Visual World Paradigm

A classical visual world study showing how people predict upcoming words with the help of Gazepoint eye tracker

Pritom Gogoi, Manpa Barman, Kapil Chander Mulchandani

2023-08-28

Abstract

The study presented in this paper explores the dynamics of predictive language processing through the visual world paradigm (VWP), a widely employed method in cognitive psychology. The primary objective of the research is to unwind how individuals anticipate or predict forthcoming words during the unfolding of the spoken instructions, leveraging the Gazepoint eye tracker for precise gaze pattern analysis. The investigation delves into the impact of competitor words on gaze patterns, to study the cognitive mechanisms underlying real-time language comprehension. Our experiment uses a collection of competitor words sharing phonetic or semantic similarities with the target, and validates the hypothesis that the existence of such competitors leads to an increased number of fixations on them, reflecting the participants' evolving predictions of the upcoming word.

Table of Contents

Download the pdf version.

Chapter 1

Introduction

1.1 Visual World Paradigm

The visual world paradigm is an experimental framework that investigates language processing by monitoring participants' eye movements while they interact with visual stimuli. Introduced by psychologists Richard Cooper and Thomas P. McDermott in the late 1990s, this paradigm have been continuosly refined and expanded, adapting it to different research questions and using advancements in eye-tracking technology to gain deeper insights into real-time language comprehension and visual attention processes. Through this framework the researchers try to simulate the integration of spoken language and visual information as they naturally occur in everyday situations so that we can draw inferences on the attention focus on specific objects in their visual display over time.

1.2 Objective of our project

Our research question is:

Does the presence of similar-sounding words influence our tendency to focus on those words apart from the target word as the word unfolds?

Our project is to study the nature of spoken word recognition as the word unfolds. Here we try to investigate the visual world paradigm by using the participants' eye movements which serve as an index of their ongoing language processing and interpretation.

We aim to explore two fundamental conclusions concerning spoken word recognition and the underlying models, based upon the established research in this domain:

- Spoken word recognition is dynamic in nature which suggests that listeners continuously update and refine their interpretations as more information becomes available.

- Spoken word recognition models make assumptions that multiple candidates compete for recognition during the unfolding of the spoken word.

Paul D.Allopenna, James S. Magnuson and Michael K. Tanenhaus in their paper "Tracking the Time Course of Spoken Word Recognition Using Eye Movements: Evidence for Continuous Mapping Models" investigated a similar structure of the experiment and found the following results:

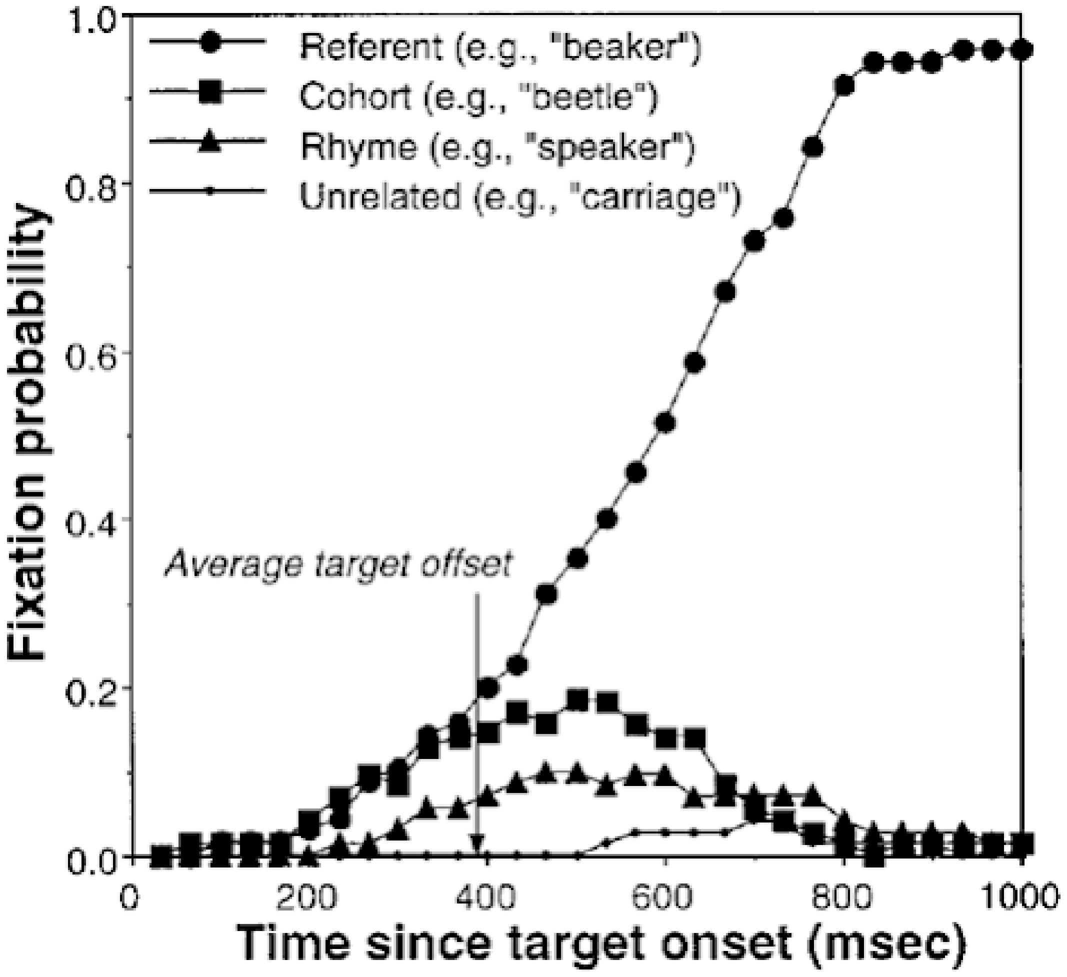


Figure 1.1: Probability of fixating on each item type over time in the full competitor condition

