0.03, 0.05, 0.12, 0.20, 0.32, and 0.48 Hz) and pressing a response button whenever one of the dials passed its threshold. With the resulting sampling behavior data of participants, the predictions of the models could be evaluated. The results showed, according to Senders (1983), that there was a difference between the predictions and the participants' sampling behavior. However, no exact data about simulations is presented in his doctoral thesis (Senders, 1983). Because no proper simulation for each model was performed to give insight into the extent to which there were differences and similarities, this should be investigated further.

The six-dial bandwidth experiment of Senders (1983) was replicated by Eisma et al. (2018). In this study, the characteristics of sampling behavior and participant performance were analyzed and compared with the results of the six-dial experiments of Senders (1983). The visual sampling characteristics found in the replicated experiment were similar to the findings of the original experiments. In addition to the replicated experiment, Eisma et al. (2020) reviewed the sampling models (PSM, RCM, and CSM) of Senders using computer simulations and illustrative graphs.

A notable assumption for the models and experiments is that it is assumed that the participants will sample based on the queuing model in the three models that he presented (Senders, 1983). When multiple signals ask for simultaneous attention, the human operator will put the extra signals in a queue. After the operator samples one signal, it will move to the next value in the queue. This model does not consider the peripheral view of human operators, and Senders mentioned that “instruments are designed by designers who look at what they are drawing, thus designing them for foveal viewing” (Senders, 1983, p.14). Peripheral viewing is actually important in a broad range of tasks (Rosenholtz, 2016) and in the experiment of Eisma et al. (2018), it also became clear that peripheral vision plays a role in visual sampling. In Eisma et al. (2018), a relationship between pointer speed and sampling was observed. Namely, that a higher pointer speed attracts more attention, which is not included in the sampling strategies presented by Senders (1983). However, it is not clear to what extent that is the case with the experiments (Senders, 1983; Eisma et al., 2018) that were conducted.

Goal

It can be noted that it is vital to know to what extent peripheral viewing affects sampling behavior in the setup of Eisma et al. (2018) and whether the sampling models of Senders (1983) can be used to predict the visual sampling behavior of participants. This study aimed to quantify the effect of a gaze-contingent window on sampling behavior and task performance. A gaze-contingent window was added to exclude peripheral sight. This prevents a participant from seeing salient factors that are present in the peripheral area, as found by Eisma et al. (2018). Besides that, this study aimed to quantify the sampling characteristics of the predictions of sampling models by Senders (1983). The predictions will be called "sampling trajectories." A sampling trajectory is a list of numbers that correspond to an area of interest (AOI) per frame. Note that this trajectory consists of discrete values (AOIs).

The aim of this study was divided into two research objectives: experimenting and predicting, of which both have research questions.

- An experiment was conducted to gather sampling behavior data using the same setup as used by Eisma et al. (2018), which was adjusted to allow the addition of a setup with a gaze-contingent window.
    - How do the results of the original setup relate to the experiment conducted by Eisma et al. (2018)?
    - What is the impact of a gaze-contingent window on sampling behavior and task performance?
- Sampling trajectory predictions will be made using pragmatic models, which are based on the (change of) pointer angles and three different sampling models of Senders (1983), namely PSM, RCM, and CSM. Predictions will be made without using previously known visual sampling data

(offline, as opposed to online, where based on a previous sampling point, the next point is predicted).

- What kind of sampling characteristics (glance rate, percentage of time spent on the area of interest, mean glance duration) do the predictions result in?
- How do these sampling characteristics compare to the sampling characteristics seen in experimental data?

Methods

The two different objectives each have their own method. First, an experiment was conducted to gather eye movement data from participants, with which a data analysis was performed. After the experiments and analysis took place, the sampling models of Senders (1983) were used to predict sampling trajectories.

Experiment

Participants

The participant group consisted of 33 students at the Technical University of Delft. The mean age was 23.88 with a standard deviation of 2.37. Three participants (numbers 8, 11 and 18) wore glasses during the experiment. All participants have signed a written consent form (Appendix A). While this group is not diverse in terms of age and study background, the participants of previous experiments were students as well (Senders, 1983; Eisma et al., 2018). The timestamps within which the experiments took place varied between 10:00 am and 6:15 pm.

Setup

Eye tracking

The eye movements of participants were recorded by an EyeLink 1000 Plus eye tracker (SR Research, n.d.). This eye tracker measured eye movements at 2000 Hz. The eye tracker was positioned between a chin rest (allowing for a steady head while participating in trials) and a monitor (24-inch BenQ XL2420T-B, 1920x1080px, 91.79PPI). The setup is shown together with a participant in Fig. 1.

Videos

Seven videos were made by Eisma et al. (2018), which contained six dials with a moving pointer and a stationary threshold (Fig. 2). Each video represents another effort level, which corresponds to a certain configuration of dials (Table 1). The dials in each video have a different bandwidth and threshold. The bandwidths for the dials are: 0.03, 0.05, 0.12, 0.20, 0.32, and 0.48 Hz. Further on, the dials will be referred to by numbers as well: 1, 2, 3, 4, 5, and 6 respectively. The thresholds and pointer angles were randomly generated for each video. The videos were 4500 frames (90 s) long (videos at 50 fps). To accommodate an extra setup with foveal viewing and to not overload participants, the last 30 seconds of the videos were removed, so that the videos were shortened to 3000 frames (60 s at 50 fps).

Viewing modes

This study introduces a viewing mode to analyze the effect of a gaze-contingent window. Participants were presented with two different setups. One setup in which the whole screen is visible (Fig. 2), and another setup in which only a circular area of 500×500 px of the video is visible at the point where participants are looking (Fig. 3). The setup where all the dials are shown is called FV. The setup where only the foveal area is shown is called GCV. The foveal viewing area is determined to have a size of 500×500 px (138.35x138.35 mm). The foveal viewing area follows the gaze of the participants.

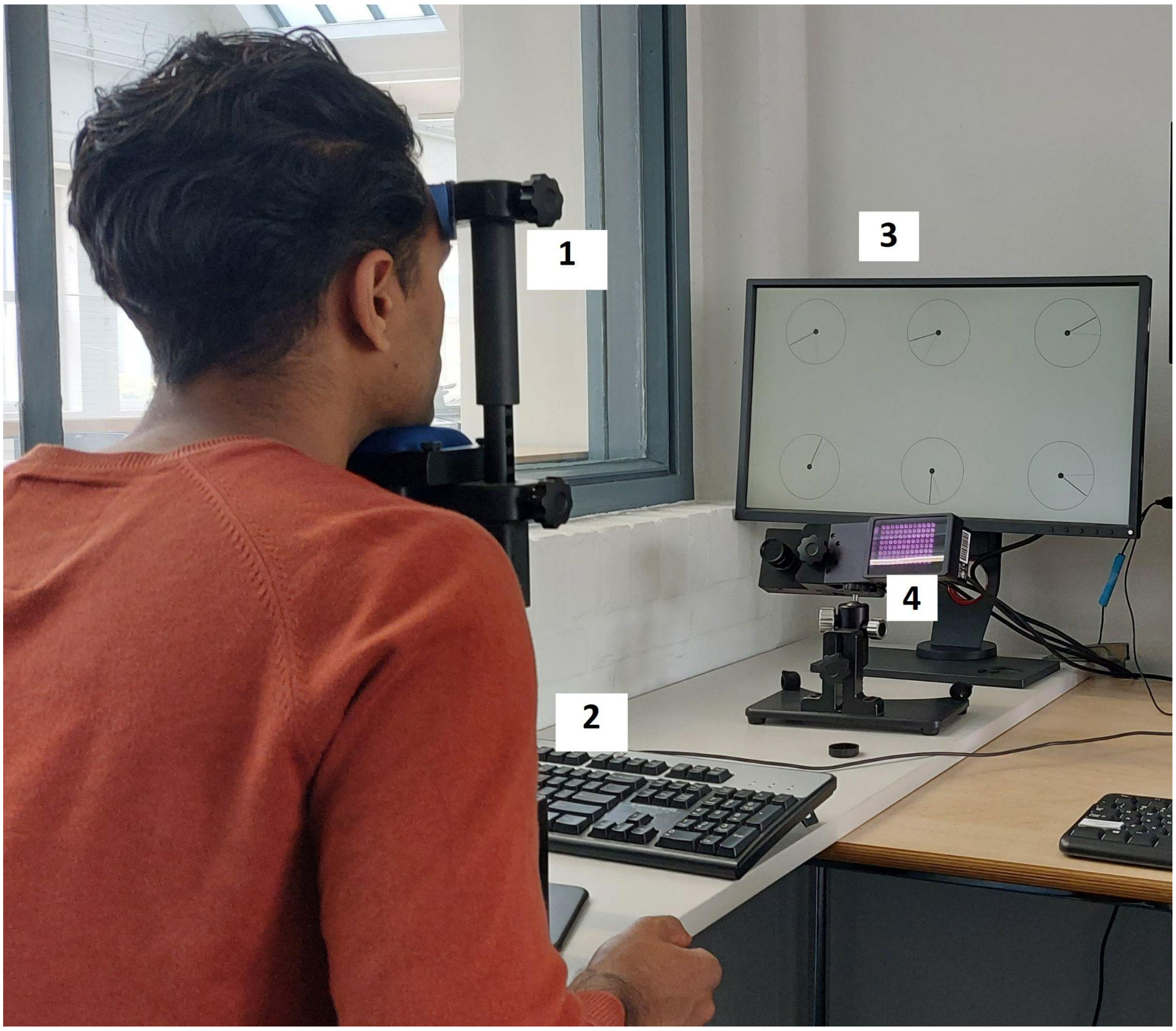


Fig. 1: The experimental setup, which includes the chin rest (1), keyboard (2), the display (3), and eye tracking camera and infrared illuminator (4), is shown. A participant is performing the experiment. His head rests on the chin rest while his left hand (not shown) is hovering on the spacebar. He is looking at the display to find the moments where the dial pointer crosses its threshold and presses the spacebar. Meanwhile, the eye tracker registers his eye movements.

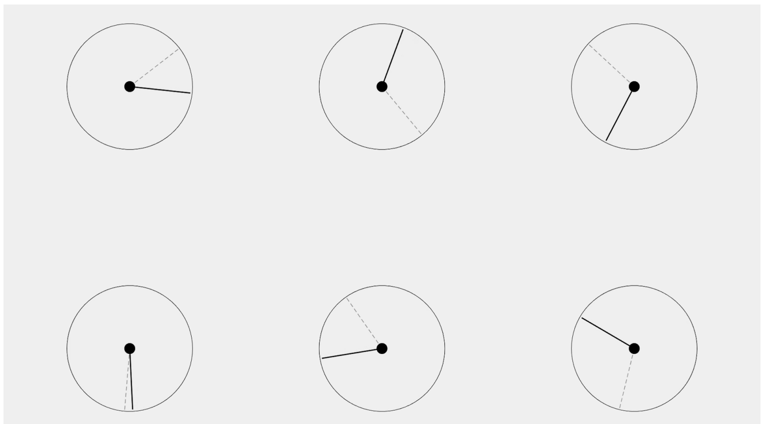


Fig. 2: Full view. Content of videos, displaying six dials with different bandwidths and threshold positions.

Table 1: Effort levels with corresponding dial bandwidth (Hz) per position in video display. Taken from Eisma et al. (2018, p. 530).

Video effort level	**Top left**	**Top middle**	**Top right**	**Bottom left**	**Bottom middle**	**Bottom right**	**Effort level**
1	0.12	0.48	0.05	0.20	0.32	0.03	3422
2	0.20	0.48	0.03	0.32	0.05	0.12	3686
3	0.03	0.12	0.20	0.32	0.48	0.05	3896
4	0.32	0.12	0.05	0.48	0.03	0.20	4097
5	0.48	0.05	0.03	0.12	0.20	0.32	4314
6	0.12	0.32	0.05	0.20	0.03	0.48	4532
7	0.32	0.03	0.20	0.12	0.05	0.48	4969

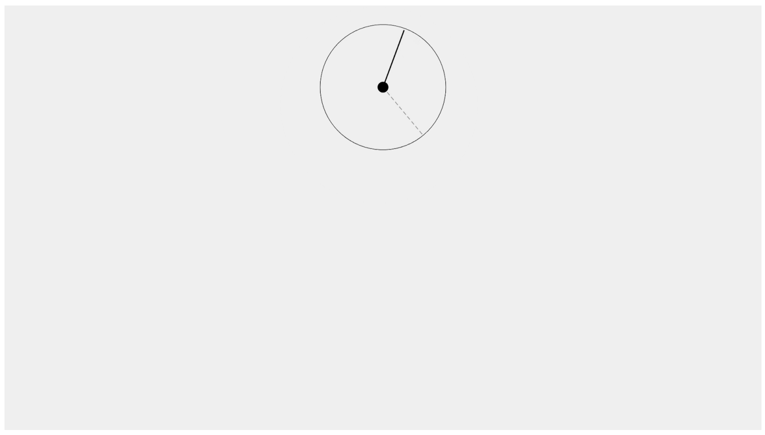


Fig. 3: Gaze Contingent View. Example state of the setup where only the foveal view is shown to the participants (assuming the participant is looking at the top middle of the screen). When the fixation point moves, the contingent window moves to the appropriate position. The rest of the dials are still active but are not visible to the participant.

Experimental procedure

The interface of the experiment was built using Experiment Builder (SR Research, 2020). The interface consisted of an instruction for the participants about the experiment. The participants were instructed by an instruction screen to detect threshold crossings (the moment when a dial pointer crosses the threshold line) and press the spacebar on the keyboard whenever they noticed a threshold crossing.

Participants were told orally that the dial bandwidth and its direction or speed were irrelevant for determining whether the participant had to press the spacebar key. Moreover, it was said orally that if the participant detected multiple threshold crossings at once, they could press the spacebar multiple times. After the instructions are read, the eye tracker is calibrated. Following that, a training session for FV and GCV begins. The trials in the training session have a duration of 20 seconds each. In the training session, only one video is shown (effort level 1). It was possible to retry the training session in case the participant asked for it, and it was proposed to retry in case the experiment supervisor assumed that it was needed. It was assumed that a retry was needed whenever the participant showed visual sampling behavior like staring at one dial or not pressing the spacebar.

After the training session was passed successfully, the recorded trials start. In the recorded trials, participants were shown the 7 videos for both setups (Fig. 2, Fig. 3). This results in each participant going through 14 trials. The trials are separated into two blocks, one for each setup. Participants first went through FV or GCV. This is randomized (order attributions can be seen in Table 2). In those setups, the order of videos is randomized as well (order attributions can be seen in Table 3). Each trial begins with a drift correction to account for any minor movements made by a participant during or between trials.

Table 2: An overview of which participants started with FV, and which participants started with GCV.

Setup order	**Participants**
Started with FV	11,12,13,15,16,17,2,22,26,27,3,4,5,6,7,8,1,30,32
Started with GCV	10,14,18,19,20,21,23,24,25,28,9,29,31,33

Table 3: An overview of the order of videos per participant. The top row indicates the participant number; the left column indicates the order (from 1 to 7). The order of videos was the same for both setups.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**
1	2	7	5	1	2	2	3	1	1	6	2	3	7	2	1	5	4	4	6	4	2	6	4	6	5	4	6	6	4	4	7	6	5
2	5	1	4	6	7	3	4	7	2	1	1	4	5	4	3	4	1	5	7	1	5	7	2	5	7	7	1	4	1	7	5	2	3
3	6	5	2	5	1	4	2	3	7	2	3	7	3	7	6	7	6	1	4	5	1	2	5	1	6	5	7	7	5	6	2	7	2
4	4	2	1	4	6	6	1	6	4	3	6	6	4	6	5	3	5	7	2	7	4	3	1	4	2	2	2	3	2	5	6	5	7
5	1	3	7	3	4	7	5	2	3	5	7	1	6	3	7	1	2	6	1	3	6	5	6	7	4	1	3	5	6	3	1	4	4
6	7	4	3	2	3	1	6	4	5	7	4	2	2	1	2	6	3	3	3	6	7	4	7	2	1	6	4	2	3	1	3	3	1
7	3	6	6	7	5	5	7	5	6	4	5	5	1	5	4	2	7	2	5	2	3	1	3	3	3	3	5	1	7	2	4	1	6

Gaze data processing

The eye movement dataset of the trials contained the average (of left and right eye) x and y coordinates per participant, setup, video, and frame combination. Participants may, at some point, blink or glance away from the screen, which results in missing data points. Those were filled in by linearly interpolating between the previous available gaze values and the next available gaze values.

In this study, the dial that is watched is of importance for analyzing sampling behavior instead of x and y coordinates. Therefore, the x and y coordinates were converted into dial numbers. This was done by converting the x and y values to area of interests, which are rectangle boundaries of 420×420 px surrounding the center of a dial, as was done in Eisma et al. (2018). This area is called the "area of interest" (AOI). The AOI numbers correspond directly to dial numbers. The videos were played at 50 fps, and the eye tracker recorded at 2000 Hz. This means that each frame number occurs multiple times in the dataset. To reduce the data to a single data point per frame, the median of AOI's was selected for each frame. This results in a dataset that contains a single row of data for each participant, setup, video, and frame combination and its corresponding AOI that is being watched.

Dial data rounding and extending

The base data of the dials consists of frame numbers and pointer angles (in radians) relative to the stationary threshold line for each effort level. The pointer angles are positive for clockwise angles relative to the threshold and negative for counterclockwise angles relative to the threshold. To ease interpretation and calculation, these values were converted to degrees. This data is extended by calculating the speed, direction, and time-to-crossing of the threshold.
The speed is calculated by subtracting the pointer angle of the previous frame from the current frame and multiplying it by 50 (Eq. 1). The direction is determined by the speed and angle. When the speed is positive and the pointer angle is negative, or the speed is negative and the pointer angle is positive, the direction is defined as 1 (meaning that the pointer is moving towards the threshold), and otherwise the direction is defined as 0 (meaning that the pointer is moving away from the threshold). The time-to-crossing values are calculated by calculating the distance towards the threshold divided by the speed of the pointer (Eq. 2).

$$v = (angle_n - angle_{n-1}) * 50 \qquad (1)$$

$$TTC = angle_n / v \qquad (2)$$

To aggregate the data in certain increments, the angle and speed values were rounded off to the nearest 5 deg or deg/s, respectively. The time-to-crossing values have been rounded off to the nearest 0.5 second.

The script used to do the data processing is presented in Appendix C.

Data analysis

This section shows the different ways to analyze the data. The script used is shown in Appendix J.

Glance rate (Hz)

The glance rate is defined as the number of times per second that a certain AOI is being fixated on by participants (Eq. 3). Eye movements with a duration of less than 40 ms were excluded, as was chosen in Eisma et al. (2018) as well.

$$Glance\ rate(AOI) = \frac{Total\ number\ of\ frames\ participants\ fixated\ on\ AOI}{Total\ number\ of\ frames\ that\ participants\ have\ sampled\ over\ all\ AOI's} \qquad (3)$$

Mean glance duration (s)

The mean glance duration is defined as the average time that a fixation lasts on a certain AOI (Eq. 4). Data points with eye movements of less than 40 ms were excluded in this metric as well.

$$Mean\ glance\ duration(AOI) = \frac{Total\ number\ of\ frames\ that\ an\ AOI\ is\ sampled}{Fixations\ on\ AOI} \qquad (4)$$

Percent AOI (% of time)

Percentage time on AOI is defined as the amount of time that an AOI is sampled divided by the total amount of time that participants have sampled (Eq. 5).

$$Percent\ AOI(AOI) = \frac{Total\ number\ of\ frames\ sampled\ on\ AOI}{Total\ frames\ sampled} * 100 \qquad (5)$$

The definition changes a bit when percentage time on AOI is mentioned in terms of dial characteristics (pointer angle, pointer speed, and time-to-crossing). The percentage of time on AOI is then defined as the number of frames that a certain AOI with a certain characteristic value is sampled divided by the total amount of time that a certain characteristic value of an AOI has taken place (Eq. 6).

A fictional example: dial 2, the AOI, has a pointer angle of 20 degrees at 10 frames in video 1. Participant 5 has sampled dial 2 for 4 frames in video 1 while it had a pointer angle of 20 degrees. This means that the percentage of time spent on area of interest will be 4/10×100 = 40%.

$$Percent\ AOI(AOI) = \frac{Total\ frames\ sampled\ on\ AOI\ while\ it\ had\ a\ specific\ dial\ characteristic}{Total\ frames\ where\ AOI\ had\ a\ specific\ dial\ characteristic} * 100 \qquad (6)$$

Task performance (%)

Task performance is defined as the number of threshold crossings that have been acted upon (spacebar key pressed) divided by the total number of threshold crossings (Eq. 7).

$$Task\ performance = \frac{Crossings\ that\ participants\ have\ acted\ upon\ by\ pressing\ the\ spacebar}{Total\ number\ of\ crossings} * 100 \qquad (7)$$

Predictions

General

Using six different models, sampling trajectory predictions were made. A sampling trajectory is a list containing 3000 values (corresponding to 3000 frames, of which sampling trajectories of participants consist) that describe the AOI that is predicted. Three prediction models are coupled under the term "pragmatic", as they are predicted based on the base values (the pointer angles) of the dials and nothing else. An important note here is that previously known data (pointer angles and their change over time) is used. The PSM, RCM, and CSM models are described separately. Depending on the number of parameters, multiple predictions were made with the PSM, RCM and CSM models, which are described in the next section. The Python setup used for prediction is shown in Appendix B. The main script used for running the predictions is shown in Appendix D. In appendix I custom functions used for the prediction scripts are shown.