In this figure we have the probability of fixation on four words:

1. Referent (e.g beaker) : Target Word
2. Cohort (e.g beetle) : Similar Sounding Word
3. Rhyme (e.g speaker) : Rhyming word
4. Unrelated (e.g carriage) : Unrelated word to the rest (phonetically or semantically.)

In the beginning the participants hear [bi], which could be the beginning of beaker but also could be the beginning of beetle. So during the first 400 ms the particpants start looking at both of those words, more than they look at the others. After some time as they hear the [k] i.e. now they are hearing [bik], thus they discard their choice of beetle and stop looking at it. But by the time they've heard the whole word beaker, they might realize that beaker rhymes almost exactly with speaker and get confused about if they heard speaker at the very first place. For the last word carriage the pronunciation is totally unrelated to the target beaker, so there is a very less probability of the participant actually fixation at the unrelated word.

Through we try to replicate the results obtained by Paul D.Allopenna, James S. Magnuson and Michael K. Tanenhaus in their paper "Tracking the Time Course of Spoken Word Recognition Using Eye Movements: Evidence for Continuous Mapping Models" and validate their hypothesis.

Chapter 2

Experiment

2.1 Software and Hardware

- Software : The experiment is designed in OpenSesame which is a open source software to create experiments for psychology, neuroscience, and experimental economics.
- Language : Python was used along with the OpenSesame GUI to create the experiment. Libraries like Pandas, Numpy, Matplotlib were used to analyze the data. For backend processing in the OpenSesame GUI, PsychoPy was used.
- Eye Tracker: The experiment is conducted using the GazePoint GP3+ eye tracker. It is a binocular eye tracker that can record at 150 Hz The eye tracker is connected to the computer and the participants are seated at a distance of 60 cm from the screen. The experiment was conducted in a dimly lit laboratory setup to avoid any external light source that might interfere with the eye tracking.

2.2 Structure of the Stimulus

The experiment is designed to test the participants' ability to predict the upcoming word in a spoken instruction. The experiment is designed in such a way that the participants are presented with a visual display of four objects in a grid and they are instructed to click on the object that matches the spoken instruction.

A trial in the experiment means the response to one spoken instruction.

The following User Interface is presented to the participants:

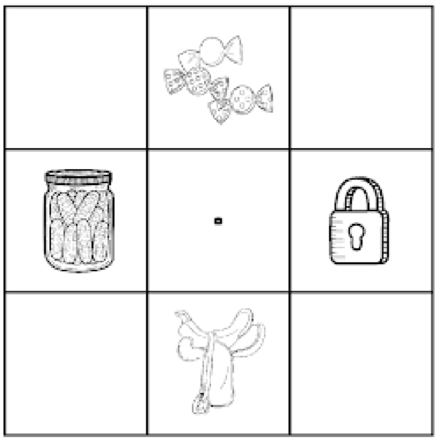


Figure 2.1: User Interface of one trial - In each trial we have a 3x3 grid structure with four objects in it. Each trial has four stimulus displayed. The four stimulus are a combination of referent, cohort, rhyme and unrelated words. All or atleast two of the combinations are present in each trial according to the condition set (Full competitor, Rhyme competitor, Unrelated competitor etc.), which will be referred in the later part of the report. The stimulus used are only line drawings of the objects, to remove anmibuity in the visual display. The dot in the center grid is the fixation point for the participants.

2.3 Design of the Experiment

The experiment follows the following structure in OpenSesame:

1. Introduction to the experiment : It contains some preliminary instructions for the participants to understand the experiment. It also mentions that each progression will require a mouse click. The foreground color of the text is set to black and the background color is set to white throughout the experiment.

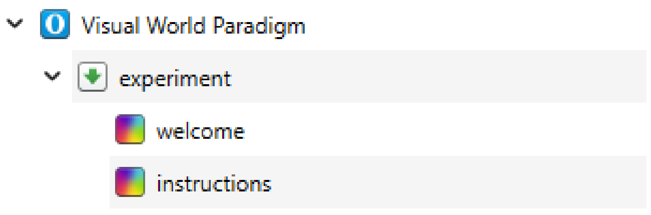
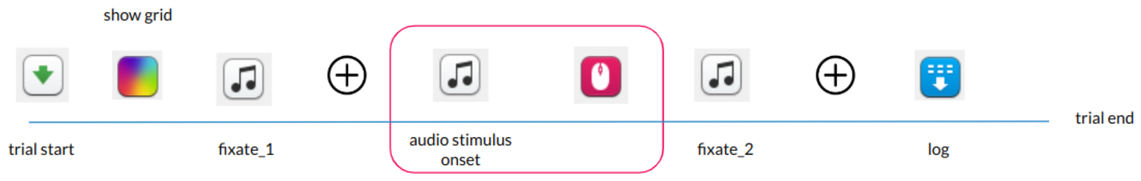


Figure 2.2: Introduction

2. Initialization of variables : The position variables (Top, Buttom, Left and Right) are initialized and are used to set the position of the objects in the grid. The pygaze module is also intialized to record the eye movements of the participants.