Parameters

The parameter ranges are selected based on a sensitivity analysis with a wider range of values and on the experimental data. This is done to make sure that no values are used that go beyond the abilities of participants or are far away from what participants are doing in the experiments. The ranges are used to create multiple combinations of parameter values, as seen in Appendix K. The Saltelli method (Campolongo et al., 2011; Herman et al., 2021) was used to generate the different combinations of parameter values based on input ranges that were given. This method used Sobol Sequences (Sobol', 1967) to create uniformly distributed parameter spaces. The ranges that were selected are as follows:

- Sampling frequency: 1.5-2.5 Hz. This range is chosen based on the formulas of Senders (1983). In those formulas, the value of 2 Hz is used. To allow for differences between participants (as no participant is the same), a 0.5 Hz margin is added.
- Sampling duration: 0.2-0.6 seconds. This range was chosen based on the sampling duration that participants demonstrated, as seen in Fig. 9. A margin has been added to allow for some experimenting; a range between 0.2 and 0.6 seconds is assumed to be reasonable.
- Wait time: 1–7 frames. The wait time is used to put an AOI in the queue to be watched. It is assumed that the human operator knows that there is another AOI that needs to be sampled. Therefore, only a small number of frames are selected. From the sensitivity analysis, it was found that the variation in the wait time did not significantly impact the performance of predictions.
- Interval correction: 0.5-1.5. This value is used to multiply the interval calculation that is calculated in the RCM. The range is selected based on a sensitivity analysis, from which the best predicted chance is achieved with a value between 0.7 and 1.3.

Pragmatic

The pragmatic prediction model is used to predict sampling behavior using the base values that are available from the information sources. The base values of the information sources consist of frame number and pointer angle. Based on the change of the pointer angle over the frames, the speed, direction, and time-to-crossing were derived. The models of pointer angle, pointer speed, and time-to-crossing are selected based on the analysis of Eisma et al. (2018, p. 536), from which it can be seen that there is a relation between the percentage of time on area of interest and the pointer angle, pointer speed, and time-to-crossing. Therefore, it was assumed that using these properties to predict trajectories might result in adequate results. The pseudocode for each pragmatic model is shown in Fig. 4 and the Python script is displayed in Appendix E.

### *Pointer angle (da)*

If the pointer angle is closer to the threshold, the probability of crossing it is also assumed to be higher. The dial that is the closest to the threshold was picked as a prediction for a certain frame.

### *Pointer speed (sp)*

Dials with a higher absolute speed will reach the threshold quicker than those with a lower absolute speed. The dial with the highest absolute speed was picked as the prediction for a certain frame.

### *Time-to-crossing (ttc)*

The time-to-crossing value is derived from the distance to the threshold and the speed of the dial. The lower the time-to-crossing, the higher the probability that the dial will cross the threshold in the near future. The dial that has the lowest absolute time-to-crossing was picked as a prediction for a certain frame.

```
# Time to crossing
ttc:
    for each frame in dial data
        select the dial number with the lowest absolute time to crossing

# Pointer speed
sp:
    for each frame in dial data
        select the dial number with the highest absolute speed

# Pointer angle
da:
    for each frame in dial data
        select the dial number with the lowest absolute pointer angle
```

Fig. 4: Pseudocode for pragmatic models

Periodic Sampling Model

The Periodic Sampling Model (PSM) can be used to estimate the attentional demand of information sources based on sampling duration and signal bandwidth. The human operator is assumed to have an attentional demand capacity of 1. If the total attentional demand required by the information sources exceeds 1, it means that there is a possibility of information loss. The attentional demand of a dial can be used as a measure of how likely it is that a dial will be sampled.

The attentional demand of a dial can be calculated using Eq. 8. (where i = dial number, W = bandwidth in Hz, D = sampling duration in seconds).

$$T_i = 2W_i \times D_i \qquad (8)$$

The total attentional demand can then be calculated using Eq. 9 (where *i* = dial number, *m* = total dials).

$$T_{sum} = \Sigma \sum_{i=1}^{m} T_i \qquad (9)$$

The 2W part of the attentional demand formula is the Nyquist rate, based on the Nyquist-Shannon theorem described by Shannon (1949), which states that, to observe all the information emitted by a signal, the observer has to sample twice the signal bandwidth. While this is the case for an ideal observer, Senders (1964) found that in his experiments the participants had sampled at a frequency of 2.44W. This leads to handling the number 2 in the attentional demand formula as a parameter which can be adjusted.

The sampling duration of the dials is assumed to be unknown, thus D is a parameter in this case. In Senders’s research (Senders, 1983), it is assumed that D is the same for all dials. That is because the required precision to read the dials is the same for all.

Parameters: Sampling frequency, sampling duration.

Trajectory

The attentional demand was used to choose the dial to predict. The attentional demands of all dials were normalized, which led to the probabilities of looking at a certain dial. If an observer has a capacity of 1 attentional demand and a dial has an attentional demand of 0.6, it means that the observer has to attribute 60% of its time to that dial.

Using the probabilities (normalized attentional demands), a dial was picked for a frame. Starting from that frame, the same dial is selected for a selected number of frames (which is based on the sampling duration). After that, a new dial was selected for a selected number of frames, and this process was repeated until there were 3000 predicted values. The pseudocode for this process can be seen in Fig. 5 and the Python script is displayed in Appendix F.

This process was iterated multiple times, with multiple different combinations of parameter values, as indicated before. The combinations of the parameters can be seen in Appendix K. 96 predictions were made in total.

```
# psm
selected values = a list of randomly selected dial numbers based on the normalized attentional
demands with a length of the video length
predicted trajectory = []

for each value in selected values
    append the value a number of times (based on sampling duration) to the predicted trajectory
    if length of predicted trajectory is equal to video length
        break
```

Fig. 5: Pseudocode for the periodic sampling model

Random Constrained Sampling model

According to Senders (1983), PSM is not feasible when there is more than one dial. As a response to that, the Random Constrained Sampling Model (RCM) was introduced. As the name implies, this model assumes that the human operator samples the dials randomly instead of periodically. While the attentional demands that resulted from the PSM were normalized into probabilities, the formula to calculate those probabilities differed. Besides, RCM also provides information about the interval between two fixations and a correction for the sampling duration.

The probability of sampling a dial randomly is given by Eq. 10 (where i = dial number, P = probability, W = bandwidth in Hz).

$$P_i = \frac{W_i}{\sum_{i=1}^{m} W_i} \qquad (10)$$

The time interval between two fixations on any particular dial, which can be simplified due to the assumption that sampling duration is the same for all dials, is shown in Eq. 11 (where *i* = dial number, *D* = sampling duration, *p* = probability).

$$\mu_{interval,i} = 1Dp_i(1 - p_i)^0 + 2Dp_i(1 - p_i)^1 + 3Dp_i(1 - p_i)^2 + ... = \frac{D}{p_i} \qquad (11)$$

This indicates the time it will roughly take until a new sample is taken of the same dial. Because of the impact that it has on the sampling process, a correction factor for the interval is also introduced. That number will be used to multiply the interval, thus making it smaller or larger.

Because the sampling duration in reality can be higher due to repeated sampling of the same dial, the corrected sampling duration is given in Eq. 12 (where, i = dial number, D = sampling duration, p = probability).

$$\overline{D_0} = D\frac{1}{1-p_i} \qquad (12)$$

In this model, the sampling duration and interval correction are parameters that are unknown prior to the experiment and can be varied to find out what value best fits reality.

Parameters: Sampling duration, interval correction.

Trajectory

The probability of sampling (P_i) was used to choose the dial to predict for a certain frame. The duration of sampling was determined by D_0, which is the corrected sampling duration. After the dial was chosen, another dial had to be chosen based on P_i(which is also sampled for a duration of D_0). This process was repeated until there were predictions for all frames. So far, this process is similar to the PSM.
The difference with this process is that there is a soft requirement that the dial be sampled again after $\mu_{interval,i}$ seconds. This means that the sampling will still be done randomly, but if there is a dial in the queue for which the interval has expired (meaning it needs to be sampled), that dial will be sampled next. The pseudocode for this process is shown in Fig. 6 and the Python script is displayed in Appendix G.

This process was iterated multiple times as well, with multiple different combinations of parameter values. The combinations of the parameters can be seen in Appendix K. 96 predictions were made in total.

```
# rcm
selected values = a list of randomly selected dial numbers based on the probabilities with a length of the video length
tracking matrix = a list of interval times for each dial number
predicted trajectory = []
for each value in selected values
    if value is not in the queue (interval value of bigger than 0) and there is another value in the queue (interval value of 0)
        continue until value that is in the queue is selected
    else
        update interval value in tracking matrix of value that is selected

    for the whole sampling duration
        append selected value to predicted trajectory
        subtract 1 from the interval values for all dials in the tracking matrix if interval value is larger than 0

        if length of predicted trajectory is equal to video length
            break
```
Fig. 6: Pseudocode for random constrained sampling model

Conditional Sampling Model

The Conditional Sampling Model (CSM) assumes that a human operator not only samples based on the bandwidth of the dials (as in PSM/RCM), but also on the absolute pointer angle and/or its rate of change. This model results in an expected pointer angle after a certain amount of time.

The expected value of pointer angle with respect to time, given an initial reading Y is shown in Eq. 13.

$$\mu_{\hat{y}_i(t+\tau)|y_i(t)=Y_i} = \rho_i(\tau) Y_i \qquad (13)$$

Auto-correlation function is shown in Eq. 14 (where *i* = dial number, *W* = bandwidth in Hz, τ= time since last sampled in seconds).

$$\rho_i(\tau) = \frac{sin(2\pi W_i \tau)}{2 W_i \tau} \qquad (14)$$

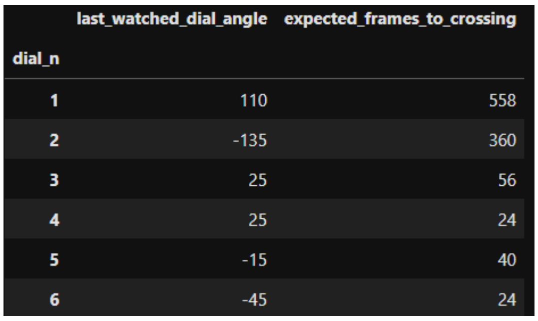
The sampling frequency ($2W$) used in the auto-correlation function can be assumed as a parameter because $2W$ applies to the ideal observer and may differ for a group of random humans.

In this model, the model may indicate that two dials are expected to cross at the same time, in which case one dial (not a specific one) must wait before being sampled. The amount of time that the dial has to wait before it gets samples is determined as a parameter called "wait time."

Parameters: Sampling frequency, wait time.

Trajectory

It is assumed that the human operator keeps track of the dial states in some way. Therefore, a tracking matrix was initialized which contains the pointer angles at frame 1 (taken from dial data) and the expected time after which the dial will cross the threshold, which is calculated based on formulas of CSM, as seen in Fig. 7. This means that an assumption was made that the participant knew all the dial states at the start, which was not the case. The expected pointer angle was calculated, and the timestamp at which the pointer angle is assumed to become 0 is chosen as the number of expected frames to crossing for each dial separately.



	last_watched_dial_angle	expected_frames_to_crossing
dial_n		
1	110	558
2	-135	360
3	25	56
4	25	24
5	-15	40
6	-45	24

Fig. 7: The tracking matrix with the last watched pointer angle and the expected frames to crossing per dial. The values differ from frame to frame and per video.

An initial dial was picked based on the chosen parameter value. It kept choosing that same dial for each next frame, updated the latest watched pointer angle and the expected frames to crossing value for the dial that was being watched, and subtracted 1 from the expected frames to crossing values of the dials that were not watched. This was done until one (or multiple) dials showed a value of 0 for the expected frames to crossing. In that case, the dial that had a value of 0 for expected frames to crossing was chosen as the dial that was being watched, and its corresponding values were updated in the tracking matrix. If there were multiple dials that showed a value of 0 for expected frames to crossing, the other dials had to wait for a specified number of frames until they got sampled. This process continued until there was a complete prediction of 3000 values. The pseudocode for this process can be seen in Fig. 8 and the Python script is displayed in Appendix H.

This process was repeated multiple times, with multiple different combinations of the parameter values. The combinations of the parameters can be seen in Appendix K. 96 predictions were made in total.

```
# csm
tracking matrix = a list of expected frames to crossing and last watched pointer angle for each dial
number
predicted trajectory = [initial state]
for each frame in video length
    candidate =  previously selected value
    if another value is present with an expected frames to crossing of 0
        candidate = value with expected frames to crossing of 0
    subtract 1 from expected frames to crossing in tracking matrix
    if there are negative expected frames to crossing in tracking matrix
        replace the expected frames to crossing with wait time

    append selected value to predicted trajectory
    replace last watched pointer angle and expected frames to crossing in tracking matrix with the
pointer angle of last predicted dial
```
Fig. 8: Pseudocode for conditional sampling model

Evaluation

The evaluation of the predictions will happen based on the same metrics used for the data analysis of the experimental data. That is: Glance rate (Hz), mean glance duration (s) and Percent AOI (%). Refer to the experiment method section for the definitions. The format of the predictions is of the same format as the results from the experiments. The script used to evaluate the predictions is shown in Appendix J.

Results

Throughout the results, the setups are called FV (for a full view of the display) and GCV (for a gaze-contingent view of the display). The results from the experiment and the predictions are divided in two different sections.

Experiment

Sampling behavior in general

Fig. 9 shows the relationship between the signal bandwidths and the sampling behavior. For the glance rate, the line for FV is steeper than GCV. The difference in setups in terms of glance rate for the higher bandwidth dials is bigger than the lower bandwidth dials.
The percentage of time on AOI shows, again, a steeper line for FV compared to GCV. The lower bandwidth dials get more time, and the higher bandwidth dials get less time allocated. The mean glance duration lines are similarly steep for both setups, but participants allocate more time to each fixation in GCV compared to FV. Table 4 shows the linear regression values for the data points observed with participant data. The correlation (Scipy.Stats.Linregress — SciPy v1.8.1 Manual, n.d.) between the variables and the bandwidth of the dials structurally shows a lower value for GCV than for FV.

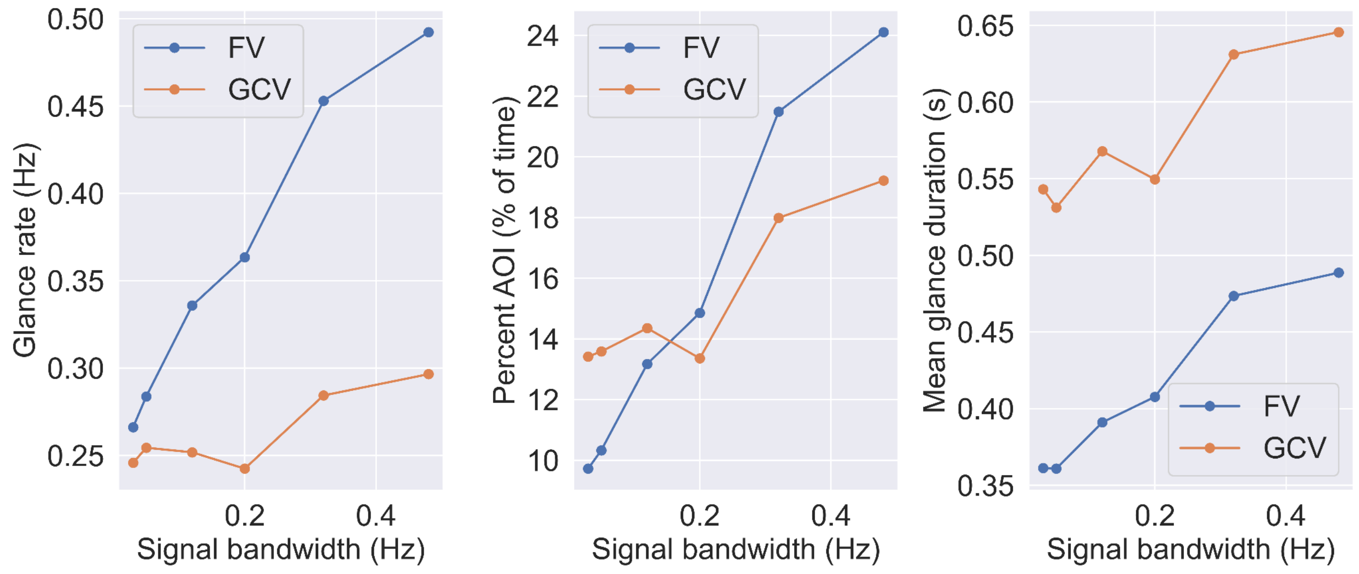


Fig. 9: Glance rate, percentage of time on area of interest and mean glance duration per bandwidth. Both the values for FV and GCV are shown.

Table 4: Sampling behavior characteristics per setup and their linear regression results. The intercept, slope and R-values of the linear regression are shown.

Setup	**Characteristic**	**Intercept**	**Slope**	**R**
FV	Glance rate	0.26	0.51	0.98
FV	Mean glance duration	0.35	0.31	0.98
FV	Percent AOI	8.88	33.69	0.99
GCV	Glance rate	0.24	0.11	0.88
GCV	Mean glance duration	0.53	0.26	0.93
GCV	Percent AOI	12.57	13.76	0.92

Sampling behavior over time

Fig. 10 shows the aggregated results for the percentage of time participants have sampled a certain AOI per timestamp. The data is grouped into bins of 10 seconds each. In total, seven videos in two setups are seen by participants in different orders. This overview shows how participants form their expectations of certain dials in the two setups over time.

The difference in percentage that an AOI is watched for low and high bandwidth dials is smaller in GCV than in FV. In FV, participants look relatively more at the two highest bandwidth dials (0.32 and 0.48 Hz) than they do in GCV. This difference is consistent over time. Another difference that can be noted between both setups, is that in FV, the participants seem to learn over time that a certain dial does not need attention. This can be seen in the lower bandwidth dials (0.03 and 0.05 Hz), where the percentage of time on AOI decreases over time in each video.

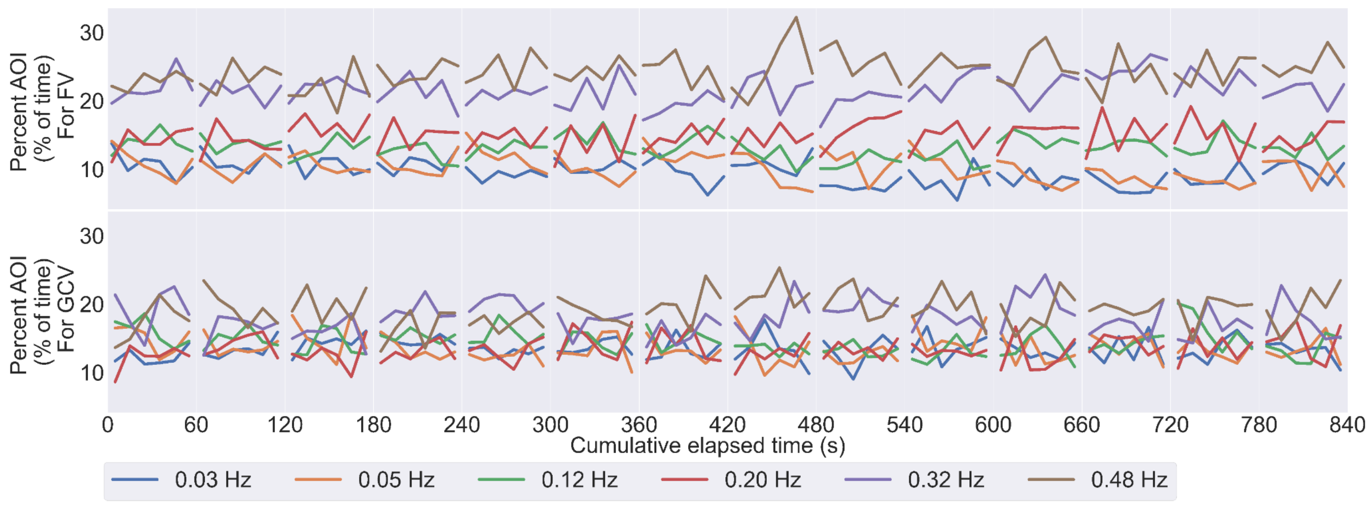


Fig. 10: Percentage of time that participants sampled a particular dial as a function of the total elapsed video time in 10 second increments. Considering 14 videos, each lasting 60 seconds, are watched in total, the total duration is 60 x 14 = 840 seconds.

Sampling behavior over dial characteristics

Fig. 11 shows what percentage of pointer angles, speeds, and time-to-crossings are seen by participants. These results are aggregated (mean) over all dials. It can be seen that participants watch a greater proportion of dials with a pointer angle close to 0 than dials with higher pointer angles. The same applies to time-to-crossing. When a pointer angle is closer to a crossing in terms of seconds, it gets more attention. For both analyses, the difference for both setups is there, but is not significant. It can be seen that certain values have a difference (in both directions) of approximately 2.5%.

In FV, higher speeds are seen more often than lower speeds are. This contrasts with GCV, where the difference between the percentage of higher speeds that are seen and the percentage of lower speeds that are seen is smaller (see Fig. 11, middle).

Fig. 12 zooms into these lines to analyze them per dial. Something that can be noted from Fig. 12 is that the dial with a bandwidth of 0.20 Hz in GCV for the pointer speed plot shows different behavior at higher speeds (both positive and negative speeds) compared to other dials.

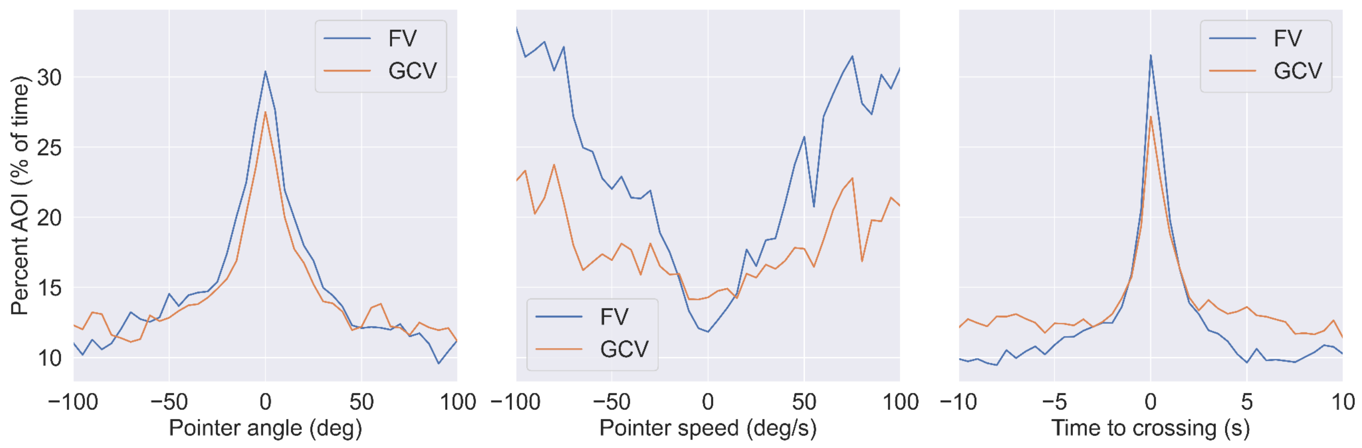


Fig. 11: Relationship between pointer angle in 5 degrees increments (left), pointer speed in 5 deg/s increments (middle), and time-to-crossing in 0.5 second increments (right) versus percent time on area of interest. These results are the aggregated results of all participants, videos and dials per setup.

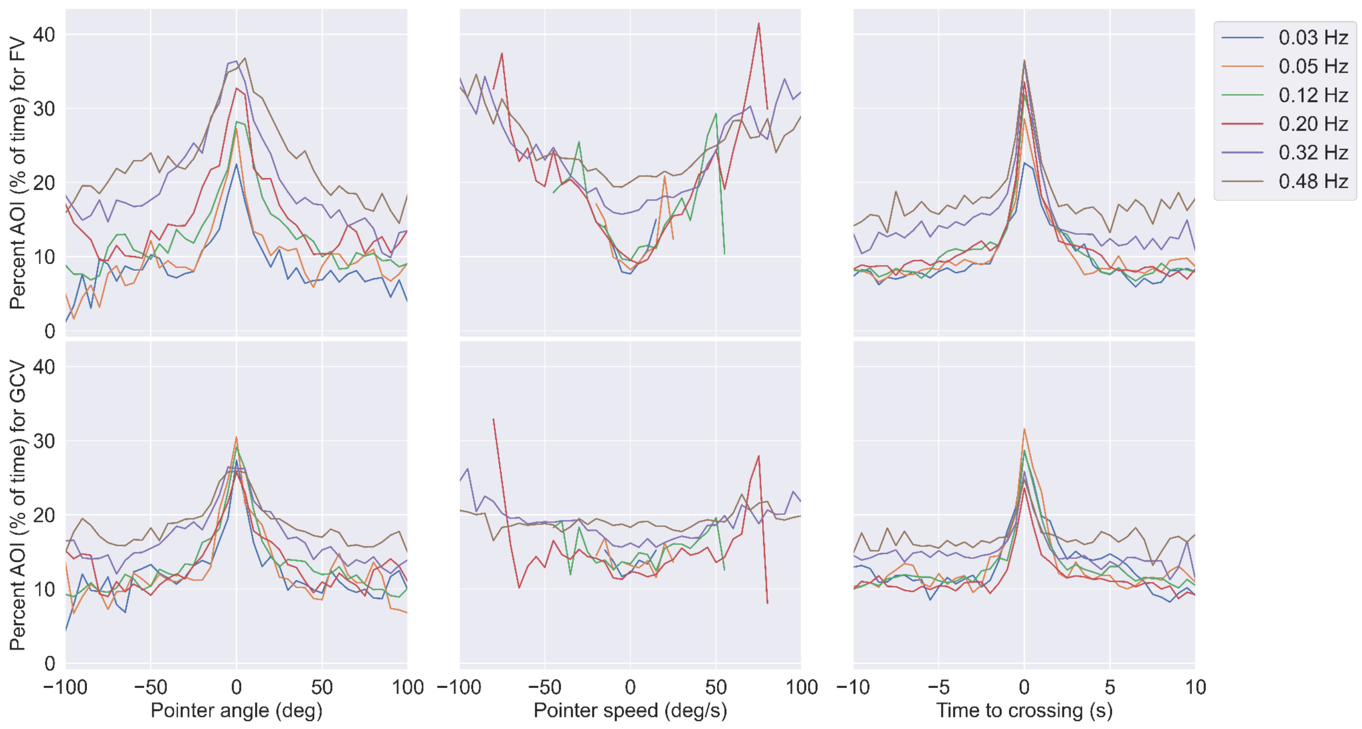


Fig. 12: Relationship between pointer angle in 5 degrees increments (left), pointer speed in 5 deg/s increments (middle), and time-to-crossing in 0.5 second increments (right) versus percent time on area of interest. These results are the aggregated results of all participants and videos per setup and per dial.

Task performance

Fig. 13 shows the task performance of participants (the correct spacebar presses divided by the number of all threshold crossings). The values included in the plot can also be inspected in Table 5. It can be seen that the task performance shows a lower value for GCV compared to FV. This difference is approximately 16%.

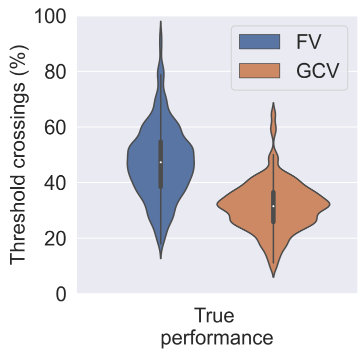


Fig. 13: Box plots and estimated kernel density plots of task performance per setup.

Table 5: Median and std of task performance for FV and GCV.

Model	**Median**	**std**
Performance FV	47.27	13.15
Performance GCV	31.48	8.99

Predictions

Sampling behavior in general

As the predictions that are made using the sampling models of Senders (1983) and the pragmatic models result in the same format as the sampling results of participants, the same type of visual as in Fig. 9 can be made to analyze the “sampling behavior” of the predictions. Fig. 14 (pragmatic) and Fig. 15 (Senders) show the relationship between the signal bandwidths and the sampling behavior for the predictions. The values in this figure are shown in Table 6 as well. Table 6 shows the linear regression results for each model and variable.

- The model based on pointer angle (da) shows negative correlation values.
- The model based on time-to-crossing (ttc) shows negative correlation values for mean glance duration, while they are positive for glance rate and percentage of time on AOI.
- The model based on the speed of the pointers (sp), psm, rcm and csm show a positive correlation for all three characteristics.

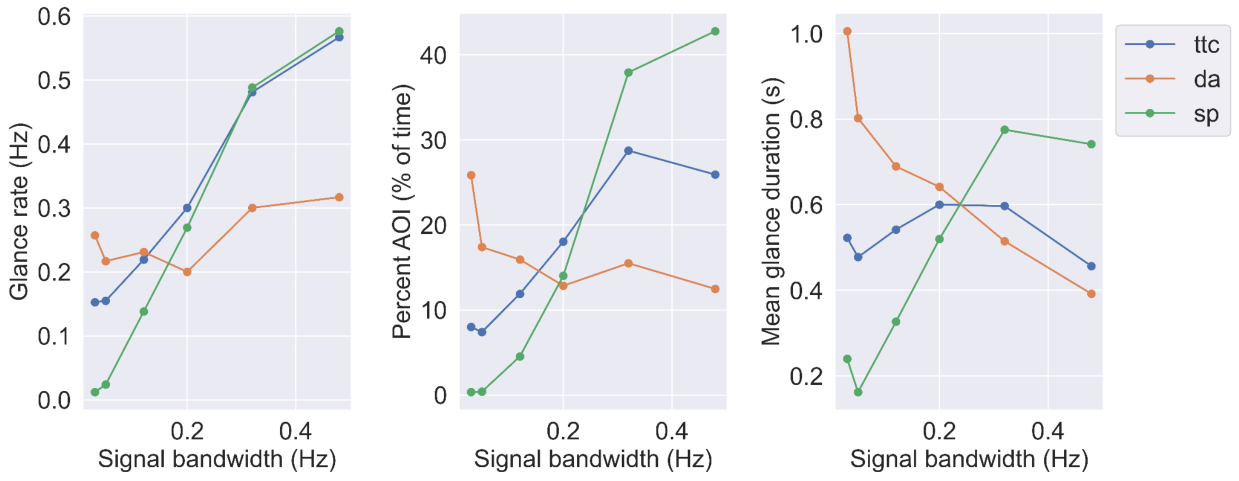


Fig. 14: Glance rate, percentage of time on area of interest and mean glance duration per bandwidth. The values are shown for the predicted trajectories. Abbreviations: ttc (time-to-crossing), da (pointer angle), sp (pointer speed).

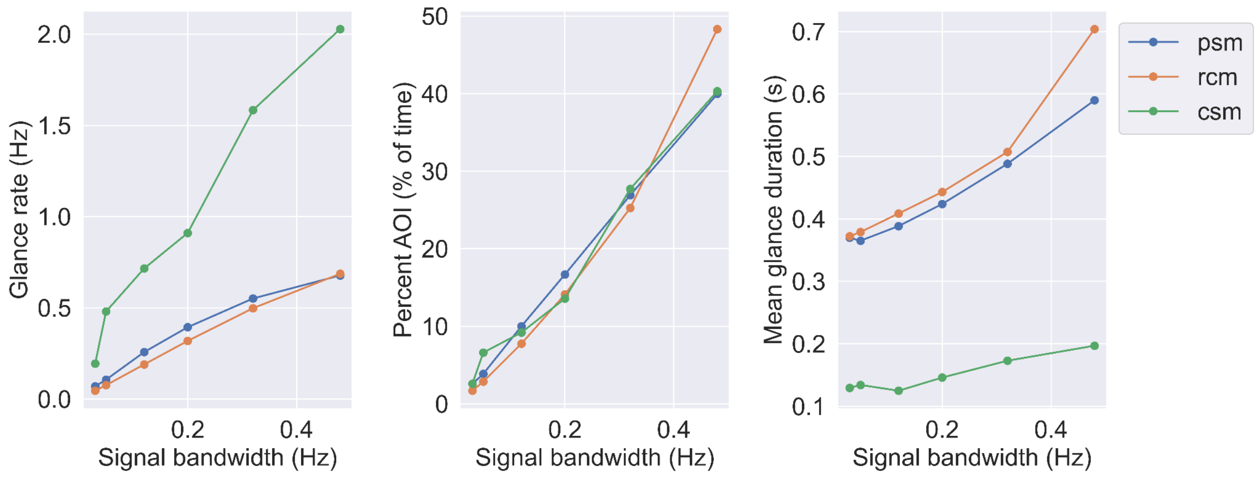


Fig. 15: Glance rate, percentage of time on area of interest and mean glance duration per bandwidth. The values are shown for the predicted trajectories. Abbreviations: psm (periodic sampling model), rcm (random constrained sampling model), csm (conditional sampling model).

Table 6: Sampling behavior characteristics per model and its linear regression results for the predicted trajectories. The intercept, slope and R-value of the linear regression is shown.

Model	**Characteristic**	**Intercept**	**Slope**	**R**
csm	Glance rate	0.19	3.95	0.99
csm	Mean glance duration	0.12	0.16	0.96
csm	Percent AOI	0.14	82.66	0.99
da	Glance rate	0.21	0.20	0.74
da	Mean glance duration	0.91	-1.16	-0.93
da	Percent AOI	20.55	-19.44	-0.69
psm	Glance rate	0.07	1.38	0.98
psm	Mean glance duration	0.34	0.50	0.99
psm	Percent AOI	-0.07	83.68	1.00
rcm	Glance rate	0.01	1.45	1.00
rcm	Mean glance duration	0.33	0.70	0.97
rcm	Percent AOI	-3.55	101.09	0.99
sp	Glance rate	-0.02	1.35	0.98
sp	Mean glance duration	0.18	1.39	0.93
sp	Percent AOI	-4.66	106.64	0.97
ttc	Glance rate	0.11	0.99	0.99
ttc	Mean glance duration	0.54	-0.03	-0.09
ttc	Percent AOI	6.99	48.38	0.92

Sampling behavior over dial characteristics

Fig. 16 (pragmatic) and Fig. 17 (Senders) show what percentage of pointer angles, speeds, and time-to-crossings are predicted to be seen by the predicted trajectories. It can be seen that certain pragmatic prediction models show that they are based on either pointer angle, pointer speed, or time-to-crossing.

- The model based on time-to-crossing shows that lower absolute time-to-crossing values result in a higher percentage of time on AOI. As this is related to the pointer angle and pointer speed as well, it can be seen that smaller absolute pointer angles and higher pointer speeds get more attention.
- For the model based on pointer angle, a spike can be seen at lower pointer angles and lower time-to-crossings. Because the line is nearly horizontal, there is no discernible difference in pointer speed.
- The pragmatic model, using pointer speed, shows that higher absolute pointer speeds and higher absolute pointer angles get more attention. No significant relationship is observable in the time-to-crossing plot.
- PSM shows that a higher absolute pointer speed, higher absolute pointer angle, and higher absolute time-to-crossing result in a higher percentage of time on AOI.
- RCM shows that a higher absolute pointer speed, higher absolute pointer angle, and higher absolute time-to-crossing result in a higher percentage of time on AOI.
- CSM shows no apparent relationship between pointer angle and time-to-crossing versus percentage of time on AOI, but shows that higher pointer speeds attract more attention.

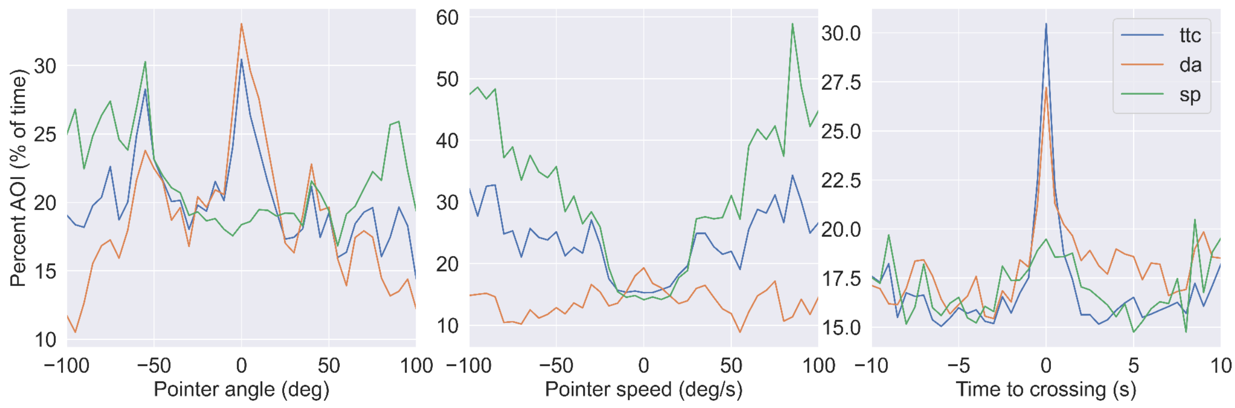


Fig. 16: Relationship between pointer angle in 5 degrees increments (left), pointer speed in 5 deg/s increments (middle), and time-to-crossing in 0.5 second increments (right) versus percent time on area of interest. These results are the aggregated results of all the dials per prediction model.

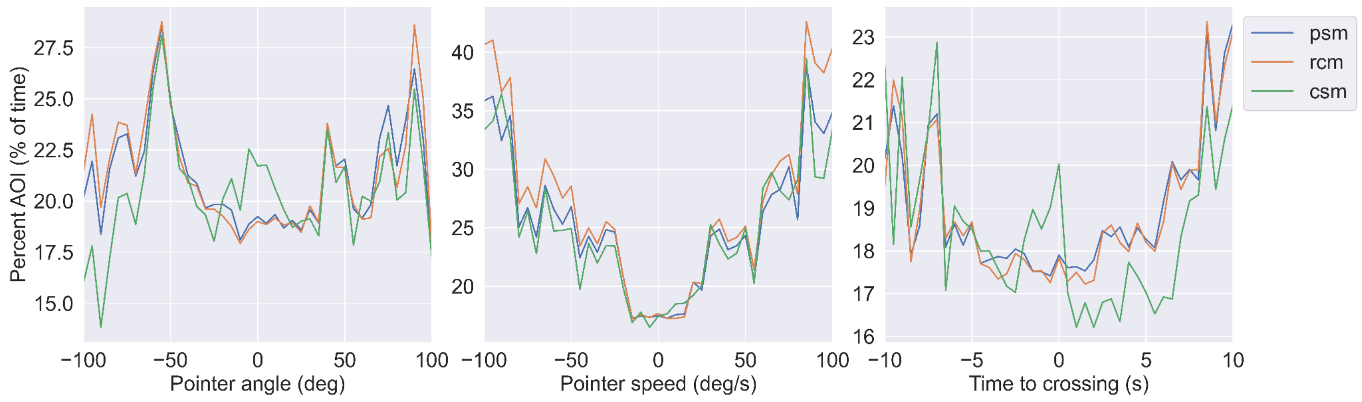


Fig. 17: Relationship between pointer angle in 5 degrees increments (left), pointer speed in 5 deg/s increments (middle), and time-to-crossing in 0.5 second increments (right) versus percent time on area of interest. These results are the aggregated results of all the dials per prediction model.

Discussion

The aim of this study was divided in two parts. The first part was to quantify the effect of a gaze-contingent window on sampling behavior and task performance. The second part was to find out in what kind of sampling behavior the predictions of pragmatic models and sampling models by Senders (1983) result in and compare the results to experimental data. The results will be discussed in two sections, which relate back to the research objectives as stated in the introduction. The first section is about the experiment, and the second section is about the predictions.

Experiment

Original setup

The findings in the results section allow for a comparison with the results of the work of Eisma et al. (2018). The glance rate, percentage of time on AOI, and mean glance duration in Fig. 9 and the results of Eisma et al. (2018) show a similar trend. The intercept is a bit higher and the slope of the linear line is slightly lower in this study. This could be related to the fact that the videos used in this experiment are 30 seconds shorter (60 seconds in this study, 90 seconds in Eisma et al., 2018), which could make the learning effect less apparent. The learning effect over time described in Eisma et al. (2018) is observable in Fig. 10 as well. Lower bandwidth dials get less attention over time, while other bandwidths get more. A reason for this could be that the subjects remembered the last state of the dials, knew that the speed was low, and concluded that no attention was needed for that specific dial for some time.
Looking at the task performance (pressing the spacebar whenever participants notice a pointer crossing a threshold), it shows comparable results to Eisma et al. (2018). The median score is 47.27% with a standard deviation of 13.15% in this study (Table 5), while it was between 47.53% and 51.40% with a standard deviation between 8.11% and 9.14% in the previous study.