3. Trial Loop Items : This loop runs the experiment for 24 trials. The trial loop contains the following sequence of events:

- Fixation Cross : A fixation cross is displayed at the center of the screen. The fixation cross is a black dot on a white background. The fixation cross is displayed to ensure that the participants are looking at the center of the screen before the spoken instruction is played. The fixation cross is displayed using the sketchpad item in OpenSesame.
- Stimulus :
  1. The visual stimulus for each trial is loaded from the `stimuli.csv` file. The csv file contains information about the four objects that are displayed in the grid. The csv file contains the following information:
     - Stimulus : The name of the objects that is displayed in the grid.
     - Type : The type of the object. The type can be referent, cohort, rhyme or unrelated. The type of the object is used to determine the condition of the trial.
     - Condition : The condition of the trial.
     - Target : The target object that the participants have to click on. You can find the stimuli.csv file here.
  2. We also have accompaning audio stimuli for each trial. The audio is digitally recorded. The audio stimuli are recorded in the following format:
     - Instruction : 'FixateHow to run the analysisbject that the participants have to click on. You can find the audio stimuli here. Each response is captured with a mouse click. The mouse click is recorded and logged using the mouse_response item in OpenSesame.
- Logging : The onset and offset of the fixation cross and the stimulus (both audio and visual) is logged for each trail. We also log the position of the mouse click and the target object along with its position (top,right, buttom, left) that the participants clicked on. (More details will be proviided in the preprocessing and analysis section.)
- Gaze Contingency : Two fixation audio prompts which says 'Fixate at the center' and 'Now fixate at the center' and marks the beginning and end of one trail.The fixation audio prompt is played to ensure that the participants are looking at the center of the screen before the spoken instruction is played. This is implemented by introducing a delay of 1.1s after the prompt.



4. End of Experiment : The experiment ends with a thank you message for the participants.

2.4 Logic of the experiment

1. Stimuli are chosen as per the different pairs of sets included in the reference paper by Paul D.Allopenna, James S. Magnuson and Michael K. Tanenhaus.
   - For eg. One criteria of choosing those words are based on frequency per million words in the Kucera and Francis, 1967, corpus.
2. A fixation at any point on the screen indicates that the participant is paying attention to it. Thus we record the fixations throughout the experiment to deduce the attention of the pariticipant when we instruct them to fixate or look at a certain point of the canvas.
3. Noting timestamps of the samples is essential. Our experiment is designed to record how a participants attends to the stimuli and how they respond to the spoken instruction while the instruction is unfolding. Thus we record the timestamps of the samples to understand the chronology of the fixation events.
4. Fixations at the centre of the screen marks the start and end of a trial. This is to make sure we don't overlap the data of two trials while recording the data since each participant will have different response times and thus a fixed duration for each trial for timeout will not be feasible.

Chapter 3

Stimulus Design

Chapter 4

Stimuli Preprocessing

The stimuli collected needed to be preprocessed before they could be used in the experiment. The preprocessing steps are as follows:

1. Resizing the line drawings into 256x256 pixels. The OpenCV library was was used to read and resize the images.

   ```
   img = cv2.resize(img, size)
   ```

2. Converting the audio files to `.wav` format. The audio files were generated in `.mp3` format. The

   ```
   for file in $DIRPATH/*.mp3; do
       filename=$(basename "$file")
       filename="${filename%.*}"
       ffmpeg -i $file $OUTDIR/$filename.wav
   done
   ```

3. A trailing silence after each audio was observed which would affect the response time of the participants. The silence was removed using the `pydub` library.

   ```
   def detect_leading_silence(sound, silence_threshold, chunk_size=10):
       trim_ms = 0 # ms
       while sound[trim_ms:trim_ms+chunk_size].dBFS < silence_threshold:
           trim_ms += chunk_size
       return trim_ms
   ```

The function analyzes an audio snippet to find the duration of the silence at the beginning of the signal. It iterates over chunks of the audio and measuring the volume (dBFS) in each chunk until the volume exceeds the provided silence threshold. The accumulated time of trimmed silence is then returned as the result and then removed using the `sound[trim_ms:]` function, spectifying the start and the end trim duration.

4. The sampling rate of all the audio samples was also made equal to work with the `PsychoPy` backend. The sampling rate was changed to 48Hz.

Chapter 5

Conditions

Chapter 6

Randomization and Quality Control

Chapter 7

Experiment Organization

Chapter 8

Preprocessing and Analysis

Chapter 9

Preprocessing and Analysis

9.1 Organizing the raw data into trial-wise data

The basic unit of analysis in the visual world paradigm experiment is a trial. A trial is a single instance of the experiment. The first task was to organize the raw data into trial-wise data. Here the raw data was present in the .tsv files. Each file contained the data for a single participant. The data was read into a pandas dataframe named `df_interest` and then the columns that were not required were dropped. The columns, `TIME`, `BPOGX`, `BPOGY`, `FPOGD`, `FPOGX`, `FPOGY`, `FPOGV` and `USER` were relevant for our analysis so these were the columns remaining in the dataframe.