Gaze-contingent setup

The effect of adding a gaze-contingent window is noticeable in the results. In Fig. 9, it can be seen that the glance rate is lower for low bandwidth dials and much lower for higher bandwidth dials. This indicates that participants looked longer at each dial, on average. This is observable in the mean glance duration plot as well. The reason for this could be that participants needed more time to be sure of a certain condition of the dials. Another reason could be that participants felt like they would risk losing a point if they looked away from a dial that they thought would cross the threshold. The percentage of time on the AOI plot shows that participants distributed their attention more evenly over the dials compared to the original setup.

Fig. 10 shows the effect of the gaze-contingent window on the percentage of time on AOI over time. In contrast to the original setup, the difference between the higher and lower bandwidth dials is smaller and there is no apparent learning. As seen in Fig. 11, with the gaze-contingent window, the effect of certain dial characteristics changes. The difference in pointer angle and time-to-crossing is negligible, but when looking at pointer speed, the distribution of attention changes significantly. The percentage of time on AOI at lower speeds (between -15 and 15 deg/s), is higher, and at higher speeds (smaller than -15 and bigger than 15 deg/s) the percentage of time on AOI is lower. This could suggest that certain speeds are noticed from the peripheral view, which attracts attention, as suggested by Eisma et al. (2018).

The addition of a gaze-contingent window lowered the task performance, as seen in Fig. 13. The median score for that setup is 31.48%, with a standard deviation of 8.99% (Table 5). This shows that the lack of a peripheral view lowers task performance.

Predictions

Fig. 14 shows that the pragmatic model using pointer angles does not result in the expected relationship (higher bandwidth means higher glance rate, percent AOI and mean glance duration), while the model based on time-to-crossing (except for the mean glance duration, which shows a near horizontal line) and speed does result in such a relationship. This could be because the higher bandwidth dials change pointer angles quickly, in contrast to lower bandwidth dials, which stay around the same pointer angle for a longer time. Thus, if a lower bandwidth dial has a lower absolute pointer angle, it will have it for a longer period of time and will be chosen as a prediction more frequently. Fig. 15 shows that the CSM model predicts sampling trajectories with different characteristics than the PSM and RCM models. While the PSM and RCM models show characteristics that are expected, the CSM does not. The CSM model shows a steep curve for the glance rate, with an approximate maximum average value of 2 Hz. This value is not realistic, as it would mean that the dial gets fixated on twice per second, while still having five other dials to fixate on as well. In Fig. 15, it is seen that, based on the expectations, the mean glance duration is very low (compared to other models), which then results in higher glance rates. This means that the expected frames to crossing is low for a frequent number of predicted frames. A reason for this discrepancy between the expected characteristics and prediction characteristics could be that CSM is based on expectations of pointers crossing their thresholds, which is directly related to task performance (noticing threshold crossings, then pressing the spacebar key).

In Fig. 16, the pragmatic models show what they are based on, namely, the pointer angle, pointer speed and time-to-crossing. This complies with the expectations. In Fig. 17, the CSM model shows differences compared to the PSM and RCM models. This is expected to be for the same reason as described before. All models show an expected relationship between the percentage of time on AOI and pointer speed, but they do not show the expected relationship between the percentage of time on AOI and pointer angle and time-to-crossing. The expected reason for that is that the models do not predict based on dial characteristics but on probabilities (PSM and RCM) and expectations (CSM).

The similarity between the characteristics of the predictions and experimental data is hard to measure. This is mainly because some characteristics of a prediction model may comply with the characteristics of experimental data, while others will not. When the graphs are compared between the predictions and experimental data, it can be seen that the highest similarities are found in the PSM and RCM models.

Conclusion and recommendations

The aim of this study was to gather insight into how adding a gaze-contingent window affects sampling behavior and task performance and how the predictions of sampling models by Senders (1983) compare to experimental data. The results of the setup with all the dials shown from this study show similarities with the original setup, as used in Eisma et al. (2018). Both sampling behavior and task performance were affected by the addition of a gaze-contingent window. In the setup with a gaze-contingent window, participants distributed their attention more equally than in the original setup, and their task performance score dropped. A big contributor in the peripheral area seems to be the pointer speed of the dials. A notable difference is that there is no discernible learning effect in the gaze-contingent setup, which was present in the original setup.

The predictions showed similarities with the experimental data on some characteristics, while they showed differences with others. The PSM and RCM showed the highest similarity with the experimental data. Other models either show characteristics that do not comply with reality or participant data, or the correlations are off.

Future work could be done on the diversity of participants. All three experiments (Senders, 1983; Eisma et al., 2018; this study) had a sample group of students. It may be interesting to see how sample behavior changes with age, study background, and experience in demanding sampling behavior settings.

Considering the effect of peripheral vision, it could be investigated how the sampling would be affected if, instead of a gaze-contingent setup, a perifoveal display is used as described in Heun et al. (2012). In this study, the information stream in the foveal area is more detailed and dense, while the information streams in the peripheral view become more abstract the further they are away from the fixation point. An implementation could be to enlarge dials that are in the peripheral vision while shrinking the dial in the foveal area. This dynamic setup could lead to another view on visual sampling behavior and may open doors to further improve monitoring situations.

This study has implemented models that were able to create sampling trajectories. An interesting study could be whether a setup can be created in which participants are instructed to follow the predicted trajectories instead of their own sampling strategy. A setup could be created, in which the dial that is predicted to be sampled is highlighted in such a way that a participant does not have to hesitate about where to look at. Another option is to only show one dial on a screen, but change the dial that is being shown based on the predicted trajectory. This removes the need for the participant to move their eyes. With the results of that experiment, the task performance can be compared to conclude whether the task performance is similar. This could lead to a setup where predictions help operators by presenting the relevant information to them, thus removing the need for the operators to gaze around.

This study contributes by quantifying the effect of the peripheral view on sampling behavior in the experiments performed by Senders (1983) and Eisma et al. (2018). Furthermore, sampling models of Senders (1983) have been implemented to predict complete trajectories, of which some can potentially perform as well as participants.

Bibliography

Alam, T. (2021). Cloud-Based IoT Applications and Their Roles in Smart Cities. Smart Cities, 4(3), 1196–1219. https://doi.org/10.3390/smartcities4030064

Campolongo, F., Saltelli, A., & Cariboni, J. (2011). From screening to quantitative sensitivity analysis. A unified approach. Computer Physics Communications, 182(4), 978–988. https://doi.org/10.1016/j.cpc.2010.12.039

Dadashi, N., Golightly, D., & Sharples, S. (2016). Seeing the woods for the trees: The problem of information inefficiency and information overload on operator performance. IFAC-PapersOnLine, 49(19), 603–608. https://doi.org/10.1016/j.ifacol.2016.10.628

Eisma, Y. B., Cabrall, C. D. D., & De Winter, J. C. F. (2018). Visual sampling processes revisited: Replicating and extending senders (1983) using modern eye-tracking equipment. IEEE Transactions on Human-Machine Systems, 48, 526–540. https:// doi. org/ 10.
1109/ THMS. 2018. 2806200

Eisma, Y. B., Hancock, P. A., & De Winter, J. C. F. (2020). On Senders's Models of Visual Sampling Behavior. Human Factors: The Journal of the Human Factors and Ergonomics Society, 001872082095995. https://doi.org/10.1177/0018720820959956

Herman, J., & Usher, W. (2021). SALib.sample.saltelli — SALib 1.4.6b0.post1.dev20+gd3c9348.d20220605 documentation. SALib. Retrieved 27 June 2022, from https://salib.readthedocs.io/en/latest/_modules/SALib/sample/saltelli.html

Heun, V., von Kapri, A., & Maes, P. (2012). Perifoveal display: combining foveal and peripheral vision in one visualization. In Proceedings of the 2012 ACM Conference on Ubiquitous Computing (pp. 1150-1155).

Perrow, C. (1981). Normal accident at three Mile Island. Society, 18(5), 17–26. https://doi.org/10.1007/bf02701322

Peterson, M. J. (2009). Case Study: Bhopal Plant Disaster. International Dimensions of Ethics Education in Science and Engineering, 4.

Rosenholtz, R. (2016). Capabilities and Limitations of Peripheral Vision. Annual Review of Vision Science, 2(1), 437–457. https://doi.org/10.1146/annurev-vision-082114-035733

scipy.stats.linregress — SciPy v1.8.1 Manual. (n.d.). SciPy. Retrieved 28 June 2022, from https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.linregress.html

Senders, J.W. (1964) The Human Operator as a Monitor and Controller of Multidegree of Freedom Systems

Senders, J. W. (1983) Visual sampling processes [Doctoral Dissertation, Katholieke Hogeschool Tilburg].

Shannon, C. (1949). Communication in the Presence of Noise. Proceedings of the IRE, 37(1), 10–21. https://doi.org/10.1109/jrproc.1949.232969
Spachos, P., & Gregori, S. (2019). Integration of Wireless Sensor Networks and Smart UAVs for Precision Viticulture. IEEE Internet Computing, 23(3), 8–16. https://doi.org/10.1109/mic.2018.2890234

Sharma, C., Bhavsar, P., Srinivasan, B., & Srinivasan, R. (2016). Eye gaze movement studies of control room operators: A novel approach to improve process safety. Computers & Chemical Engineering, 85, 43–57. https://doi.org/10.1016/j.compchemeng.2015.09.012

Sobol', I. (1967). On the distribution of points in a cube and the approximate evaluation of integrals. USSR Computational Mathematics and Mathematical Physics, 7(4), 86–112. https://doi.org/10.1016/0041-5553(67)90144-9

SR Research (2020). Experiment Builder (Version 2.3.38) [Computer Software]. https://www.sr-research.com/

SR Research (n.d.). EyeLink 1000 Plus [Apparatus and software]. https://www.sr-research.com/eyelink-1000-plus/

Wickens, C. D., Helleberg, J., Goh, J., Xu, X., & Horrey, W. J. (2001). Pilot task management: Testing an attentional expected value model of visual scanning. Savoy, IL, UIUC Institute of Aviation Technical Report.

Zhang, X., Mahadevan, S., Lau, N., & Weinger, M. B. (2020). Multi-source information fusion to assess control room operator performance. Reliability Engineering & System Safety, 194, 106287. https://doi.org/10.1016/j.ress.2018.10.012

Appendices

Appendix A

Consent form which is signed by all participants.

Gaze trajectory prediction using Senders (1983) models and evaluating them with experimental data with and without contingent windows

Informed consent form for participants

Researchers

MSc student: A Bakay
E-mail: a.bakay@student.tudelft.nl
Tel: +[redacted]

Supervisor: Dr.ir. Y.B. Eisma
E-mail: y.b.eisma@tudelft.nl

Supervisor: Dr.ir. J.C.F. de Winter
E-mail: j.c.f.dewinter@tudelft.nl

Location

Delft University of Technology
Faculty of Mechanical, Maritime and Material Engineering (3mE)
Department of Cognitive Robotics
Mekelweg 2, 2628 CD, Delft
Room 34-F-1-600

This document describes the purpose of this research, experiment procedure, risks of participating, right to withdraw, data treatment, and preventative measures related to COVID-19. Please read all sections carefully and respond to the statements on page 3.

Purpose of this research

In supervisory tasks where visual sampling of multiple sources of information is relevant, the human operator is expected to distribute its attention over the sources in a manner that critical information is not missed. To model the distribution of the operator, Senders (1983) has described various mathematical models of attentional distribution. This project further develops the mathematical models as proposed by Senders (1983) to analyze relations between information source parameters and accommodate prediction of visual sampling in time domain. The aim of this experiment is to analyze the effect of contingent windows on attention distribution of an operator.

Senders, J. W. (1983) Visual sampling processes [Doctoral Dissertation, Katholieke Hogeschool Tilburg].

Experiment procedure

Before starting the experiment, you will be seated in front of a computer screen and comfortably position your head on a desk-mounted headrest to be able to measure your eye movements accurately via an eye-tracker (Figure 1).

You will be guided through the experiment in a stepwise manner via instructions shown on the screen. First, the eye-tracker will be calibrated. Second, a training session will be given, so that you can get an idea of what to do. Finally, the experiment will begin.

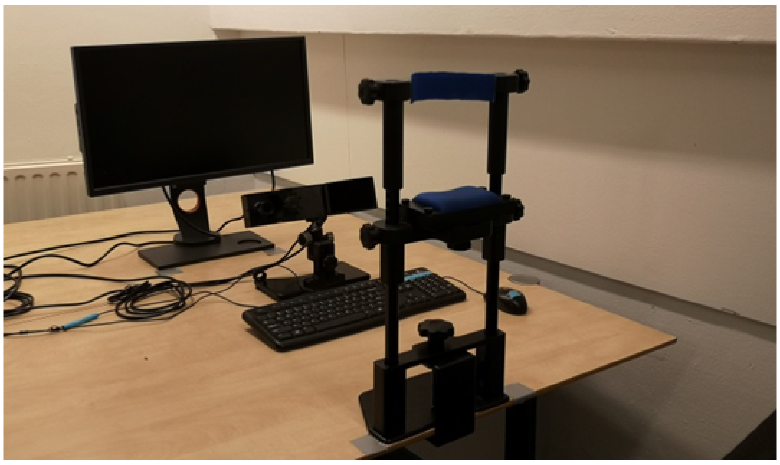


Figure 1. Experimental setup with head support and eye tracker

During the experiment, you will take the perspective of an operator that has to monitor 6 dials which all have a different bandwidth (speed). There are two setups. In the first setup you will be shown all dials at once, in the second setup, your vision will be limited to the location you are focusing on, which limits your ability to detect significant changes in your perifoveal vision.

Your task is to press the spacebar whenever any of the dials on the screen crosses its threshold.

The total length of the experiment is estimated to be 20-25 minutes.

Risks of participating

There are no expected risks to participants. If you experience any discomfort, please inform the experiment supervisor so that the experiment can be stopped. You may take your head out of the headrest any time if you feel unwell.

Right to withdraw

Your participation is completely voluntary, and you may stop at any time during the experiment for any reason. There will be no negative consequences for withdrawing from the experiment.

Data treatment

All data collected during the experiment will be stored anonymously and used for academic research only. The eye-tracker only records eye movements and no images of your eyes or face. You will not be personally identifiable in any future publications based on this work or in any data files shared with other researchers. This signed consent form will be kept by Dr.ir. Y.B. Eisma in a dedicated locker.

Preventative measures related to COVID-19

To minimize the risk of COVID-19, you may not participate if you:

- Show any symptoms indicative of COVID-19;
- Have been in contact with COVID-19 patients within the last 14 days.

The following preventative measures will be required for you to participate:

- Wash your hands thoroughly before entering the lab

All equipment used in the experiment will be disinfected before participation.

Please respond to the following statements

Statement	**Yes**	**No**
I consent to voluntarily participate in this study.		
I have read and understood the information provided in this document.		
I adhere to the preventative measures with regards to COVID-19 as explained above.		
I understand that I can withdraw from the study at any time without any negative consequences.		
I consent that the data gathered during the experiment may be used for a MSc thesis and possible future academic research and publications.		

Signature

Name: ………………………………….

Date: …………………………………... Signature: …………………………………………………

Appendix B

Python kernel information and dependencies.

Python kernel and JupyterLab information

- Python 3.8.10 (tags/v3.8.10:3d8993a, May 3 2021, 11:48:03) [MSC v.1928 64 bit (AMD64)]
- IPython 7.24.0
- JupyterLab Version 3.3.2

Dependencies

- matplotlib==3.4.2
- numpy==1.20.3
- pandas==1.2.4
- SALib==1.4.5
- scikit_learn==1.1.1
- scipy==1.7.3
- seaborn==0.11.2

Appendix C

Data processing script.

```
#!/usr/bin/env python
# coding: utf-8
import matplotlib.pyplot as plt
import scipy.io
import pandas as pd
import mat73
import numpy as np
from matplotlib.patches import Rectangle
get_ipython().run_line_magic('matplotlib', 'inline')

# Initiate data locations
input_human_directory = "../2 Data/input_human/"
output_screen_directory = "../2 Data/output_screen/"

# Load dial data and assign the speed and time to cross values
dials = pd.read_parquet(input_human_directory + "dial_data.parquet")
dials.sort_values(['video_number', 'dial_watched', 'frame_number'], ascending=[True, True, True],
inplace=True)

tmp = []

# Function to set direction (1 is towards threshold, -1 is away from threshold, 0 is unknown)
def set_direction(row):
    if (row["speed"]>0 and row["watched_dial_angle"]<0) or (row["speed"]<0 and
row["watched_dial_angle"]>0):
        return 1
    elif (row["speed"]>0 and row["watched_dial_angle"]>0) or (row["speed"]<0 and
row["watched_dial_angle"]<0):
        return -1
    else:
        return 0


for i in range(1,8):
    for j in range(1,7):
        dialtmp = dials.copy()
        dialtmp = dialtmp[(dialtmp["video_number"] == i) & (dialtmp["dial_watched"] == j)]
        dialtmp["speed"] = dialtmp['watched_dial_angle'].diff()*50
        dialtmp["speed"].fillna(0, inplace=True)


        dialtmp["time_to_cross"] = dialtmp["watched_dial_angle"] / -dialtmp["speed"]
        dialtmp["time_to_cross"].replace([np.inf, -np.inf], 90, inplace=True)

        dialtmp["speed"]= dialtmp["speed"].apply(lambda x: custom_round(x, base=5))
        dialtmp["watched_dial_angle"] = dialtmp["watched_dial_angle"].apply(lambda x:
custom_round(x, base=5))
        dialtmp["time_to_cross"] = dialtmp["time_to_cross"].apply(lambda x: round_off_half(x))

        dialtmp = dialtmp.assign(direction=dialtmp.apply(set_direction, axis=1))

        tmp.append(dialtmp)
```

```
dials = pd.concat(tmp)
dials.to_parquet(input_human_directory + "dial_data_w_relative.parquet")

def custom_round(x, base=5):
    try:
        return int(base * round(float(x)/base))
    except:
        return 0

def round_off_half(number):

    return round(number * 2) / 2

# this matrix designates the bandwidth of each dial (top left, top middle, top right, bottom left,
bottom middle, bottom right), for video 1 to 7
dial_config=[
    [3, 6, 2, 4, 5, 1],
    [4, 6, 1, 5, 2, 3],
    [1, 3, 4, 5, 6, 2],
    [5, 3, 2, 6, 1, 4],
    [6, 2, 1, 3, 4, 5],
    [3, 5, 2, 4, 1, 6],
    [5, 1, 4, 3, 2, 6]];


# Define circle variables and plot circles
start_x = 326
interval_x = 634
interval_y = 658
start_y = [211, 211+interval_y]
clock_coordinates = [(start_x, start_y[0]),(start_x+interval_x, start_y[0]),(start_x+interval_x*2,
start_y[0]),(start_x, start_y[1]),(start_x+interval_x, start_y[1]),(start_x+interval_x*2,
start_y[1])]

# Rect data
rectangles = [[start_x-210,start_y[0]-210,420,420],
        [start_x+interval_x-210,start_y[0]-210,420,420],
        [start_x+interval_x*2-210,start_y[0]-210,420,420],
        [start_x-210,start_y[1]-210,420,420],
        [start_x+interval_x-210,start_y[1]-210,420,420],
        [start_x+interval_x*2-210,start_y[1]-210,420,420]]


def plot_screen_with_data(data):
    # Plot gaze points
    figure, axes = plt.subplots(1,1, squeeze=False)
    data.plot(x="gaze_x",y="gaze_y", style=",", legend=False, figsize=(10,10/1.77777777),
ax=axes[0,0])

    for coordinate in clock_coordinates:
        axes[0,0].add_patch(plt.Circle(coordinate, 158, color='g', fill=False, linewidth=2))

    for count, rect in enumerate(rectangles):
        a = 0
        axes[0,0].add_patch(Rectangle((rect[0],rect[1]), rect[2], rect[3], facecolor="none", ec='k',
lw=2))

    # Set plot parameters
#     plt.title("Coordinates for participant {} video {}".format(candidate, video))
    plt.xlim(0,1920)
    plt.ylim(0,1080)
    plt.gca().invert_yaxis()

# ## Load participant data and clean

participant_data_raw = pd.read_csv(input_human_directory + 'results_raw.xls', sep="\t",
dtype={"RECORDING_SESSION_LABEL": str, "setups_str": str, "video_int": int, "VIDEO_FRAME_INDEX":
str, "AVERAGE_GAZE_X": str, "AVERAGE_GAZE_Y": str})


participant_data = participant_data_raw[participant_data_raw["VIDEO_FRAME_INDEX"]!="."]
participant_data['setups_str'].replace({"A": 1, "B": 2}, inplace=True)
participant_data['RECORDING_SESSION_LABEL'].replace({"p01": 1}, inplace=True)
participant_data['RECORDING_SESSION_LABEL'] =
participant_data['RECORDING_SESSION_LABEL'].astype(int)
participant_data['setups_str'] = participant_data['setups_str'].astype(int)
participant_data.columns = ["participant_number", "setup", "video_number", "frame_number", "gaze_x",
"gaze_y"]
participant_data['frame_number'] = participant_data['frame_number'].astype(int)

participant_data = participant_data.replace('.', np.nan)
participant_data['gaze_x'] = participant_data['gaze_x'].astype(float)
participant_data['gaze_y'] = participant_data['gaze_y'].astype(float)
participant_data["gaze_x"] = participant_data["gaze_x"].interpolate()
participant_data["gaze_y"] = participant_data["gaze_y"].interpolate()

participant_data["dial_watched"] = 0
participant_data


# ## Extract viewing order
rank_prep = participant_data.groupby(["participant_number", "setup", "video_number"], sort=False,
as_index=False).count().iloc[:,:3]
rank_prep['Rank'] = rank_prep.groupby(['participant_number']).cumcount()+1

rank_prep.to_csv(input_human_directory + "legit_combos_ranks.csv" ,index=False)

# ## Create legit combos
legit_combos =
participant_data.groupby(["participant_number","setup","video_number"]).count().reset_index().iloc[:
,:3]
pd.Series(list(legit_combos.itertuples(index=False, name=None))).to_csv(input_human_directory +
"legit_combos_3000.csv" ,index=False)


# ## Load legit combos
legit_combos = pd.read_csv(input_human_directory + "legit_combos_3000.csv")
legit_combos = legit_combos['0'].str[1:-1].str.split(',', expand=True).astype(int)
legit_combos
```

```python
# ### Loop through data to categorize into dial bounds
## Assign the watched dial to each measurement point
result_frames = []
for index,combo in legit_combos.iterrows():
    ## Select person and video
    candidate = combo[0]
    setup = combo[1]
    video = combo[2]

    # Load gaze data
    person = participant_data[(participant_data["participant_number"]==candidate) &
(participant_data["setup"]==setup) & (participant_data["video_number"]==video)].copy()

    # Check which dial (location of dial from left ro right, then top to bottom) was watched. Note
that if the coordinate is not in the rectangles, the dial_watch value will stay 0
    for count, rect in enumerate(rectangles):
        # Using dial config to locate the "real" dial number which corresponds to the frequency
        person.loc[(person["gaze_x"] >= rect[0]) & (person["gaze_x"] <= (rect[0]+rect[2])) &
(person["gaze_y"] >= rect[1]) & (person["gaze_y"] <= rect[1]+rect[3]),'dial_watched'] =
dial_config[video-1][count]
    result_frames.append(person)

# Concatenate results, note that the results do not include the areas between the dials (free area)
dials_attached = pd.concat(result_frames)

dials_attached.to_parquet(input_human_directory + 'dials_attached.parquet', index=False)

dials_attached = pd.read_parquet(input_human_directory + 'dials_attached.parquet')

# Test results
filt = dials_attached[dials_attached["dial_watched"]==5]
plot_screen_with_data(filt[(filt["video_number"]==1)])


# ### Group watched dials per frame
dials_watched = pd.read_parquet(input_human_directory + 'dials_attached.parquet')

participants = []
for index,combo in legit_combos.iterrows():
    ## Select person and video
    participant_n = combo[0]
    setup = combo[1]
    i = combo[2]

    dials_watched_pp = dials_watched[(dials_watched["video_number"]==i) &
(dials_watched["setup"]==setup) &
(dials_watched["participant_number"]==participant_n)].reset_index()

    # Select first video and reduce frequency of measurements by grouping the frames
    # Groupby the most occuring value
    dials_watched_pp = dials_watched_pp.groupby("frame_number").agg(lambda
x:x.value_counts().index[0]).reset_index()

    ### Pivot the table to be able to merge it with the dial data
    # Count watched dials per frame
    dials_watched_pp_grouped = 
dials_watched_pp.groupby(["frame_number","dial_watched"]).count().iloc[:,0].reset_index()
    # Select most watched dial per frame
    dials_watched_pp_grouped = 
dials_watched_pp_grouped.loc[dials_watched_pp_grouped.groupby(['frame_number'])[dials_watched_pp_gro
uped.columns[-1]].idxmax()]
    # Clean up
    dials_watched_pp_grouped = 
dials_watched_pp_grouped.set_index("frame_number")[["dial_watched"]].reset_index()
    dials_watched_pp_grouped.insert(0,'participant_number','')
    dials_watched_pp_grouped.insert(0,'video_number','')
    dials_watched_pp_grouped['video_number'] = i
    dials_watched_pp_grouped["participant_number"] = participant_n
    dials_watched_pp_grouped["setup"] = setup

    # Add participant number
    dials_watched_pp_grouped

    participants.append(dials_watched_pp_grouped)
    print("Participant {}".format(participant_n))

# Double check count per combo (should be 3000)
ddd = pd.concat(participants)
for index,combo in legit_combos.iterrows():
    ## Select person and video
    participant_n = combo[0]
    setup = combo[1]
    i = combo[2]
    print(len(ddd[(ddd["video_number"]==i) & (ddd["setup"]==setup) & 
(ddd["participant_number"]==participant_n)]))

# If everything is correct, save
ddd.to_parquet(input_human_directory + "final_gaze_3000.parquet", index=False)
```

Appendix D

Main script for predictions.

```
# Multiprocessing libraries
import datetime as dt
from datetime import datetime
import multiprocessing
from multiprocessing import Process, current_process, Manager
import sys

# Generic libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import random


# Import functions for SA
from SALib.sample import saltelli
from SALib.analyze import sobol, morris, fast

# Import functions for models
from a_pragmatic_predicting import predict_pragmatic_pp
from b_periodic_sampling_model import predict_using_psm
from c_random_sampling_model import predict_using_rcm
from d_conditional_sampling_model import run_csm

# Where to save results
results_dir = "../4 Results/"

####################################################################################################
#
####### Define the parameters for different methods
####################################################################################################
#

# Initialize empty dict
all_parameters_chunks = {}


# Pragmatic
# - This chunk contains the different options inside pragmatic
all_parameters_chunks["pragmatic"] = np.array_split(["ttc", "sp", "da", "spdd", "dadd",
"equal_probs", "single_value"], 2)

# PSM SA
problem = {
    "num_vars": 2,
    "names": ["samp_freq", "sampl_dur"],
    "bounds": [[1.5, 2.5], [0.2, 0.6]]
}
param_values = saltelli.sample(problem, 16)
all_parameters_chunks["psm"] = np.array_split(param_values, int(len(param_values)/4))
psm_shape = param_values.shape


# RCM SA
```

```python
problem = {
    "num_vars": 2,
    "names": ["sampl_dur","interval_correction"],
    "bounds": [[0.2, 0.6], [0.5, 1.5]]
}
param_values = saltelli.sample(problem, 16)
all_parameters_chunks["rcm"] = np.array_split(param_values, int(len(param_values)/4))
rcm_shape = param_values.shape


# CSM SA
problem = {
    "num_vars": 2,
    "names": ["double_crossing_wait", "sampling_frequency"],
    "bounds": [[1, 7], [1.5, 2.5]]
}
param_values = saltelli.sample(problem,16)
# Add constant values (video and initial_value) to the options
new_param_values = []
for par in param_values:
    toa = np.insert(par, 0, 4) # X is chosen as default video value as the change of had no impact
    toa = np.insert(toa, 0, 4)  # X is chosen as default initial value as the change of had no
impact
    new_param_values.append(list(toa))
param_values = np.array(new_param_values)
all_parameters_chunks["csm"] = np.array_split(param_values, int(len(param_values)/4))
csm_shape = param_values.shape


####################################################################################################
#
####### Define the function used to run the methods
####################################################################################################
#

# Functions for the processes to use
def do_it(x, results, method, video_length, save_trajectories):
    print(f"Method: {method}")

    # Print the start of the process
    print(current_process().name + " started")
    sys.stdout.flush()

    # All options and running them with the correct arguments, then appending it to the list of the
manager
    if method=="psm":
        perf_it = predict_using_psm(mont_carlo_count=1, samp_freq=x[0], sampl_dur=x[1],
video_length=video_length, save_trajectories=save_trajectories)
        results.append(pd.DataFrame(perf_it))
    elif method=="rcm":
        perf_it = predict_using_rcm(mont_carlo_count=1, sampl_dur=x[0], interval_correction=x[1],
video_length=video_length, save_trajectories=save_trajectories)
        results.append(pd.DataFrame(perf_it))
    elif method=="csm":
        perf_it = run_csm(mont_carlo_count=1, initial_state=int(x[0]), video=int(x[1]),
double_crossing_wait=x[2], sampling_frequency=x[3], video_length=video_length,
save_trajectories=save_trajectories)
        results.append(pd.DataFrame(perf_it))
    elif method=="pragmatic":
        if x=="ttc":
            perf_it = predict_pragmatic_pp(mode="ttc", video_length=video_length,
save_trajectories=save_trajectories)
            results.append(pd.DataFrame(perf_it))
        elif x=="sp":
            perf_it = predict_pragmatic_pp(mode="sp", video_length=video_length,
save_trajectories=save_trajectories)
            results.append(pd.DataFrame(perf_it))
        elif x=="da":
            perf_it = predict_pragmatic_pp(mode="da", video_length=video_length,
save_trajectories=save_trajectories)
            results.append(pd.DataFrame(perf_it))
        elif x=="spdd":
            perf_it = predict_pragmatic_pp(mode="spdd", video_length=video_length,
save_trajectories=save_trajectories)
            results.append(pd.DataFrame(perf_it))
        elif x=="dadd":
            perf_it = predict_pragmatic_pp(mode="dadd", video_length=video_length,
save_trajectories=save_trajectories)
            results.append(pd.DataFrame(perf_it))
        elif x=="equal_probs":
            perf_it = predict_pragmatic_pp(mode="equal_probs", mont_carlo_count=5,
video_length=video_length, save_trajectories=save_trajectories)
            results.append(pd.DataFrame(perf_it))
        elif x=="single_value":
            for i in range(1,7):
                perf_it = predict_pragmatic_pp(mode="single_value", single_predict_value=i,
video_length=video_length, save_trajectories=save_trajectories)
                results.append(pd.DataFrame(perf_it))
    # Print the end of the process
    print(current_process().name + " ended")
    sys.stdout.flush()

####################################################################################################
#
####### Main program
####################################################################################################
#

if __name__ == '__main__':
    # Initiate manager
    with Manager() as manager:
        # Create list based on results
        results = manager.list()
        # Monitor duration
        startTime = datetime.now()

        # Get selected method from args
        save_trajectories = False
        try:
            print("Selected method: ", sys.argv[1])
            selected_method = sys.argv[1]
            print("Video length: ", sys.argv[2])
```

```python
            video_length = int(sys.argv[2])
        except:
            selected_method = "all"
            video_length = 3000


        print(f"Total options for PSM: {psm_shape}")
        print(f"Total options for RCM: {rcm_shape}")
        print(f"Total options for CSM: {csm_shape}")

        # Go through all methods that are added to the dict
        for method in list(all_parameters_chunks.keys()):
            if selected_method!="all":
                if(method!=selected_method): continue
            # Go through all chunks per method
            for chunk in all_parameters_chunks[method]:
                # Initiate worker pool
                worker_pool = []
                # Go through all parts of the chunk
                for i in range(chunk.shape[0]):
                    # Get chunk values
                    x = chunk[i]
                    # Create and start process and pass arguments
                    p = Process(target=do_it, args=(x,results,method,video_length,
save_trajectories, ))
                    p.start()
                    worker_pool.append(p)

                # Wait for all processes of the current chunk to finish
                for p in worker_pool:
                    p.join()

                # Report time since start of current chunk
                print(datetime.now() - startTime)
            # Concatenate all results to csv file
            if(save_trajectories):
                pd.concat(results).to_csv(results_dir +
"results_{}_{}_{}.csv".format(selected_method, video_length, save_trajectories), index=False)
            else:
                pd.concat(results).to_csv(results_dir + "results_{}_{}.csv".format(selected_method,
video_length), index=False)
```

Appendix E

Pragmatic prediction script.

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import scipy
import random

from x_custom_functions import evaluate_on_participant_data

# Folder where data is located
input_human_directory = "../2 Data/input_human/"
results_dir = "../4 Results/"

# Unit step function
def step(x):
    return 1 * (x == 0)

# Load dial data

def predict_pragmatic_pp(mode, single_predict_value=None, mont_carlo_count=None, video_length=None,
save_trajectories=False):
    dial_data = pd.read_parquet(input_human_directory + "dial_data_w_relative.parquet")
    dial_data = dial_data[dial_data["frame_number"]<=video_length]


    # Initiate performance list
    performance = []

    def perform_evaluation(pivoted_dial_data_t):
        # Go through all videos (because each video has different dial angle values)
        for i in range(1,8):
            # Select the dial angle of the corresponding video (i) and clean things up
            pivoted_dial_data = pivoted_dial_data_t.loc[i]
            pivoted_dial_data = pivoted_dial_data.reset_index()
            pivoted_dial_data.columns = ["frame_number","dial_watched_pred"]
            y_pred = pivoted_dial_data["dial_watched_pred"].tolist()

            # Evaluate prediction and append results to list
            if(save_trajectories):
                evaluation = [[y_pred, {"Type":("pragmatic_"+mode)}]]
            else:
                evaluation = evaluate_on_participant_data(y_pred, spec_video=i,
parameters={"Type":("pragmatic_"+mode)}, video_length=video_length)

            performance.extend(evaluation)

    # Determine which mode to use
    # Time to crossing
    if mode == "ttc":
        # Pivot dial data and find dial with lowest time to cross value
        pivoted_dial_data_t =
pd.DataFrame(dial_data.pivot(index=["video_number","frame_number"],columns='dial_watched',
values=['watched_dial_angle','speed','time_to_cross',
'direction'])["time_to_cross"].abs().idxmin(axis="columns"))
        perform_evaluation(pivoted_dial_data_t)
    # Dial speed
    elif mode == "sp":
        # Pivot dial data and find dial with highest speed
        pivoted_dial_data_t =
pd.DataFrame(dial_data.pivot(index=["video_number","frame_number"],columns='dial_watched',
values=['watched_dial_angle','speed','time_to_cross',
'direction'])["speed"].abs().idxmax(axis="columns"))
        perform_evaluation(pivoted_dial_data_t)
    # Dial angle
    elif mode == "da":
        # Pivot dial data and find dial with lowest dial angle
        pivoted_dial_data_t =
pd.DataFrame(dial_data.pivot(index=["video_number","frame_number"],columns='dial_watched',
values=['watched_dial_angle','speed','time_to_cross',
'direction'])["watched_dial_angle"].abs().idxmin(axis="columns"))
        perform_evaluation(pivoted_dial_data_t)
    # Speed and dial direction
    elif mode == "spdd":
        # Pivot dial data
        pivoted_dial_data_t =
pd.DataFrame(dial_data.pivot(index=["video_number","frame_number"],columns='dial_watched',
values=['watched_dial_angle','speed','time_to_cross', 'direction']))
        # Loop through all videos
        for i in range(1,8):
            # Initiate prediction array
            y_pred = []
            # Loop through all rows with dial data
            for count, row in pivoted_dial_data_t.loc[i].iterrows():
                # Check if there is a dial available that is going towards the threshold
                if(len(row["direction"][row["direction"]==1])>0):
                    # Check which dial of the dials that go to the threshold has the highest speed
                    pos_dir_highest_sp =
row["speed"].loc[row["direction"][row["direction"]==1].index.values.tolist()].idxmax(axis="columns")
                    # Append to prediction
                    y_pred.append(pos_dir_highest_sp)
                else:
                    # Check which dial has the highest speed
                    highest_sp = row["speed"].idxmax(axis="columns")
                    # Append to prediction
                    y_pred.append(highest_sp)

            # Evaluate prediction and append results to list
            if(save_trajectories):
                evaluation = [[y_pred, {"Type":("pragmatic_"+mode)}]]
            else:
                evaluation = evaluate_on_participant_data(y_pred, spec_video=i,
parameters={"Type":("pragmatic_"+mode)}, video_length=video_length)
            performance.extend(evaluation)


    # Dial angle and dial direction
    elif mode == "dadd":
        # Pivot dial data
        pivoted_dial_data_t =
pd.DataFrame(dial_data.pivot(index=["video_number","frame_number"],columns='dial_watched',
values=['watched_dial_angle','speed','time_to_cross', 'direction']))
```

```
        # Loop through all videos
        for i in range(1,8):
            # Initiate prediction array
            y_pred = []
            # Loop through all rows with dial data
            for count, row in pivoted_dial_data_t.loc[i].iterrows():
                # Check if there is a dial available that is going towards the threshold
                if(len(row["direction"][row["direction"]==1])>0):
                    # Check which dial of the dials that go to the threshold has the lowest absolute
dial angle
                    pos_dir_highest_sp =
row["watched_dial_angle"].loc[row["direction"][row["direction"]==1].index.values.tolist()].abs().idx
min(axis="columns")
                    # Append to prediction
                    y_pred.append(pos_dir_highest_sp)
                else:
                    # Check which dial has the lowest absolute dial angle
                    highest_sp = row["watched_dial_angle"].abs().idxmin(axis="columns")
                    # Append to prediction
                    y_pred.append(highest_sp)

            # Evaluate prediction and append results to list
            if(save_trajectories):
                evaluation = [[y_pred, {"Type":("pragmatic_"+mode)}]]
            else:
                evaluation = evaluate_on_participant_data(y_pred, spec_video=i,
parameters={"Type":("pragmatic_"+mode)}, video_length=video_length)
            performance.extend(evaluation)


    # Single value for all frames
    elif mode == "single_value":
        # Initiate prediction array
        y_pred = []
        # Loop through all rows with dial data
        for _ in range(video_length):
            # Append to prediction
            y_pred.append(single_predict_value)

        # Evaluate prediction and append results to list
        if(save_trajectories):
            evaluation = [[y_pred, {"Type":("pragmatic_"+mode),"Single predict value":
single_predict_value}]]
        else:
            evaluation = evaluate_on_participant_data(y_pred,
parameters={"Type":("pragmatic_"+mode),"Single predict value": single_predict_value},
video_length=video_length)
        performance.extend(evaluation)

    # Single value for all frames
    elif mode == "equal_probs":
        for _ in range(mont_carlo_count):
            # Initiate prediction array
            y_pred = []
            # Loop through all rows with dial data
            for _ in range(video_length):
                # Append to prediction
                y_pred.append(random.choices(population=[1,2,3,4,5,6], k=1)[0])

            # Evaluate prediction and append results to list
            if(save_trajectories):
                evaluation = [[y_pred, {"Type":("pragmatic_"+mode), "Monte Carlo Count":
mont_carlo_count}]]
            else:
                evaluation = evaluate_on_participant_data(y_pred,
parameters={"Type":("pragmatic_"+mode), "Monte Carlo Count": mont_carlo_count},
video_length=video_length)
            performance.extend(evaluation)

    # Return accuracies
    return performance
```

Appendix F

Periodic sampling model prediction script.

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import math
import random

from x_custom_functions import evaluate_on_participant_data, return_bandwidths

# samp_freq in Hz, sampl_dur in seconds
def predict_using_psm(mont_carlo_count, samp_freq, sampl_dur, evaluate_on_video=False,
video_length=None, save_trajectories=False):
    # Create random predictions
    performance = []

    # Sampling duration, assumed to be the same for all dials (s)
    D = np.linspace(sampl_dur, sampl_dur, 6)
    # Bandwidths (Hz)
    W = return_bandwidths()
    # Attentional demand (-)
    T = samp_freq*W*D
    # Normalized attentional demand
    p = T/T.sum()

    # Go through each instance
    for _ in range(mont_carlo_count):
        y_pred = []

        # Select a number of values using normalized attentional demand
        selected_values = random.choices(population=[1,2,3,4,5,6], weights=p, k=video_length)

        # Go through each selected dial
        for value in selected_values:
            # Based on sampling duration, append to prediction
            for i in range(int(D[value-1]*50)):
                # Break if X frames are reached
                if len(y_pred) == video_length: break
                y_pred.append(value)

            # Break if X frames are reached
            if len(y_pred) == video_length: break

        # Evaluate the prediction
        if(evaluate_on_video!=False):
            if(save_trajectories):
                evaluation = [[y_pred, {"Type":"PSM", "Monte Carlo Count": mont_carlo_count,
"Sampling frequency": samp_freq, "Sampling duration": sampl_dur}]]
            else:
                evaluation = evaluate_on_participant_data(y_pred, spec_video=evaluate_on_video,
parameters={"Type":"PSM", "Monte Carlo Count": mont_carlo_count, "Sampling frequency": samp_freq,
"Sampling duration": sampl_dur}, video_length=video_length)
        else:
            if(save_trajectories):
                evaluation = [[y_pred, {"Type":"PSM", "Monte Carlo Count": mont_carlo_count,
"Sampling frequency": samp_freq, "Sampling duration": sampl_dur}]]
```

```
            else:
                evaluation = evaluate_on_participant_data(y_pred, parameters={"Type":"PSM", "Monte 
Carlo Count": mont_carlo_count, "Sampling frequency": samp_freq, "Sampling duration": sampl_dur}, 
video_length=video_length)
        performance.extend(evaluation)

    return performance
```