In order to organize the entries of the dataframe into trials, the rows corresponding to the start and end of the trials needed to be identified. After acquiring the indices of the corresponding rows, the dataframe was split into multiple dataframes, each corresponding to a single trial.

```
# get the indices of the rows where the user column contains the phrase
 ↪  'START_TRIAL'
start_indices =
 ↪  df_interest[df_interest['USER'].str.contains('START_TRIAL')].index

# get the indices of the rows where the user column contains the phrase
 ↪  'FINAL_FIXATION_END'
end_indices =
 ↪  df_interest[df_interest['USER'].str.contains('FINAL_FIXATION_END')].index

# split the dataframe based on the start and end indices
```

```
df_list = [df_interest.iloc[start_indices[i]:end_indices[i]] for i in
 ↪  range(len(start_indices))]
```

At this point, the dataframes corresponding to the individual trials were ready. Our analysis is mainly concerned with the gaze data from the point of onset of the audio stimilus up till the point at which the participant clicks on a stimulus. So, the dataframes were further sliced to retain only the data from the specified interval. In order to perform this, the `USER` column was used. The rows corresponding to the start of the audio stimulus and the end of the click response were identified by the strings `LOG_AUDIO_TARGET_START` and `CLICK_RESPONSE_END` respectively. The rows between these two rows were sliced and the resulting dataframes were stored in a list named `audio_df_list`.

```
# extract the row index where the user column contains the phrase
 ↪  'LOG_AUDIO_TARGET_START'
audio_start_index =
 ↪  selected_df[selected_df['USER'].str.contains('LOG_AUDIO_TARGET_START')].index[0]
# extract the row index where the user column contains the phrase 'LOG_AUDIO_TARGET_END'
audio_end_index =
 ↪  selected_df[selected_df['USER'].str.contains('CLICK_RESPONSE_END')].index[0]
# split the dataframe based on the audio start and end indices
# store the split dataframe in a list
audio_df_list.append(selected_df.iloc[audio_start_index:audio_end_index + 1])
```

9.2 Extracting trial information

The logs present in the .tsv files are important for our analysis. Apart from containing the data recordings from the experiments, they also contain the information about the individual trials. In the text block below, the logs for a sample trial are shown.

```
START_EXP
START_TRIAL: 0 T: SADDLE.PNG R: PICKLE.PNG B: PADLOCK.PNG L: CANDY.PNG
FIXATE_CENTER_AUDIO_ONSET, COND: 12 TARGET: CANDY
CENTRE_GAZE_START
INSTRUCTION_TO_CLICK_ONSET
LOG_AUDIO_TARGET_START
LOG_AUDIO_TARGET_END
CLICK_RESPONSE_END
FINAL_FIXATION_START, SELECTED: CANDY.PNG
```

```
FINAL_FIXATION_END
….
….
….
….
STOP_EXP
```

The logs are in the form of a sequence of events. Each log event is a line in the log file. Out of these log events, the following ones are relevant for our analysis:

- `START_TRIAL: 0 T: SADDLE.PNG R: PICKLE.PNG B: PADLOCK.PNG L: CANDY.PNG` : This line contains the information about the trial. The trial number is 0. The images on the left, right, top and bottom of the target image are `SADDLE.PNG`, `PICKLE.PNG`, `PADLOCK.PNG` and `CANDY.PNG` respectively.
- `FIXATE_CENTER_AUDIO_ONSET, COND: 12 TARGET: CANDY`: Other than indicating the onset of the audio for fixation, this line also contains the target word and the condition number. In this case, the target word is `CANDY` and the condition number is 12.
- `FINAL_FIXATION_START, SELECTED: CANDY.PNG`: This line indicates the onset of the final fixation on the target image and the stimulus that was selected. The participant selected the image `CANDY.PNG` as the target image.

Following the slicing procedure mentioned the previous section, the indices of the rows corresponding to these three log events are retrieved.

```python
# get all rows whose indices are stored in start_indices
# will be used to extract the position of the stimuli
trial_strings = df_interest.iloc[start_indices]['USER'].reset_index(drop=True)

# get the indices of rows that contain the phrase 'FIXATE_CENTER_AUDIO_ONSET'
target_row_indices =
 ↪  df_interest[df_interest['USER'].str.contains('FIXATE_CENTER_AUDIO_ONSET')].index
target_rows = df_interest.iloc[target_row_indices].reset_index(drop=True)['USER']

# get the indices of rows that contain the phrase 'FINAL_FIXATION_START'
fixation_row_indices =
 ↪  df_interest[df_interest['USER'].str.contains('FINAL_FIXATION_START')].index
fixation_rows = df_interest.iloc[fixation_row_indices].reset_index(drop=True)['USER']
```

The retrieved data were all of the datatype string so regex was used to extract the data points of interest. This consisted of the name of the stimulus at the top, bottom, left and right positions of the grid, the target word, the condition number and the selected stimulus. The extracted data were stored in a python dictionary named `stimuli_loc_dict` with appropriate keys.

```
# use regex to extract the number afer 'COND:'
cond_numbers = [re.findall(r'COND: (\d+)', row)[0] for row in target_rows]
# use regex to extract the word after 'TARGET:'
target_words = [re.findall(r'TARGET: (\w+)', row)[0] for row in target_rows]
# use regex to extract the word after 'SELECTED: '
selected_words = [re.findall(r'SELECTED: (\w+)', row)[0] for row in fixation_rows]

# use regex to extract the image names at the top, bottom, right and left positions
top_stimuli = [re.findall(r'T: (\w+)', trial_string)[0] for trial_string in
 ↪  trial_strings]
bottom_stimuli = [re.findall(r'B: (\w+)', trial_string)[0] for trial_string in
 ↪  trial_strings]
right_stimuli = [re.findall(r'R: (\w+)', trial_string)[0] for trial_string in
 ↪  trial_strings]
left_stimuli = [re.findall(r'\sL: (\w+)', trial_string)[0] for trial_string in
 ↪  trial_strings]
```