Appendix G

Random constrained sampling model prediction script.

```python
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import random

from x_custom_functions import evaluate_on_participant_data, return_bandwidths

def predict_using_rcm(mont_carlo_count, sampl_dur, interval_correction, evaluate_on_video=False,
video_length=None, save_trajectories=False):
    # W - Taken from dial data (Hz)
    W = return_bandwidths()
    # D - Guesstimated using parameter (s)
    D = np.linspace(sampl_dur, sampl_dur, 6)
    # P - RCM
    P_i = W/W.sum()

    # Variance of interval between fixations (s)
    def add_interval_variance(int_arr):
        return np.random.normal(0,int_arr/10,6)

    # Interval between fixations (s)
    mu_interval = D/P_i

    # Fixation duration (s)
    D0 = D/(1-P_i)

    # Interval between fixations with variance and using the interval correction parameter
    in_mu_interval = np.rint(interval_correction*((mu_interval +
add_interval_variance(mu_interval))*50))

    # A tracking matrix to keep track of the intervals and how much time left until a dial enters a
imaginary queue (when frames_to_watch = 0, a dial is in the queue)
    init_tracking_matrix = pd.DataFrame(data=[[
        1,
        2,
        3,
        4,
        5,
        6],[
        in_mu_interval[0],
        in_mu_interval[1],
        in_mu_interval[2],
        in_mu_interval[3],
        in_mu_interval[4],
        in_mu_interval[5]]
            ]).T
    init_tracking_matrix.columns = ["dial", "frames_to_watch"]


    # Initiate evaluation matrices
    performance = []

    # Go through each instance
    for _ in range(mont_carlo_count):
        y_pred = []

        # Create an array with randomly selected dials using the the P_i probabilities
        selected_values = random.choices(population=[1,2,3,4,5,6], weights=P_i, k=video_length)

        # Create a copy of the tracking matrix for this instance
        tracking_matrix = init_tracking_matrix.copy()

        # Loop through each randomly selected value
        for count, value in enumerate(selected_values):
            # If there is a dial in the queue
            if(tracking_matrix.loc[value-1].frames_to_watch!=0 and
len(tracking_matrix[tracking_matrix["frames_to_watch"]==0])>0):
                # Skip this dial and try another dial until the dial in the queue is selected
                if(tracking_matrix.loc[value-1].frames_to_watch!=0): continue
            else:
                # Continue with this dial and reset the frames counter because the dial is now being
fixated on
                tracking_matrix.loc[value-1].frames_to_watch =
np.rint(interval_correction*((mu_interval[value-1] +
add_interval_variance(mu_interval)[value-1])*50))

            # Get fixation duration for current dial
            fix_duration = round(D0[value-1]*50,0)
            # Loop through the fixations
            for _ in range(int(fix_duration)):
                # End the prediction if there are X frames
                if len(y_pred) == video_length: break

                # Check if dial is already at 0
                for xk in range(1,7):
                    if(tracking_matrix.loc[xk-1].frames_to_watch!=0):
                        # Otherwise subtract 1 frame
                        tracking_matrix.loc[xk-1].frames_to_watch-=1
                # Add dial to prediction array
                y_pred.append(value)
            # End the prediction if there are X frames
            if len(y_pred) == video_length: break
        # Evaluate the prediction
        par = {"Type":"RCM", "Monte Carlo Count": mont_carlo_count, "Interval correction":
interval_correction, "Sampling duration": sampl_dur}
        if(evaluate_on_video!=False):
            if(save_trajectories):
                evaluation = [[y_pred, par]]
            else:
                evaluation = evaluate_on_participant_data(y_pred, spec_video=evaluate_on_video,
parameters=par, video_length=video_length)
        else:
            if(save_trajectories):
                evaluation = [[y_pred, par]]
            else:
                evaluation = evaluate_on_participant_data(y_pred, parameters=par,
video_length=video_length)
        performance.extend(evaluation)

    return performance
```

Appendix H

Conditional sampling model prediction script.

```python
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from scipy.stats import norm
from sklearn.metrics import accuracy_score
import random

from x_custom_functions import evaluate_on_participant_data, return_bandwidths

# Folder where data is located
input_human_directory = "../2 Data/input_human/"

# Load dial data
dial_data = pd.read_parquet(input_human_directory + "dial_data_w_relative.parquet")

W = np.array([0.03, 0.05, 0.12, 0.2, 0.32, 0.48])
colors = ["green","red","blue","purple","fuchsia","orange"]

## Equation 8, 9 and 10 from Eisma 2020, this function returns the expected amount of time until the
dial will cross the threshold (in frames)
def exc_prob(dial_n, tau, W_i, L_i, Y_i,  dial_deviations, sampling_frequency):
    # The assumptions of the angle given time (tau) that has passed
    p_i = np.sin(sampling_frequency*np.pi*W_i*tau)/(sampling_frequency*np.pi*W_i*tau)
    mu = p_i*Y_i

    # Standard deviation of video x and dial y
    sigma_i = dial_deviations.loc[dial_n]

    # Uncertainty of dial angle
    sigma_hat = sigma_i * np.sqrt(1-p_i**2)

    # Probability of dial exceeding L_i
    P_exceeding = 1 - norm.cdf((L_i-mu)/sigma_hat)
    return P_exceeding*100, mu, mu+sigma_hat, mu-sigma_hat

def get_csm_results(dial_n, last_angle, given_limit, total_time, dial_deviations,
sampling_frequency):
    # Bandwidth of dial, constant
    W_i = W[dial_n-1]
    # Limit to calculate the probability of exceeding of, constant
    L_i = given_limit
    # Last reading angle of participant, variable
    Y_i = last_angle

    x = []
    y_exp = []
    y_exp_p = []
    y_exp_m = []
    y_prob = []
    for i, value in enumerate(np.linspace(0.0001, total_time, 126)):
        res = exc_prob(dial_n, value, W_i, L_i, Y_i, dial_deviations, sampling_frequency)
        x.append(value)
        y_prob.append(res[0])
        y_exp.append(res[1])
        y_exp_p.append(res[2])
        y_exp_m.append(res[3])
    result_prob = pd.DataFrame(columns=["tau","prob"],data=list(zip(x,y_prob)))
    max_value = result_prob.iloc[result_prob["prob"].idxmax(),:]

    data = pd.DataFrame(data=[x,y_exp_m,y_exp,y_exp_p]).T
    data.columns = ["x","y_exp_m","y_exp","y_exp_p"]
    data.set_index("x", inplace=True)

    res = data[np.sign(data).diff().ne(0)]
    try:
        res = res.dropna(axis = 0, how = 'all').iloc[1:].iloc[0]
        return int(round(res.name/(1/50),0))
    except:
        return False


def run_csm(initial_state, mont_carlo_count, video, double_crossing_wait, sampling_frequency,
evaluate_on_video=False, video_length=None, save_trajectories=False):
    double_crossing_wait = round(double_crossing_wait)
    # Get first value of dial angles, this assumes that the human sampler knows the initial states
of the dials (on frame 1)
    idd = dial_data[(dial_data["video_number"]==video) & (dial_data["frame_number"]==1)]
    a1=idd[idd["dial_watched"]==1]["watched_dial_angle"].iloc[0]
    a2=idd[idd["dial_watched"]==2]["watched_dial_angle"].iloc[0]
    a3=idd[idd["dial_watched"]==3]["watched_dial_angle"].iloc[0]
    a4=idd[idd["dial_watched"]==4]["watched_dial_angle"].iloc[0]
    a5=idd[idd["dial_watched"]==5]["watched_dial_angle"].iloc[0]
    a6=idd[idd["dial_watched"]==6]["watched_dial_angle"].iloc[0]

    # Calculate dial deviations (used in calculating the )
    dial_deviations =
dial_data[dial_data["video_number"]==1].groupby("dial_watched").std()["watched_dial_angle"]

    # Initiate tracking matrix
    tracking_matrix = pd.DataFrame(data=[[
        1,
        2,
        3,
        4,
        5,
        6],[
        a1,
        a2,
        a3,
        a3,
        a4,
        a6],[
        get_csm_results(1, a1, 0, 15, dial_deviations, sampling_frequency),
        get_csm_results(2, a2, 0, 10, dial_deviations, sampling_frequency),
        get_csm_results(3, a3, 0, 10, dial_deviations, sampling_frequency),
        get_csm_results(4, a4, 0, 10, dial_deviations, sampling_frequency),
        get_csm_results(5, a5, 0, 10, dial_deviations, sampling_frequency),
        get_csm_results(6, a6, 0, 10, dial_deviations, sampling_frequency),
            ]]).T
    tracking_matrix.columns = ["dial_n","last_watched_dial_angle","expected_frames_to_crossing"]
    tracking_matrix.set_index("dial_n", inplace=True)



    # Initiate evaluation matrices
    performance = []

    # Go through each instance
    for _ in range(mont_carlo_count):
        # Create copy of tracking matrix to track when to switch to which dial
        copy = tracking_matrix.copy()

        # Set first frame "prediction"
        y_pred = [initial_state]

        # Loop through the remaining X frames
        for i in range(1,video_length):
            # Select the last frame as the candidate for th
            candidate_next = y_pred[-1]

            # Loop through all dials
            for dial_n in range(1,7):
                # Check whether another frame is expected to cross the threshold
                if(copy.loc[dial_n].loc["expected_frames_to_crossing"] == 0):
                    # Replace candidate with dial that is expected to cross threshold
                    candidate_next = dial_n
                else:
                    pass
            # Subtract 1 frame for all dials in the tracking matrix
            copy["expected_frames_to_crossing"] -= 1
            # If there were were more than 1 dials with expected_frames_to_crossing = 0, replace the
other ones with 1 frame, so that that dial will also be sampled asap (double crossing wait in
frames)
            copy["expected_frames_to_crossing"] = np.where(copy["expected_frames_to_crossing"] < 0,
double_crossing_wait, copy["expected_frames_to_crossing"])

            # Add the selected candidate to the prediction
            y_pred.append(candidate_next)


            # Get the dial angle that the human sampler has seen last
            new_angle = dial_data[(dial_data["video_number"]==video) &
(dial_data["frame_number"]==i) &
(dial_data["dial_watched"]==y_pred[-1])]["watched_dial_angle"].iloc[0]

            # Determine tau len based on which dial is selected (a dial with high frequency requires
lower dial and vice versa)
            if(y_pred[-1]==1):
                tau_len = 12
            elif(y_pred[-1]==2):
                tau_len = 8
            elif(y_pred[-1]==3):
                tau_len = 4
            elif(y_pred[-1]==4):
                tau_len = 2
            elif(y_pred[-1]==5):
                tau_len = 2
            elif(y_pred[-1]==6):
                tau_len = 1


            # Replace expected_frames_to_crossing based on new_angle
            copy.loc[y_pred[-1]] = [new_angle, get_csm_results(y_pred[-1], new_angle, 0, tau_len,
dial_deviations, sampling_frequency)]

        # Evaluate the prediction
        par = {"Type":"CSM", "Monte Carlo Count": mont_carlo_count, "Initial state": initial_state,
"Video": video, "Sampling Frequency": sampling_frequency, "double_crossing_wait":
double_crossing_wait}
        if(evaluate_on_video!=False):
            if(save_trajectories):
                evaluation = [[y_pred, par]]
            else:
                evaluation = evaluate_on_participant_data(y_pred, spec_video=evaluate_on_video,
parameters=par, video_length=video_length)
        else:
            if(save_trajectories):
                evaluation = [[y_pred, par]]
            else:
                evaluation = evaluate_on_participant_data(y_pred, parameters=par,
video_length=video_length)
        performance.extend(evaluation)

    return performance
```

Appendix I

Custom functions.

```python
# This script uses legit_combos_{} and final_gaze_{} files in the input folders

# Folder where data is located
input_human_directory = "../2 Data/input_human/"
output_screen_directory = "../2 Data/output_screen/"
results_dir = "../4 Results/"

import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score, r2_score

# Load dial data
dial_data = pd.read_parquet(input_human_directory + "dial_data_w_relative.parquet")

def evaluate_on_crossings(video, y_pred):
    correct = 0
    wrong = 0
    # Do some magic to get the consecutive crossing frames
    piv_dat = dial_data[(dial_data["video_number"]==video) &
(dial_data["frame_number"]<=3000)].copy()
    piv_dat['value_grp'] = (piv_dat.watched_dial_angle.diff(1) != 0).astype('int').cumsum()
    piv_dat = piv_dat[piv_dat["watched_dial_angle"]==0]
    crossing_sessions = pd.DataFrame({'beginFrame' :
piv_dat.groupby('value_grp').frame_number.first(),
                  'endFrame' : piv_dat.groupby('value_grp').frame_number.last(),
                  'duration' : piv_dat.groupby('value_grp').size(),
                  'dial_no' :
piv_dat.groupby('value_grp').dial_watched.first()}).reset_index(drop=True)
    y_pred = pd.Series(y_pred).reset_index()
    y_pred = y_pred.set_index("index")
    y_pred.index += 1
    for index, row in crossing_sessions.iterrows():
        period = y_pred.loc[row['beginFrame']:row['endFrame']]
        res_count = len(period[period[0]==row["dial_no"]])
        if res_count > 0:
            correct += 1
        else:
            wrong += 1
    return correct/(correct+wrong)*100

def return_bandwidths():
    return np.array([0.03, 0.05, 0.12, 0.20, 0.32, 0.48])

def evaluate_on_participant_data(y_pred, spec_video=None, parameters=None, video_length=None):
    performance = []

    # Load legit combos
    legit_combos = pd.DataFrame(pd.read_csv(input_human_directory +
"legit_combos_{}.csv".format(video_length)))
    legit_combos = legit_combos['0'].str[1:-1].str.split(',', expand=True).astype(int)

    # Load participant data per frame
    participant_data = pd.read_parquet(input_human_directory +
"final_gaze_{}.parquet".format(video_length))
```

```
    # Go through all legit combos
    for index,combo in legit_combos.iterrows():

        # If spec_video is defined, only compare with a specific video number
        if spec_video!=None and combo[1]==spec_video:
            # Determine y_true of participant
            if(video_length==3000):
                y_true = participant_data[(participant_data["video_number"]==combo[2]) &
(participant_data["setup"]==combo[1]) &
(participant_data["participant_number"]==combo[0])][["frame_number",
"dial_watched"]]["dial_watched"].tolist()
            else:
                y_true = participant_data[(participant_data["video_number"]==combo[0]) &
(participant_data["participant_number"]==combo[1])][["frame_number",
"dial_watched"]]["dial_watched"].tolist()
            # Calculate accuracy and append to list
            acc = accuracy_score(y_true, y_pred)

            # Determine distribution differences per dial (pred-true)
            y_true_counts = [y_true.count(1), y_true.count(2), y_true.count(3), y_true.count(4),
y_true.count(5), y_true.count(6)]
            y_pred_counts = [y_pred.count(1), y_pred.count(2), y_pred.count(3), y_pred.count(4),
y_pred.count(5), y_pred.count(6)]
            di = r2_score(y_true_counts, y_pred_counts)

            # Count the difference in distribution over all dials and subtract a 1/6th chance of
participant having the intention to go to the correct dial
            diff = (pd.Series(y_true_counts)-pd.Series(y_pred_counts)).abs().sum()/2
            if(diff<0):
                diff=0

            # Determine the amount of times the gaze was on the correct dial which was crossing the
threshold (this does not take into account the spacebar presses of experimental data!)
            sb_chance_pred = evaluate_on_crossings(combo[2],y_pred)
            sb_chance_true = evaluate_on_crossings(combo[2],y_true)

            # Append to performance
            if(video_length==3000):
                performance.append([combo[0], combo[1], combo[2], acc, di, diff, sb_chance_true,
sb_chance_pred, parameters])
            else:
                performance.append([combo[0], combo[1], acc, di, diff, sb_chance_true,
sb_chance_pred, parameters])



        elif spec_video==None:
            # Determine y_true of participant
            if(video_length==3000):
                y_true = participant_data[(participant_data["video_number"]==combo[2]) &
(participant_data["setup"]==combo[1]) &
(participant_data["participant_number"]==combo[0])][["frame_number",
"dial_watched"]]["dial_watched"].tolist()
            else:
                y_true = participant_data[(participant_data["video_number"]==combo[0]) & 
(participant_data["participant_number"]==combo[1])][["frame_number",
"dial_watched"]]["dial_watched"].tolist()
            # Calculate accuracy and append to list
            acc = accuracy_score(y_true, y_pred)


            # Determine distribution differences per dial (pred-true)
            y_true_counts = [y_true.count(1), y_true.count(2), y_true.count(3), y_true.count(4),
y_true.count(5), y_true.count(6)]
            y_pred_counts = [y_pred.count(1), y_pred.count(2), y_pred.count(3), y_pred.count(4),
y_pred.count(5), y_pred.count(6)]
            di = r2_score(y_true_counts, y_pred_counts)

            # Count the difference in distrubtion over all dials and subtract a 1/6th chance of
participant having the intention to go to the correct dial
            diff = (pd.Series(y_true_counts)-pd.Series(y_pred_counts)).abs().sum()/2
            if(diff<0):
                diff=0

            # Determine the amount of times the gaze was on the correct dial which was crossing the
threshold (this does not take into account the spacebar presses of experimantal data!)
            sb_chance_pred = evaluate_on_crossings(combo[2],y_pred)
            sb_chance_true = evaluate_on_crossings(combo[2],y_true)

            # Append to performance
            if(video_length==3000):
                performance.append([combo[0], combo[1], combo[2], acc, di, diff, sb_chance_true,
sb_chance_pred, parameters])
            else:
                performance.append([combo[0], combo[1], acc, di, diff, sb_chance_true,
sb_chance_pred, parameters])
    # Return accuracies list
    return performance
```

Appendix J

Data analysis script.

```python
#!/usr/bin/env python
# coding: utf-8

# ### Load libraries, directory names, general files and general functions
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import json
import seaborn as sns
from os import listdir
from os.path import isfile, join
import scipy.io
import ast
import scipy

import matplotlib as mpl
# Set DPI for all figures
mpl.rcParams['figure.dpi'] = 60

# Initiate some variables and data
input_human_directory = "../2 Data/input_human/"
output_screen_directory = "../2 Data/output_screen/"
results_dir = "../4 Results/"
dials = pd.read_parquet(input_human_directory + "dial_data_w_relative.parquet")
# Limit dial data to 60s/3000frames
dials = dials[dials["frame_number"]<=3000]
# Load gaze data
dials_watched = pd.read_parquet(input_human_directory + "final_gaze_3000.parquet")

# Function to round to nearest 5th
def custom_round(x, base=5):
    try:
        return int(base * round(float(x)/base))
    except:
        return 0

# Functions to process parameter data
def get_values_1(row):
    try:
        return json.loads(row["parameters"].replace("\'", "\""))["Single predict value"]
    except:
        return ""
def get_values_2(row):
    try:
        return json.loads(row["parameters"].replace("\'", "\""))["Monte Carlo Count"]
    except:
        return ""

import session_info
session_info.show()


# ### Load predicted trajectories and results
all_data = []
```

```python
all_data.append(pd.read_csv(results_dir + "results_pragmatic_3000_True.csv"))
all_data.append(pd.read_csv(results_dir + "results_psm_3000_True.csv"))
all_data.append(pd.read_csv(results_dir + "results_rcm_3000_True.csv"))
all_data.append(pd.read_csv(results_dir + "results_csm_3000_True.csv"))
all_data = pd.concat(all_data)
all_data.columns = ["y_pred", "parameters"]
all_data["Type"] = all_data.apply(lambda row: json.loads(row["parameters"].replace("\'",
"\""))["Type"],axis=1)


datasets = []
datasets_types = ["pragmatic", "psm", "rcm", "csm"]
prediction_types = ["pragmatic_single_value", "pragmatic_equal_probs", "pragmatic_sp",
"pragmatic_da", "pragmatic_ttc", "pragmatic_spdd", "pragmatic_dadd", "psm", "rcm", "csm"]
prediction_types_labels = ["pragmatic_ttc", "pragmatic_da", "pragmatic_sp", "pragmatic_spdd",
"pragmatic_single_value", "pragmatic_equal_probs", "pragmatic_dadd", "psm", "rcm", "csm"]
for type_un in all_data["Type"].unique():
    tmp_res_2 = all_data[all_data["Type"]==type_un]
    length_2 = len(tmp_res_2)
    tmp_res_2 = pd.DataFrame(tmp_res_2["y_pred"].apply(ast.literal_eval).tolist()).T
    tmp_res_2.index = range(1,3001)

    tmp_res_2 = tmp_res_2.reset_index()
    tmp_res_2 = tmp_res_2.rename(columns={'index': 'frame_number'})
    tmp_res_2 = pd.melt(tmp_res_2, id_vars='frame_number', value_vars=range(0,length_2))
    tmp_res_2 = tmp_res_2.rename(columns={'variable': 'prediction_number', 'value': 'dial_watched'})
    tmp_res_2["method"] = type_un
    datasets.append(tmp_res_2)

# Load Pragmatic Results
results_pragmatic = pd.read_csv(results_dir + "results_pragmatic_3000.csv")
results_pragmatic.columns = ["participant", "setup", "video", "accuracy", "r2",
"distribution_error", "sb_chances_true", "sb_chances_pred", "parameters"]

results_pragmatic["Type"] = results_pragmatic.apply(lambda row:
json.loads(row["parameters"].replace("\'", "\""))["Type"],axis=1)
results_pragmatic["Single predict value"] = results_pragmatic.apply(get_values_1, axis=1)
results_pragmatic["Monte carlo count"] = results_pragmatic.apply(get_values_2, axis=1)
results_pragmatic["all"] = ""

# Load PSM Results
results_psm = pd.read_csv(results_dir + "results_psm_3000.csv")
results_psm.columns = ["participant", "setup", "video", "accuracy", "r2", "distribution_error",
"sb_chances_true", "sb_chances_pred", "parameters"]

results_psm["Sampling frequency"] = results_psm.apply(lambda row:
json.loads(row["parameters"].replace("\'", "\""))["Sampling frequency"],axis=1)
results_psm["Sampling duration"] = results_psm.apply(lambda row:
json.loads(row["parameters"].replace("\'", "\""))["Sampling duration"],axis=1)
results_psm["all"] = ""


# Load RCM Results
results_rcm = pd.read_csv(results_dir + "results_rcm_3000.csv")
results_rcm.columns = ["participant", "setup", "video", "accuracy", "r2", "distribution_error",
"sb_chances_true", "sb_chances_pred", "parameters"]
```

```python
results_rcm["Interval correction"] = results_rcm.apply(lambda row:
json.loads(row["parameters"].replace("\'", "\""))["Interval correction"],axis=1)
results_rcm["Sampling duration"] = results_rcm.apply(lambda row:
json.loads(row["parameters"].replace("\'", "\""))["Sampling duration"],axis=1)
results_rcm["all"] = ""


# Load CSM Results
results_csm = pd.read_csv(results_dir + "results_csm_3000.csv")
results_csm.columns = ["participant", "setup", "video", "accuracy", "r2", "distribution_error",
"sb_chances_true", "sb_chances_pred", "parameters"]
results_csm = results_csm[results_csm.parameters.str.contains('CSM')].copy()
results_csm["Video_csm"] = results_csm.apply(lambda row: json.loads(row["parameters"].replace("\'",
"\""))["Video"],axis=1)
results_csm["Sampling Frequency"] = results_csm.apply(lambda row:
json.loads(row["parameters"].replace("\'", "\""))["Sampling Frequency"],axis=1)
results_csm["Initial state"] = results_csm.apply(lambda row:
json.loads(row["parameters"].replace("\'", "\""))["Initial state"],axis=1)
results_csm["double_crossing_wait"] = results_csm.apply(lambda row:
json.loads(row["parameters"].replace("\'", "\""))["double_crossing_wait"],axis=1)
results_csm["all"] = ""


# ### Plotting
# #### Figure 6
# Initiate variables
resultsTable6 = pd.DataFrame(columns=['Method', 'Plot', 'Intercept', 'Slope', 'R'])
freqs = [0.03, 0.05, 0.12, 0.20, 0.32, 0.48]
setups = ["FV", "GCV"]
sns.set(rc = {'figure.figsize':(15,6)})
ssize = 20

fig, axes = plt.subplots(1, 3)
plt.subplots_adjust(wspace=0.4, hspace=0.01)

# Go trhough both setups
for setup in range(1,3):
    # Filter gaze data on setup
    dials_watched_six =  dials_watched[dials_watched["setup"]==setup].copy()
    # Create groups of when the same dial is consequently being watched
    dials_watched_six['vl_grop'] = (dials_watched_six.dial_watched.diff(1) !=
0).astype('int').cumsum()

    result1 = []
    for dial in range(1,7):
        a = dials_watched_six[["dial_watched","vl_grop"]]

        # Total seconds of participant data
        total_secs = len(dials_watched_six.groupby(["video_number", "participant_number",
"setup"]).count().reset_index())*60
        # Add fixation filter to remove groups which have a duration shorter than 40ms (2 frames)
        fixation_filter = a[a["dial_watched"]==dial].groupby("vl_grop").count()
        fixation_filter = fixation_filter[fixation_filter["dial_watched"]>=2].reset_index()

        # Filter on dial watched and group that is 40ms long at minimum, then count how many groups
there are and divide through the total amount of time
```

```
        result1.append(len(a[(a["dial_watched"]==dial) &
(a["vl_grop"].isin(fixation_filter["vl_grop"]))]["vl_grop"].unique().tolist())/total_secs)

    # Plotting results
    axes[0].plot(freqs, result1, "o-", label=setups[setup-1])
    axes[0].legend(fontsize=ssize)
    axes[0].set_xlabel("Signal bandwidth (Hz)", size=ssize)
    axes[0].set_ylabel("Glance rate (Hz)", size=ssize)

    axes[0].tick_params(axis='both', which='major', labelsize=ssize)

##############

    # Filter gaze data on setup
    dials_watched_six =  dials_watched[dials_watched["setup"]==setup].copy()
    # Count amount of times a dial is watched
    total_watch_time_pd = dials_watched_six.groupby(["dial_watched"]).count().iloc[1:]

    # Count the amount of frames: videos*3000 frames = total frames
    total_sec = len(dials_watched_six.groupby(["video_number", "participant_number",
"setup"]).count().reset_index())*3000
    # Calculate proportion frames per dial divided by total frames
    prop_result = total_watch_time_pd["setup"]/total_sec
    # Plot results
    axes[1].plot(freqs, prop_result*100, 'o-', label=setups[setup-1])
    axes[1].legend(fontsize=ssize)
    axes[1].set_xlabel("Signal bandwidth (Hz)", size=ssize)
    axes[1].set_ylabel("Percent AOI (% of time)", size=ssize)
    axes[1].tick_params(axis='both', which='major', labelsize=ssize)

##############

    # Filter gaze data on setup
    dials_watched_six =  dials_watched[dials_watched["setup"]==setup].copy()
    result2 = []
    # Create groups of when the same dial is consequently being watched
    dials_watched_six['vl_grop'] = (dials_watched_six.dial_watched.diff(1) !=
0).astype('int').cumsum()

    # Loop through all dials
    for dial in range(1,7):
        # Count total frames per dial per group
        count_dial_frames = dials_watched_six.groupby(["vl_grop",
"dial_watched"]).count().reset_index()
        # Fixation filter
        count_dial_frames = count_dial_frames[count_dial_frames["frame_number"]>=2]
        # Select dial
        count_dial_frames = count_dial_frames[count_dial_frames["dial_watched"]==dial]
        # Append the mean duration per group (divide by 50 to get seconds)
        result2.append(count_dial_frames["setup"].mean()/50)

    # Plot
    axes[2].plot(freqs, result2, "o-", label=setups[setup-1])
    axes[2].legend(fontsize=ssize)
    axes[2].set_xlabel("Signal bandwidth (Hz)", size=ssize)
    axes[2].set_ylabel("Mean glance duration (s)", size=ssize)
    axes[2].tick_params(axis='both', which='major', labelsize=ssize)

    # Logging
    res1_reg = scipy.stats.linregress(freqs, result1)
    prop_res_reg = scipy.stats.linregress(freqs, prop_result*100)
    res2_reg = scipy.stats.linregress(freqs, result2)
    resultsTable6 = resultsTable6.append({'Method': setups[setup-1], 'Plot': 'Glance rate',
'Intercept': round(res1_reg.intercept,2), 'Slope': round(res1_reg.slope,2), 'R':
round(res1_reg.rvalue,2)}, ignore_index=True)
    resultsTable6 = resultsTable6.append({'Method': setups[setup-1], 'Plot': 'Percent AOI',
'Intercept': round(prop_res_reg.intercept,2), 'Slope': round(prop_res_reg.slope,2), 'R':
round(scipy.stats.linregress(freqs, prop_result)[2],2)}, ignore_index=True)
    resultsTable6 = resultsTable6.append({'Method': setups[setup-1], 'Plot': 'Mean glance duration',
'Intercept': round(res2_reg.intercept,2), 'Slope': round(res2_reg.slope,2), 'R':
round(scipy.stats.linregress(freqs, result2)[2],2)}, ignore_index=True)

display(resultsTable6)
plt.savefig("../5 Reporting/Figures/fig6_participants.png", dpi=600, bbox_inches = "tight")


def run_plot_ab(method="pragmatic"):
    # Initiate variables
    freqs = [0.03, 0.05, 0.12, 0.20, 0.32, 0.48]
    setups = ["Setup A", "Setup B"]
    sns.set(rc = {'figure.figsize':(15,6)})
    ssize = 20

    # Cosmetics
    legend_labels_tmp = [w.replace('pragmatic_', '') for w in prediction_types_labels]
    legend_labels_tmp = [w.replace('single_value', 'sv') for w in legend_labels_tmp]
    legend_labels_tmp = [w.replace('equal_probs', 'ep') for w in legend_labels_tmp]

    fig, axes = plt.subplots(1, 3)
    plt.subplots_adjust(wspace=0.4, hspace=0.01)

    resultsTable6 = pd.DataFrame(columns=['Method', 'Plot', 'Intercept', 'Slope', 'R'])

    # Go through all methods
    for i in range(len(datasets)):
        if(method=="pragmatic"):
            if(i>6):
                continue

        elif(method=="senders"):
            if(i<=6):
                continue
        # Skip methods which did not show significant results
        if(legend_labels_tmp[i]=="sv" or legend_labels_tmp[i]=="ep" or legend_labels_tmp[i]=="spdd"
or legend_labels_tmp[i]=="dadd"):
            continue

        dials_watched_six =  datasets[i].copy()
        # Create groups of when the same dial is consequently being watched
        dials_watched_six['vl_grop'] = (dials_watched_six.dial_watched.diff(1) !=
0).astype('int').cumsum()
```

```
        result1 = []
        for dial in range(1,7):
            b = dials_watched_six[["dial_watched","vl_grop"]]

            # Total seconds of participant data
            total_secs = 
len(dials_watched_six.groupby(["prediction_number"]).count().reset_index())*60
            # Add fixation filter to remove groups which have a duration shorter than 40ms (2 
frames)
            fixation_filter = b[b["dial_watched"]==dial].groupby("vl_grop").count()
            fixation_filter = fixation_filter[fixation_filter["dial_watched"]>=2].reset_index()

            # Filter on dial watched and group that is 40ms long at minimum, then count how many 
groups there are and divide through the total amount of time
            result1.append(len(b[(b["dial_watched"]==dial) & 
(b["vl_grop"].isin(fixation_filter["vl_grop"]))]["vl_grop"].unique().tolist())/total_secs)

        # Plotting results
        axes[0].plot(freqs, result1, "o-", label=legend_labels_tmp[i])
        axes[0].set_xlabel("Signal bandwidth (Hz)", size=ssize)
        axes[0].set_ylabel("Glance rate (Hz)", size=ssize)
        axes[0].tick_params(axis='both', which='major', labelsize=ssize)


        ######################

        dials_watched_six =  datasets[i].copy()

        # Count amount of times a dial is watched
        total_watch_time_pd = dials_watched_six.groupby(["dial_watched"]).count()
        # Count the amount of frames: videos*3000 frames = total frames
        total_sec = len(dials_watched_six.groupby(["prediction_number"]).count().reset_index())*3000
        # Calculate proportion frames per dial divided by total frames
        prop_result = total_watch_time_pd["prediction_number"]/total_sec
        # Plot results
        axes[1].plot(freqs, prop_result*100, 'o-', label=legend_labels_tmp[i])
        axes[1].set_xlabel("Signal bandwidth (Hz)", size=ssize)
        axes[1].set_ylabel("Percent AOI (% of time)", size=ssize)
        axes[1].tick_params(axis='both', which='major', labelsize=ssize)


        ###################

        dials_watched_six =  datasets[i].copy()
        result2 = []
        # Create groups of when the same dial is consequently being watched
        dials_watched_six['vl_grop'] = (dials_watched_six.dial_watched.diff(1) != 
0).astype('int').cumsum()

        # Loop through all dials
        for dial in range(1,7):
            # Count total frames per dial per group
            count_dial_frames = dials_watched_six.groupby(["vl_grop", 
"dial_watched"]).count().reset_index()
            # Fixation filter
            count_dial_frames = count_dial_frames[count_dial_frames["frame_number"]>=2]
            # Select dial
            count_dial_frames = count_dial_frames[count_dial_frames["dial_watched"]==dial]
            # Append the mean duration per group (divide by 50 to get seconds)
            result2.append(count_dial_frames["prediction_number"].mean()/50)

        # Plot
        axes[2].plot(freqs, result2, "o-", label=legend_labels_tmp[i])
        axes[2].set_xlabel("Signal bandwidth (Hz)", size=ssize)
        axes[2].legend(bbox_to_anchor=(1, 1), fontsize=ssize)
        axes[2].set_ylabel("Mean glance duration (s)", size=ssize)
        axes[2].tick_params(axis='both', which='major', labelsize=ssize)

        # Logging
        res1_reg = scipy.stats.linregress(freqs, result1)
        prop_res_reg = scipy.stats.linregress(freqs, prop_result*100)
        res2_reg = scipy.stats.linregress(freqs, result2)
        resultsTable6 = resultsTable6.append({'Method': legend_labels_tmp[i], 'Plot': 'Glance rate',
'Intercept': round(res1_reg.intercept,2), 'Slope': round(res1_reg.slope,2), 'R':
round(res1_reg.rvalue,2)}, ignore_index=True)
        resultsTable6 = resultsTable6.append({'Method': legend_labels_tmp[i], 'Plot': 'Percent AOI',
'Intercept': round(prop_res_reg.intercept,2), 'Slope': round(prop_res_reg.slope,2), 'R':
round(scipy.stats.linregress(freqs, prop_result)[2],2)}, ignore_index=True)
        resultsTable6 = resultsTable6.append({'Method': legend_labels_tmp[i], 'Plot': 'Mean glance
duration', 'Intercept': round(res2_reg.intercept,2), 'Slope': round(res2_reg.slope,2), 'R':
round(scipy.stats.linregress(freqs, result2)[2],2)}, ignore_index=True)

    display(resultsTable6)
    plt.savefig("../5 Reporting/Figures/fig6_predictions_{}.png".format(method), dpi=600,
bbox_inches = "tight")

run_plot_ab("pragmatic")
run_plot_ab("senders")
# #### Figure 7
def run_plot_7():
    # Load the ranks which determine in which order videos en setups have been viewed
    ranks = pd.read_csv(input_human_directory + "legit_combos_ranks.csv")
    # Merge the ranks with gaze data
    dials_watched_with_ranks = dials_watched.merge(ranks, on=["participant_number", "video_number",
"setup"])
    freqs = ['0.03 Hz', '0.05 Hz', '0.12 Hz', '0.20 Hz', '0.32 Hz', '0.48 Hz']


    ssize = 60

    setups = ["FV", "GCV"]
    sns.set(rc = {'figure.figsize':(60,20)})

    fig, axes = plt.subplots(2, 14, sharey=True)
    plt.subplots_adjust(wspace=0, hspace=0.01)
    # Loop through all ranks and setups
    for rank in range(1,15):
        for setup in range(1,3):
            # Filter gaze data accordingly
            dials_watched_six =  dials_watched_with_ranks[(dials_watched_with_ranks["setup"]==setup)
& (dials_watched_with_ranks["Rank"]==rank)].copy()
            result = []
            frames = []
            # Use bins of 10 seconds to determine the time on AOI
            frame_bins = [[0, 10*50], [10*50,20*50], [20*50, 30*50], [30*50, 40*50], [40*50, 50*50],
[50*50, 60*50]]

            # For each frame count how many frames each dial is watched and divide by total amount
of frames  all dials are watched
            for f_bin in frame_bins:
                frame_watches = dials_watched_six[(dials_watched_six["frame_number"]>=f_bin[0]) &
(dials_watched_six["frame_number"]<f_bin[1])].groupby("dial_watched").count()
                frame_watched_prop = (frame_watches/frame_watches["setup"].sum()*100)[["setup"]]
                # Remove index 0
                frame_watched_prop = frame_watched_prop[frame_watched_prop.index!=0]
                proportions = [np.nan, np.nan, np.nan, np.nan, np.nan, np.nan]
                # Process dial counts into a list
                for row in frame_watched_prop.iterrows():
                    proportions[row[0]-1] = row[1][0]
                result.append(proportions)
                # Keep track of which frames are related by giving in the average timestamp of the
bin being used
                frames.append(round((f_bin[0]+f_bin[1])/2,0))

            # Plot each dial
            for d in range(1,7):
                axes[setup-1,rank-1].plot(frames,[row[d-1] for row in result], linewidth=8.0)
            axes[setup-1,rank-1].grid(False)
            # Some cosmetical code
            if(rank==1):
                axes[setup-1,rank-1].set_ylabel("Percent AOI \n(% of time)\n For
{}".format(setups[setup-1]), size=ssize)
            axes[setup-1,rank-1].tick_params(axis='both', which='major', labelsize=ssize)
            if(setup==2):
                axes[setup-1,rank-1].set_xticks([0,3000])
                axes[setup-1,rank-1].set_xticklabels([(rank-1)*60,""])
            if(setup==2 and rank==14):
                axes[setup-1,rank-1].set_xticklabels([(rank-1)*60,(rank)*60])

    fig.text(0.5, 0.05, 'Cumulative elapsed time (s)', ha='center', fontsize=ssize)
    fig.legend(freqs, loc='lower right', bbox_to_anchor=(0.8,-0.06), ncol=len(freqs),
bbox_transform=fig.transFigure, prop={'size': ssize})
run_plot_7()
plt.savefig("../5 Reporting/Figures/fig7.png", dpi=600, bbox_inches = "tight")


# #### Figure 9

def run_plot_9():
    sns.set(rc = {'figure.figsize':(20,12)})
    ssize = 20
    fig, axes = plt.subplots(2, 3, sharey=True)
    plt.subplots_adjust(wspace=0.15, hspace=0.01)
    freqs = ['0.03 Hz', '0.05 Hz', '0.12 Hz', '0.20 Hz', '0.32 Hz', '0.48 Hz']

    for mode in ["watched_dial_angle", "speed", "time_to_cross"]:
        # Set the upper and lower limit for the information to plot and select column to plot
        if mode=="watched_dial_angle":
            upper = 100
            lower = -100
            column = 0
            xlabel = "Pointer angle (deg)"
        elif mode=="speed":
            upper = 100
            lower = -100
            column = 1
            xlabel = "Pointer speed (deg/s)"
        elif mode=="time_to_cross":
            upper = 10
            lower = -10
            column = 2
            xlabel = "Time to crossing (s)"

        # Go through each setup, dial, video and participant
        for setup in range(1,3):
            dial_plot_data = []
            for dial_watched in range(1,7):
                dial_dfs = []
                for video in range(1,8):
                    for participant in range(1,34):
                        # Count the occurances of the dials per X and select video and dial
                        dial_tmp = dials.groupby(["video_number", "dial_watched",
mode]).count().loc[video].loc[dial_watched]
                        dial_tmp = dial_tmp[["frame_number"]]
                        dial_tmp.columns = ["dial_true"]
                        # Filter on lower and upper limit
                        dial_tmp  = dial_tmp[(dial_tmp.index >= lower) & (dial_tmp.index <= upper)]

                        # Get the participant data for specific loop
                        a = dials_watched[(dials_watched["video_number"]==video) &
(dials_watched["dial_watched"]==dial_watched) & (dials_watched["setup"]==setup) &
(dials_watched["participant_number"]==participant)]
                        # Combine with general dial data
                        a = a.merge(dials, on=["video_number","frame_number","dial_watched"])
                        # Also count the occurance of X for that dial
                        a = a.groupby([mode]).count()[["video_number"]]
                        a.columns = ["dial_true"]
                        # Filter on lower and upper limit
                        a  = a[(a.index >= lower) & (a.index <= upper)]
                        # Calculate time on AOI
                        result = a.div(dial_tmp, axis='index').fillna(0)
                        dial_dfs.append(result)

                # Combine dataframes and calculate the mean of all time on AOI's
                dial_plot_data.append(pd.concat(dial_dfs).groupby(level=0).mean()*100)

            # Plot
            for i in range(0,6):
                dial_plot_data[i].plot(ax=axes[setup-1,column])
            axes[setup-1,column].set_xlim(lower,upper)
            axes[setup-1,column].grid(True)
            if setup==2:
                axes[setup-1,column].set_xlabel(xlabel, size=ssize)
            else:
```

```python
                axes[setup-1,column].set_xlabel("", size=1)
                axes[setup-1,column].xaxis.set_ticklabels([])

            if column==0:
                if setup==1:
                    axes[setup-1,column].set_ylabel("Percent AOI (% of time) for FV", size=ssize)
                elif setup==2:
                    axes[setup-1,column].set_ylabel("Percent AOI (% of time) for GCV", size=ssize)
            axes[setup-1,column].tick_params(axis='both', which='major', labelsize=ssize)
            if column==2 and setup==2:
                axes[setup-1,column].legend(freqs, bbox_to_anchor=(1.5, 2), prop={'size': ssize})
            else:
                axes[setup-1,column].get_legend().remove()
run_plot_9()
plt.savefig("../5 Reporting/Figures/fig9.png", dpi=600, bbox_inches = "tight")


def run_plot_9_2():
    sns.set(rc = {'figure.figsize':(20,6)})
    setups = ["FV", "GCV"]
    ssize = 20
    fig, axes = plt.subplots(1, 3, sharey=True)
    plt.subplots_adjust(wspace=0.15, hspace=0.01)
    freqs = ['0.03 Hz', '0.05 Hz', '0.12 Hz', '0.20 Hz', '0.32 Hz', '0.48 Hz']

    for mode in ["watched_dial_angle", "speed", "time_to_cross"]:
        # Set the upper and lower limit for the information to plot and select column to plot
        if mode=="watched_dial_angle":
            upper = 100
            lower = -100
            column = 0
            xlabel = "Pointer angle (deg)"
        elif mode=="speed":
            upper = 100
            lower = -100
            column = 1
            xlabel = "Pointer speed (deg/s)"
        elif mode=="time_to_cross":
            upper = 10
            lower = -10
            column = 2
            xlabel = "Time to crossing (s)"

        # Go through each setup, dial, video and participant
        for setup in range(1,3):
            dial_plot_data = []
            for dial_watched in range(1,7):
                dial_dfs = []
                for video in range(1,8):
                    for participant in range(1,34):
                        # Count the occurances of the dials per X and select video and dial
                        dial_tmp = dials.groupby(["video_number", "dial_watched",
mode]).count().loc[video].loc[dial_watched]
                        dial_tmp = dial_tmp[["frame_number"]]
                        dial_tmp.columns = ["dial_true"]
                        # Filter on lower and upper limit
                        dial_tmp  = dial_tmp[(dial_tmp.index >= lower) & (dial_tmp.index <= upper)]

                        # Get the participant data for specific loop
                        a = dials_watched[(dials_watched["video_number"]==video) &
(dials_watched["dial_watched"]==dial_watched) & (dials_watched["setup"]==setup) &
(dials_watched["participant_number"]==participant)]
                        # Combine with general dial data
                        a = a.merge(dials, on=["video_number","frame_number","dial_watched"])
                        # Also count the occurance of X for that dial
                        a = a.groupby([mode]).count()[["video_number"]]
                        a.columns = ["dial_true"]
                        # Filter on lower and upper limit
                        a  = a[(a.index >= lower) & (a.index <= upper)]
                        # Calculate time on AOI
                        result = a.div(dial_tmp, axis='index').fillna(0)
                        dial_dfs.append(result)
                # Combine dataframes and calculate the mean of all time on AOI's
                dial_plot_data.append(pd.concat(dial_dfs).groupby(level=0).mean()*100)

            # Plot
            pd.concat(dial_plot_data).groupby(level=0).mean().plot(ax=axes[column])
            axes[column].legend(freqs)
            axes[column].set_xlim(lower,upper)
            axes[column].set_xlabel(xlabel, size=ssize)
            if column==0:
                axes[column].set_ylabel("Percent AOI (% of time)", fontsize=ssize)

            axes[column].legend(setups, prop={'size': ssize})
            axes[column].tick_params(axis='both', which='major', labelsize=ssize)

run_plot_9_2()
plt.savefig("../5 Reporting/Figures/fig9_2.png", dpi=600, bbox_inches = "tight")


def run_plot_bc(method="pragmatic"):
    ssize = 20

    sns.set(rc = {'figure.figsize':(20,6)})
    fig, axes = plt.subplots(1, 3, sharey=False)
    plt.subplots_adjust(wspace=0.15, hspace=0.01)
    freqs = ['0.03 Hz', '0.05 Hz', '0.12 Hz', '0.20 Hz', '0.32 Hz', '0.48 Hz']
    label_text = []
    for i in range(len(prediction_types)):
        if(method=="pragmatic"):
            if(i>6):
                continue
        elif(method=="senders"):
            if(i<=6):
                continue

        relevant_dataset = datasets[i].copy()

        lbl_t = relevant_dataset.iloc[0].method
        lbl_t = lbl_t.replace('pragmatic_', '')
        lbl_t = lbl_t.replace('single_value', 'sv')
        lbl_t= lbl_t.replace('equal_probs', 'ep')
        if(lbl_t=="sv" or lbl_t=="ep" or lbl_t=="spdd" or lbl_t=="dadd"):
            continue
        label_text.append(lbl_t.lower())

        for mode in ["watched_dial_angle", "speed", "time_to_cross"]:
            # Set the upper and lower limit for the information to plot and select column to plot
            if mode=="watched_dial_angle":
                upper = 100
                lower = -100
                column = 0
                xlabel = "Pointer angle (deg)"
            elif mode=="speed":
                upper = 100
                lower = -100
                column = 1
                xlabel = "Pointer speed (deg/s)"
            elif mode=="time_to_cross":
                upper = 10
                lower = -10
                column = 2
                xlabel = "Time to crossing (s)"

            dial_plot_data = []
            for dial_watched in range(1,7):
                dial_dfs = []
                for prediction_number in range(relevant_dataset["prediction_number"].max()+1):
                    for video in range(1,8):
                        # Count the occurances of the dials per X and select video and dial
                        dial_tmp = dials.groupby(["video_number", "dial_watched",
mode]).count().loc[video].loc[dial_watched]
                        dial_tmp = dial_tmp[["frame_number"]]
                        dial_tmp.columns = ["dial_true"]
                        # Filter on lower and upper limit
                        dial_tmp  = dial_tmp[(dial_tmp.index >= lower) & (dial_tmp.index <= upper)]

                        # Get the participant data for specific loop
                        a = relevant_dataset[(relevant_dataset["dial_watched"]==dial_watched) &
(relevant_dataset["prediction_number"]==prediction_number)]
                        # Combine with general dial data
                        a = a.merge(dials, on=["frame_number","dial_watched"])
                        # Also count the occurance of X for that dial

                        for video in range(1,8):
                            b = a[a["video_number"]==video]
                            # display(b)
                            b = b.groupby([mode]).count()[["video_number"]]
                            b.columns = ["dial_true"]
                            # Filter on lower and upper limit
                            b  = b[(b.index >= lower) & (b.index <= upper)]
                            # Calculate time on AOI
                            result = b.div(dial_tmp, axis='index').fillna(0)
                            dial_dfs.append(result)
                # Combine dataframes and calculate the mean of all time on AOI's
                dial_plot_data.append(pd.concat(dial_dfs).groupby(level=0).mean()*100)

            # Plot
            pd.concat(dial_plot_data).groupby(level=0).mean().plot(ax=axes[column])
            axes[column].set_xlim(lower,upper)
            axes[column].set_xlabel(xlabel, size=ssize)
            if column==0:
                axes[column].set_ylabel("Percent AOI (% of time)", size=ssize)
            if mode=="time_to_cross":
                axes[column].legend(label_text, bbox_to_anchor=(1.0, 1), fontsize=ssize)
            else:
                axes[column].get_legend().remove()
            axes[column].tick_params(axis='both', which='major', labelsize=ssize)

        print(i, end=" ")
    plt.savefig("../5 Reporting/Figures/fig9_predictions_{}.png".format(method), dpi=600,
bbox_inches = "tight")


run_plot_bc(method="pragmatic")
run_plot_bc(method="senders")

# #### Performance plot
### Load participant space bar presses and put the scores in scores list
# Load spacebar data
all_data = []
participant_tracks = [f for f in listdir(input_human_directory + "presses_v2") if
isfile(join(input_human_directory + "presses_v2", f))]

# Combine participant spacebar data
for file in participant_tracks:
    tmp = pd.read_csv(input_human_directory + "presses_v2/" + file, sep="\t")
    try:
        tmp['Session_Name_'] = tmp['Session_Name_'].str.replace('p','').astype("int")
        tmp['setups_str'].replace({"A": 1, "B": 2}, inplace=True)
    except:
        pass
    all_data.append((tmp))
all_data = pd.concat(all_data).reset_index().iloc[:,1:]

# Initiate scores list
scores = []
# Loop through each video, setup and participant
for video in range(1,8):
    for participant in all_data["Session_Name_"].unique():
        for setup in [1, 2]:
            # Load dial angles and get the crossing frames
            mat = scipy.io.loadmat(output_screen_directory + 'dial_angles_{}.mat'.format(video))
            data_dials = pd.DataFrame(mat["alldata"])
            crossing_frames = []
            # Go through all dials and find where there is a crossing (sign change)
            for i in range(0,6):
                crossing_frames.append(pd.DataFrame(np.where(np.diff(np.sign(data_dials[i])))[0]))
            # Combine all frames to a list
            zc = pd.concat(crossing_frames).sort_values(by=0).reset_index()
            zc = zc[0].tolist()
            # Remove crossings after 3000 frames
            zc = [i for i in zc if i <=3000]

            ## Get space bar presses of participant
            space_press = all_data[(all_data["Session_Name_"]==participant) &
(all_data["setups_str"]==setup) & (all_data["video_int"]==video)].copy()

            # Initiate performance measures
            correct = 0
            wrong = 0

            # Loop through each crossing frame
            for frame_number in zc:
                # Search for spacebar presses at that time
                space_bar_presses = space_press.copy().loc[(space_press["VAR_VIDEO_FRAME"] >=
frame_number-25) & (space_press["VAR_VIDEO_FRAME"] <= frame_number+25)]

                # Add as correct and remove spacebar press to prevent further use of that spacebar
press when there are spacebar presses found
                if(len(space_bar_presses)>0):
                    correct += 1
                    space_press.drop(space_bar_presses.index[0], inplace=True)
                else:
                    wrong += 1
            # Add the performance score to the scores list
            scores.append((participant, setup, video, round(correct/(correct+wrong)*100,2)))

# Clean dataframe
scores = pd.DataFrame(scores)
scores.columns = ["participant", "setup", "video", "performance"]
scores["all"] = ""

## Load one kind of results to get the sb_chances_true
results_pragmatic_tmp = pd.read_csv(results_dir + "results_pragmatic_3000.csv")
results_pragmatic_tmp.columns = ["participant", "setup", "video", "accuracy", "r2",
"distribution_error", "sb_chances_true", "sb_chances_pred", "parameters"]
# In this dataset the participants are occuring multiple times due to multiple predictions, remove
multiple occurances
results_pragmatic_tmp = results_pragmatic_tmp.drop_duplicates(subset = ['participant', 'video',
'setup'] )
results_pragmatic_tmp["all"] = ""


legend_labels = ["FV", "GCV"]
ssize = 20
sns.set(rc = {'figure.figsize':(15,5)})
fig, ax = plt.subplots(1,1,figsize=(5,5), sharey=True)
plt.subplots_adjust(wspace=0)

sns.violinplot(x='all', y='performance', hue='setup', kind="violin", data=scores, ax=ax, bw=0.2)
ax.set_xlabel("True \nperformance\n(n_FV={},\nn_GCV={})".format(len(scores[scores["setup"]==1]),
                                                                len(scores[scores["setup"]==2])),
fontsize=ssize)
ax.set_ylabel("Threshold crossings (%)", fontsize=ssize)
ax.set_ylim(0,100)
ax.get_legend().remove()

ax.legend()
ax.legend(handles=ax.legend_.legendHandles, labels=legend_labels, prop={'size': ssize})
ax.tick_params(axis='both', which='major', labelsize=ssize)

fig.savefig("../5 Reporting/Figures/spacebar_data_1.png", dpi=600, bbox_inches = "tight")

tmp_labels = ["pragmatic_single_value", "pragmatic_equal_probs", "pragmatic_sp", "pragmatic_da",
"pragmatic_ttc", "pragmatic_spdd", "pragmatic_dadd"]
resultsTableTaskPerf = pd.DataFrame(columns=['Method', 'Median', 'std'])
resultsTableTaskPerf = resultsTableTaskPerf.append({'Method': 'Performance setup 1',
                                                    'Median':
scores[scores["setup"]==1]["performance"].median(),
                                                    'std' :
scores[scores["setup"]==1]["performance"].std()}, ignore_index=True)
resultsTableTaskPerf = resultsTableTaskPerf.append({'Method': 'Performance setup 2',
                                                    'Median':
scores[scores["setup"]==2]["performance"].median(),
                                                    'std' :
scores[scores["setup"]==2]["performance"].std()}, ignore_index=True)
resultsTableTaskPerf.round(2)
```