The gaze data contained in `audio_df_list` provided the coordinates where the participant was fixating at a given timestamp but for our task we needed to know which stimulus the participant was fixating at. The coordinates of the grid boxes were noted from the OpenSesame experiment UI. But one issues with this data is that these coordinates had the origin at the center of the screen whereas the gaze data had the origin at the top left corner of the screen. So, the coordinates of the grid boxes were converted to the coordinate system of the gaze data and then scaled to the range [0, 1][1]. The functions `shift_coordinate_system` and `shift_coordinate_system_single` were defined for this purpose. The function `shift_coordinate_system` accepted a dictionary of coordinates while the function `shift_coordinate_system_single` accepted a single set of coordinates (tuple). The functions returned the shifted coordinates.

```
# shift the origin from (0, 0) to (-960, 540)
# perform the same on outer_points and inner_points
def shift_coordinate_system(coord_dict):
    for key, value in coord_dict.items():
        coord_dict[key] = (value[0] + 960, -1 * value[1] + 540)

        # scale to [0, 1]
        coord_dict[key] = (coord_dict[key][0] / 1920, coord_dict[key][1] / 1080)
    return coord_dict
```

[1]The scaling is performed with regard to the resolution of a screen resolution of 1920x1080. Hence, a maximum value of 1 along height and width correspond to 1080 and 1920 respectively.

```python
def shift_coordinate_system_single(coord):
    coord = (coord[0] + 960, -1 * coord[1] + 540)
    coord = (coord[0] / 1920, coord[1] / 1080)
    return coord
```

The `shift_coordinate_system` function can convert OpenSesame UI coordinates to the cartesian coordinate system (origin on the bottom left corner) and scale them to the range `[0, 1]`. The gaze data acquired from the GP3 eye tracker follows a different coordinate system. The origin of the gaze data coordinate system is at the top left corner of the screen. Additionally, the y-axis is inverted, meaning that the y-coordinate increases as the participant looks down. In order to convert the gaze data to the cartesian coordinate system in order to enable comparison with the transformed OpenSesame UI coordinates, the function `shift_coordinate_system_bottom_left_to_top_left` was defined. The scaled version of the cartesian coordinates was chosen in order to enable use with plotting libraries such as matplotlib and seaborn.

```python
def shift_coordinate_system_bottom_left_to_top_left(x, y):
    return (x, -1 * y + 1)
```

Nota Bene

The GP3 gaze data coordinates are in the range `[0, 1]` so no scaling is required.

9.3 Fixation plots

The preprocessing steps described in the previous sections are performed on the recorded data, it is possible to plot the fixations of the participants . Such plots allow us visualize the fixations of the participants and identify any outliers. The plot elements can be classified into two groups:

1. Overlay elements: These elements are plotted in order to provide reference for the position of the grid and indicate the stimulus image in each grid box element. The condition number and the target word are also displayed in the plot. The function `draw_grid` is used to draw the grid.

```python
def draw_grid(inn, out, ax):
    # draw line from A to B
    ax.plot([out['A'][0], out['B'][0]], [out['A'][1], out['B'][1]], color='black',
    ↪ alpha=0.3)
    # draw line from B to C
    ax.plot([out['B'][0], out['C'][0]], [out['B'][1], out['C'][1]], color='black',
↪   alpha=0.3)
    # draw line from C to D
    ...
    ...
    # create a tiny circle at the center
    ax.scatter(inn['M'][0], inn['M'][1], color='black', s=5)
```

The text elements are plotted using `matplotlib.pyplot.text()` function. See example:

```
# top stimuli
ax.text(0.5, 0.8685, stimuli_dict[i][0].lower(), transform=ax.transAxes, fontsize=10,
    verticalalignment='top', bbox=props, ha='center')
```

2. Fixation elements: These elements are plotted in order to indicate the fixations of the participants. As indicated in the code block below, the matplotlib scatter function is used.

```
# new_fpog_x and new_fpog_y are the x and y coordinates of the fixations
ax.scatter(new_fpog_x, new_fpog_y, color='red', s=5)
```

Nota Bene

The fixation plots can be generated by running the script `generate_fixation_plots.py` in the `src` directory.

9.4 Deduce location of fixations

Using the coordinates of the edges of the grid boxes, it is possible to deduce the location of the fixations. The gridbox has four boxes that where a stimulus can be placed. The coordinates of the fixations and the coordinates of the stimulus boxes are converted to the scaled cartesian coordinate system. The function `check_if_within_rect` accepts the x and y coordinates of the fixation and the coordinates of the stimulus box and returns a boolean value indicating whether the fixation is within the stimulus box. The function `check_if_within_rect` is called for each stimulus box and the stimulus box for which the function returns `True` is the stimulus box where the participant was fixating.

```
def get_rect(x, y):
    if check_if_within_rect(x, y, top_rect):
        return 'top'
    elif check_if_within_rect(x, y, right_rect):
        return 'right'
    elif check_if_within_rect(x, y, bottom_rect):
        return 'bottom'
    elif check_if_within_rect(x, y, left_rect):
        return 'left'
    elif check_if_within_rect(x, y, centre_rect):
        return 'centre'
    else:
        return 'outside'
```

The function `get_rect` is applied to each row of the dataframe using the `df.apply` function. The resulting column is named `rect`. At the point, for each data point, we know at which stimulus box the participant was fixating.

```
# use df.apply to apply the get_rect function to each row
audio_df_valid_fixation['rect'] = audio_df_valid_fixation.apply(lambda row:
 ↪  get_rect(row['FPOGX'], row['FPOGY']), axis=1)
```

9.5 Mapping stimulus location to stimulus type

The analysis plot is concerned with the stimulus type rather than the stimulus location. The fixations have already been mapped to the stimulus location so by using the data available in the csv log file it was possible to map the stimulus location to the stimulus type for each trial. The csv logfile contains the following columns, referant, cohort, rhyme, distractor, target, trial number and condition number. Each row indicates the names of the stimulus that was assigned the role of referant, cohort, rhyme, etc., for a given trial.

The csv log file was read into a pandas dataframe named `logger_df`. This dataframe has 36 rows, each corresponding to a trial. The information available in this dataframe can be combined with that available in the dictionary `stimuli_loc_dict` to map the stimulus location (top, right, bottom, left) to the stimulus type. The columns `top`, `right`, `bottom` and `left` were added to the dataframe `logger_df` and the values were populated using the dictionary `stimuli_loc_dict`.

```
# add the data from the stimuli_loc_dict to the logger_df
logger_df['top'] = [stimuli_loc_dict[idx][0].lower() for idx in
 ↪  logger_df['count_trial_sequence']]
logger_df['right'] = [stimuli_loc_dict[idx][1].lower() for idx in
 ↪  logger_df['count_trial_sequence']]
logger_df['bottom'] = [stimuli_loc_dict[idx][2].lower() for idx in
 ↪  logger_df['count_trial_sequence']]
logger_df['left'] = [stimuli_loc_dict[idx][3].lower() for idx in
 ↪  logger_df['count_trial_sequence']]
```

These new column were filled by the names of the stimuli but we are interested in the stimulus type. The code block below shows how the contents of the `logger_df` were utilized to fill the columns `top_type`, `right_type`, `bottom_type` and `left_type`.

```
# create columns 'top_type', 'right_type', 'bottom_type', 'left_type' and populate them with
 ↪  the type of stimuli
# by checking if the stimuli is a referant, distractor, rhyme or cohort
logger_df['top_type'] = logger_df.apply(lambda row: 'referant'
        if row['top'] == row['referant'] else 'distractor'
        if row['top'] == row['distractor'] else 'rhyme'
        if row['top'] == row['rhyme'] else 'cohort'
        if row['top'] == row['cohort'] else 'NA', axis=1)
...
...
logger_df['left_type'] = logger_df.apply(lambda row: 'referant'
        if row['left'] == row['referant'] else 'distractor'
        if row['left'] == row['distractor'] else 'rhyme'
        if row['left'] == row['rhyme'] else 'cohort'
        if row['left'] == row['cohort'] else 'NA', axis=1)
```

Now every fixation could be mapped to a stimulus type i.e. whether the participant was fixating on the referant, distractor, rhyme or cohort. Although all required data for the mapping was available, the mapping was actually performed by the function `get_seen_stimuli_type` that loops through the rows of the `logger_df` dataframe and returns the stimulus type that the participant was fixating on. This function modified the `seen` column of the `audio_df_valid_fixation` dataframe so that it now contained the stimulus type that the participant was fixating on.

9.6 Issue of unequal entries per trial

The task of the final analysis plot is to visualize the proportion of fixations on the each stimulus type across trials and all participants. In order to create the plot, each trial must have equal number of data points. But the number of data points per trial is not equal, the number is dependent on the number of valid fixations made by the participant.

This issue is evident from the following plots:

9.6.1 Plotting the number of fixations per trial for one participant

This is done to visualize the number of fixations per trial for one participant. See figure below.

It is evident from the plot that the number of fixations per trial is not equal.

9.6.2 Comparing the trial durations for each condition number

The trials corresponding to each condition number posed a different task to the participant. The duration of the trials for each condition number was compared to see if the condition number had any effect on the number of fixations. See figure below.

It is evident from the plot that the duration of the trials for each condition number does not vary significantly. For the same condition number, the duration of the trials varies slightly. This is due to the fact that the participants were allowed to take their time to respond to the audio stimulus. Overall, there is no trend showing that some condition numbers have longer trials than others.

9.7 Solution: Binning the data

The data points were binned in order to ensure that each trial had equal number of data points. The binning was performed on the basis of time. An overall trial duration `avg_duration` was calculated and it was split into `N` equal parts. `N` is a user-defined parameter, it was chosen as `80` in our analysis. The data points were then binned into these parts.

```python
avg_duration = 1.6
print("Average duration is set to {} s".format(avg_duration))

# divide the avg duration into N equal parts
N = 80
duration_thresholds = np.linspace(0, avg_duration, N, endpoint=True)
```

The `duration_thresholds` array contains the time thresholds for each bin.

> **Nota Bene**
>
> The overall trial duration was calculated by averaging the trial duration across all trials and across all participants.