Appendix K

Combinations of PSM parameters

Sampling frequency	Sampling duration
1.59375	0.46249999999999997
1.59375	0.46249999999999997
1.96875	0.38749999999999996
1.59375	0.38749999999999996
1.96875	0.38749999999999996
2.46875	0.5874999999999999
2.09375	0.5874999999999999
1.96875	0.46249999999999997
2.46875	0.5874999999999999
2.09375	0.2625
2.09375	0.2625
2.46875	0.2625
2.34375	0.5625
2.34375	0.5625
2.34375	0.2875
1.71875	0.2875
1.71875	0.5625
2.21875	0.4875
1.71875	0.2875
1.84375	0.4875
2.21875	0.3625
2.21875	0.4875
1.84375	0.3625
1.84375	0.3625
1.96875	0.23750000000000002
1.96875	0.3125
2.34375	0.23750000000000002
1.96875	0.3125
2.34375	0.23750000000000002
1.84375	0.4375
2.46875	0.4375
2.34375	0.3125
2.46875	0.5125
2.46875	0.5125
1.84375	0.5125
1.84375	0.4375
2.21875	0.21250000000000002
2.21875	0.21250000000000002
2.21875	0.3375
2.09375	0.3375
2.09375	0.21250000000000002
1.71875	0.5375
2.09375	0.3375
1.59375	0.5375
1.71875	0.4125
1.59375	0.5375
1.71875	0.4125
1.59375	0.4125
2.03125	0.2625
1.65625	0.5375
1.65625	0.2625
1.65625	0.5375
2.03125	0.5375
1.53125	0.46249999999999997
2.15625	0.46249999999999997
2.03125	0.2625
2.15625	0.3375
1.53125	0.3375

1.53125	0.46249999999999997
2.15625	0.3375
2.40625	0.4375
2.40625	0.4375
2.28125	0.3625
2.40625	0.3625
1.90625	0.5625
2.28125	0.3625
1.78125	0.5625
2.28125	0.4375
1.78125	0.23750000000000002
1.78125	0.5625
1.90625	0.23750000000000002
1.90625	0.23750000000000002
1.78125	0.2875
1.78125	0.2875
1.78125	0.3125
1.65625	0.3125
2.28125	0.5125
2.15625	0.5125
1.65625	0.3125
1.65625	0.2875
2.15625	0.5125
2.15625	0.4875
2.28125	0.4875
2.28125	0.4875
1.90625	0.21250000000000002
2.03125	0.38749999999999996
2.03125	0.38749999999999996
2.03125	0.21250000000000002
1.53125	0.4125
1.90625	0.21250000000000002
2.40625	0.4125
1.90625	0.38749999999999996
1.53125	0.5874999999999999
2.40625	0.4125
1.53125	0.5874999999999999
2.40625	0.5874999999999999

Combinations of RCM parameters

Interval correction	**Sampling duration**
0.96875	0.23750000000000002
1.15625	0.23750000000000002
0.96875	0.38749999999999996
1.15625	0.23750000000000002
0.96875	0.38749999999999996
1.15625	0.38749999999999996
1.46875	0.5874999999999999
1.46875	0.4375
0.65625	0.4375
0.65625	0.4375
0.65625	0.5874999999999999
1.46875	0.5874999999999999
0.71875	0.5375
0.71875	0.2875
1.40625	0.5375
1.40625	0.5375
0.71875	0.2875
1.21875	0.4875
1.40625	0.2875
1.21875	0.3375
0.90625	0.3375

0.90625	0.3375
1.21875	0.4875
0.90625	0.4875
0.78125	0.38749999999999996
0.59375	0.38749999999999996
0.59375	0.5375
0.78125	0.38749999999999996
0.59375	0.5375
0.78125	0.5375
1.09375	0.3375
1.09375	0.5874999999999999
1.28125	0.5874999999999999
1.28125	0.5874999999999999
1.28125	0.3375
1.09375	0.3375
0.53125	0.4875
0.53125	0.4875
0.84375	0.4875
0.84375	0.4375
0.53125	0.4375
0.84375	0.4375
1.34375	0.23750000000000002
1.34375	0.2875
1.03125	0.2875
1.03125	0.23750000000000002
1.03125	0.2875
1.34375	0.23750000000000002
0.65625	0.4125
0.65625	0.2625
1.34375	0.2625
1.34375	0.2625
0.65625	0.4125
1.15625	0.46249999999999997
1.15625	0.21250000000000002
1.34375	0.4125
0.84375	0.46249999999999997
0.84375	0.21250000000000002
0.84375	0.46249999999999997
1.15625	0.21250000000000002
0.90625	0.5125
0.90625	0.5625
1.09375	0.5625
1.09375	0.5625
1.40625	0.3125
1.09375	0.5125
0.90625	0.5125
1.40625	0.3625
0.59375	0.3625
0.59375	0.3625
0.59375	0.3125
1.40625	0.3125
0.78125	0.2625
0.78125	0.3125
0.71875	0.3125
0.71875	0.3125
0.78125	0.2625
1.28125	0.5125
0.71875	0.2625
1.28125	0.46249999999999997
1.21875	0.5125
1.28125	0.46249999999999997
1.21875	0.46249999999999997
1.21875	0.5125

0.53125	0.4125
0.96875	0.4125
0.53125	0.3625
0.96875	0.4125
0.53125	0.3625
0.96875	0.3625
1.03125	0.21250000000000002
1.03125	0.5625
1.03125	0.5625
1.46875	0.5625
1.46875	0.21250000000000002
1.46875	0.21250000000000002

Combinations of CSM parameters

Sampling Frequency	**Wait time**
1.96875	4.0
2.15625	2.0
1.96875	2.0
2.15625	2.0
2.15625	4.0
2.46875	7.0
1.96875	4.0
2.46875	5.0
2.46875	7.0
1.65625	5.0
1.65625	7.0
1.65625	5.0
2.40625	6.0
1.71875	2.0
1.71875	6.0
2.40625	6.0
2.21875	3.0
2.21875	5.0
1.71875	2.0
2.40625	2.0
1.90625	3.0
1.90625	3.0
2.21875	5.0
1.90625	5.0
1.59375	4.0
1.78125	4.0
1.59375	6.0
1.78125	4.0
1.78125	6.0
2.09375	7.0
2.09375	3.0
1.59375	6.0
2.28125	7.0
2.28125	7.0
2.28125	3.0
2.09375	3.0
1.53125	5.0
1.84375	5.0
1.53125	5.0
1.84375	5.0
2.34375	2.0
1.53125	5.0
1.84375	5.0
2.34375	2.0
2.03125	2.0
2.03125	2.0
2.34375	2.0

2.03125	2.0
2.34375	2.0
2.34375	2.0
1.65625	2.0
1.65625	4.0
1.65625	4.0
2.15625	5.0
2.15625	1.0
2.34375	4.0
2.15625	1.0
1.84375	5.0
1.84375	5.0
1.84375	1.0
1.90625	6.0
1.90625	6.0
2.09375	6.0
2.09375	6.0
1.90625	6.0
2.09375	6.0
2.40625	3.0
2.40625	3.0
1.59375	3.0
1.59375	3.0
2.40625	3.0
1.59375	3.0
1.71875	3.0
1.78125	3.0
1.78125	2.0
1.71875	3.0
2.28125	6.0
1.71875	2.0
1.78125	2.0
2.28125	5.0
2.21875	6.0
2.28125	5.0
2.21875	5.0
2.21875	6.0
1.53125	3.0
1.53125	4.0
1.96875	4.0
1.96875	4.0
1.96875	3.0
2.03125	6.0
1.53125	3.0
2.03125	1.0
2.46875	1.0
2.46875	1.0
2.03125	6.0
2.46875	6.0