For determining the duration of each trial, the timestamp of the first fixation and the timestamp of the last fixation were used. The difference between these two timestamps was calculated and this was the duration of the trial. This duration was also used in the previous section to compare the trial durations for each condition number.

```python
first_fixation_time = []
last_fixation_time = []
for idx, row in logger_df.iterrows():
    trial_df = audio_df_valid_fixation[audio_df_valid_fixation['trial_number'] == idx]
    first_fixation_time.append(trial_df['TIME'].min())
    last_fixation_time.append(trial_df['TIME'].max())

logger_df['first_fixation_time'] = first_fixation_time
logger_df['last_fixation_time'] = last_fixation_time

logger_df['duration'] = logger_df['last_fixation_time'] - logger_df['first_fixation_time']
```

After the bin thresholds were determined, the data points were binned into these thresholds. One of the most important column of the `logger_df` dataframe was the `seen` column that contained the name of the type of stimulus that was fixated on by the participant. During binning, the `seen` column values of entries belonging to the same bin are replaced by a single value. The value is determined by the following rules: * First, the values `centre` and `outside` are replaced by `NA` (empty string). * If the bin contains no values, the seen value is set to `NA`. * If the bin contains only one value, the seen value is set to that value. * If the bin contains more than one value, the seen value is set to the value that occurs the most number of times in the bin.

> **ℹ Nota Bene**
>
> See `get_relevant_rect_value()` function in source code for implementation details.

A new dataframe `count_df` was created to store the binned data. The columns of this dataframe were `trial_number`, `condition_number`, `start_time`, `end_time`, `bin_start`, `bin_end`, `real_val_count`, `val_count` and `seen`.

- `trial_number` and `condition_number` were copied from the `logger_df` dataframe.
- `start_time` and `end_time` were the timestamps of the first and last fixation respectively belonging to the bin.
- `bin_start` and `bin_end` were the start and end time of the bin.
- `real_val_count` was the number of data points in the bin.
- `val_count` was the effective number of data points in the bin. This was the number of data points in the bin after calling the `get_relevant_rect_value()` function.
- `seen` was the value of the `seen` column after calling the `get_relevant_rect_value()` function.

9.8 Implementing the conditions of competitor sets

The condition numbers of the four different types of competitor sets are as follows:

```
full_competitor_sets_cond = [1, 2, 3, 4]
cohort_competitor_sets_cond = [5, 6, 7]
rhyme_competitor_sets_cond = [8, 9, 10]
distractor_competitor_sets_cond = [11, 12]
```

As per the conditions, the instances of `rhyme` in the `seen` column were replaced by `distractor` for the trials with condition numbers belonging to the cohort competitor sets.

```
count_df.loc[count_df['condition'].isin(cohort_competitor_sets_cond), 'seen'] =
↪  count_df['seen'].apply(lambda x: 'distractor' if x == 'rhyme' else x)
```

Similarly, the conditions for rhyme competitor sets and distractor competitor sets were implemented.

9.9 Prepare data for plotting

The calculation of fixation probabilities was made simpler by the one hot encoding of the `seen` column. This was done using the `pd.get_dummies()` function. The resulting dataframe was named `one_hot_count_df`. As a result of the one-hot encoding, the `seen` column was replaced by the columns `seen_cohort`, `seen_distractor`, `seen_referant` and `seen_rhyme`. The values of these columns were either 0 or 1. The value 1 indicated that the participant was fixating on the stimulus type indicated by the column name. The value 0 indicated that the participant was not fixating on the stimulus type indicated by the column name.

Nota Bene

`pd.get_dummies()` adds columns of type `boolean`. The columns were converted to type `int` using the `astype()` function.

```
# one hot encode the 'seen' column
one_hot_count_df = pd.get_dummies(count_df, columns=['seen'])
```

Several of the trials acted as filler trials. These trials were not relevant for our analysis so they were removed from the dataframe.

```
# remove the rows that are not relevant for the final analysis
one_hot_count_df = one_hot_count_df[one_hot_count_df['condition'] != 2]
...
one_hot_count_df = one_hot_count_df[one_hot_count_df['condition'] != 12]
```

The groupby function was used to group the rows by the columns `bin_start`, `bin_end`. This was done to calculate the fixation counts for each bin. The `sum()` function was used to calculate the fixation counts. The resulting dataframe was named `grouped_time_bins_df`.

```
# groupby bin_start and bin_end and sum the values in 'seen_referant', 'seen_distractor',
↪ 'seen_rhyme', 'seen_cohort'
groupby_time_bins_df = one_hot_count_df.groupby(['bin_start', 'bin_end']).sum().reset_index()
```

The `groupby_time_bins_df` dataframe was then split into three dataframes, based the competitor set. The dataframes were named `full_competitor_sets_df`, `cohort_competitor_sets_df` and `rhyme_competitor_sets_df`. The `distactor_competitor_sets_df` was not created as it was not required for the analysis.

```
one_hot_count_df_full_competitor_sets =
↪ one_hot_count_df[one_hot_count_df['condition'].isin(full_competitor_sets_cond)]
```

As per the specification of the analysis plot, the plot for the referent stimuli was created using all three competitor sets. The plot for the cohort stimuli was created using the full competitor sets and the cohort competitor sets and the plot for the rhyme stimuli was created using the full competitor sets and the rhyme competitor sets. Therefore, the dataframe `referent_calc_sets` was created by concatenating the dataframes `full_competitor_sets_df`, `cohort_competitor_sets_df` and `rhyme_competitor_sets_df`.

```
referant_calc_sets = pd.concat([one_hot_count_df_full_competitor_sets,
↪ one_hot_count_df_cohort_competitor_sets,
one_hot_count_df_rhyme_competitor_sets], ignore_index=True)
```

Similarly, the dataframes `cohort_calc_sets` and `rhyme_calc_sets` were created.

Chapter 10

Challenges and Limitations

10.1 What we did not replicate from the reference paper?

1. Average duration of auditory stimulus was changed from 375 ms to 750 ms. It was done to as it was more comprehensible for the participants.
   - The average duration was calculated by taking the average of the duration of all the audio stimuli which can be found here.
2. Number of trials for each participant was reduced from 96 to 24. It was done to reduce the time of the experiment and to avoid recalibration of the eye tracker in between the experiment.
3. Audio stimuli were digital (instead of analog). It was done to avoid any noise in the audio stimuli and also to have no influence of the experimenterś accent on the participants.
4. Participants respond with mouse clicks instead of drag-and-drop to the correct box function as does not support drag-and-drop functionality.
5. Use of a 3x3 grid instead of 5x5 as we didin't require the additional boxes for the drag-and-drop functionality as in the original paper.
6. No calibration functionality after each trial was added until the participant was moving their head or eyes too much.

10.2 Challenges

1. Balancing of trials:
   - Challenge :Our stimuli had a total of 23 unique' visual stimuli, 12 different conditions and 4 different types. Adhering to proper randomization and balancing of trials was a challenge as we had to ensure that each participant saw the same number of trials for each condition and each type and such that each object appeared equal number of times.

- Solution : Although we started tith a python script to generate the trials, we had to revert to manual creation of trials as the script was not able to generate balanced trials which was a primary requirement for our experiment.

2. Random 'freezes' during experiment :
   - Challenge : The experiment encountered random freezes during its run. The freezes were random and could not be reproduced. This was a major challenge as we had to restart the experiment from the beginning. The most frequent freezes was found on the gaze contingency check, which allowed the experiment to move forward only when the participant was looking at the center of the screen.
   - Trails :
     1. While True loop
        ```
        while True:
        gazepos = eyetracker.sample()
        ```
     2. Periodic sampling
        ```
        while True:
            if clock.time() - check_timer > diff:
                gazepos = eyetracker.sample()
        ```
     3. Switching backend to PsychoPy Made the sample rates of all the audio samples equal to work with the PsychoPy backend.
     4. Number of trials Reduced the number of trials to 24 from 12 to reduce the time of the experiment but the problem still persisted.
   - Solution : Removal of gaze contingent features and instead introducing a delay to ensure that the participants are looking at the center of the screen before the spoken instruction is played. This was implemented by introducing a delay of 1.1s after the prompt.
3. Eye tracker calibration :
   - Challenge : The eye tracker calibration was a challenge as the participants were not able to calibrate the eye tracker properly, due to many reasons like contact lenses, glasses, height of the participants, body posture during the experiment etc.
   - Solution : THe number of participants were increased to 16 to ensure that we have atleast 12 participants with proper calibration and the timing of one full experiment was reduced to a maximum of 8 minutes to avoid recalibration in between the experiments. The participants were also given a practice trial to ensure that they are comfortable with the experiment and the eye tracker calibration.

10.3 Limitations

1. Since Visual World Paradigm is a well known experimental framework, it is possible that the participants might have been aware of the purpose of the experiment and thus might have been biased in their responses despite the large number of filler trials. Also, we used a relatively small set of pictures, which might have led to a learning effect i.e the participants might have been

able to predict the target word based on the previous trials.

2. Generalizability is affected as our study participants only include university students of a specific age group, which does not fully represent the complexities and variations of real-world spoken word recognition scenarios. The results may not be applicable to other age groups or people with different educational backgrounds who might have more or less exposure to the field of cognitive psychology.

3. The study also may not fully address the universality of the observed effects across different languages as the original study as well as our replication is in English.

Chapter 11

Conclusion

In this project we attempted to investigate the dynamics of predictive language processing through the visual world paradigm. We tried to replicate the experiment conducted by Paul D.Allopenna, James S. Magnuson and Michael K. Tanenhaus in their paper "Tracking the Time Course of Spoken Word Recognition Using Eye Movements: Evidence for Continuous Mapping Models" and validate the hypothesis that the existence of similar sounding words leads to an increased number of fixations on them, reflecting the participants' evolving predictions of the upcoming word. We were able to replicate the results of the original study and thus validate the hypothesis, keeping into account the limitations of our study.

We summarize our conclusions as follows:

1. The inclusion of semantically or phonetically similar words in the spoken instructions results in a higher frequency of fixations on the visual images of the words. This phenomenon illustrates how participants are continuously adjusting their predictions for the upcoming word as they engage with the content.

2. Eye movement tracking is a reliable tool for investigating the time course of spoken word recognition and capturing the mapping process while the spoken word unfolds.

3. The results and plots obtained by us provide an empirical support for continuous and incremental mapping models of word recognition and proves that word processing is not a discrete and all-or-nothing process.

4. Our results suggest that as the spoken word unfolds over time, the listener gradually narrows down the set of candidate words based on the contextual information. This competition among candidate words occurs until a single word is identified or a clear winner emerges.