File I/O, Data Structures & Processing

Week 6

Coding for Creative Robotics

Sonia Litwin
08/11/2024

Agenda

Lecture (45 min):

- Data Types and Sensor Data Review
- Understanding Signals and Processing
- Working with Different File Types
- Data Processing in Robotics

Tutorial (3h):

- File Operations
- Data Structure Practice
- Processing Exercises

Podcast Time!

Data Types and Sensor Data

Recap of Previous Concepts

Data Types:

- Basic Types
  - Integers
  - Floats
  - Strings
  - Booleans
- Complex Structures
  - Lists
  - Dictionaries
  - Arrays

Robotic Sensors:

- Types of Sensors
  - Cameras
  - LIDAR
  - EEG
- Signal Collection
  - Raw data capture
  - Signal processing
  - Data interpretation

The Language of Signals

Understanding Robot-Environment Communication

Understanding Signals:

- Fundamental robot-environment communication
- Multiple sensor modalities
- Signal-to-data transformation

By processing the data we can bridge the gap between sensing and meaningful perception

Modality Categories:

Audio Signals:
- Speech and sound recognition
- Environmental audio

Visual Signals:
- Object recognition
- Environment mapping

Physiological Signals:
- EEG data
- Biosensor readings

From Sound Waves to Digital Data

Understanding Audio Signal Processing

Signal Processing Chain:

1. Microphones capture sound waves
2. Analog signals are digitized into numerical data
3. Process data to understand speech or environmental sounds

Example: Voice Command Recognition

WAV files contain header (metadata) and raw binary data

Sample Audio Data:

```
# Sample audio signal data
audio_data = [
    0,    1024, 2048,
    1024, 0,    -1024,
    -2048, -1024, 0
]

# Each value represents amplitude
# at a specific time point
```

Vision

From Light to Images: Capturing Visual Data

Object Recognition Example:

1. Cameras capture light
2. Light converted to digital signals
3. Images represented as pixel arrays

Image Structure:

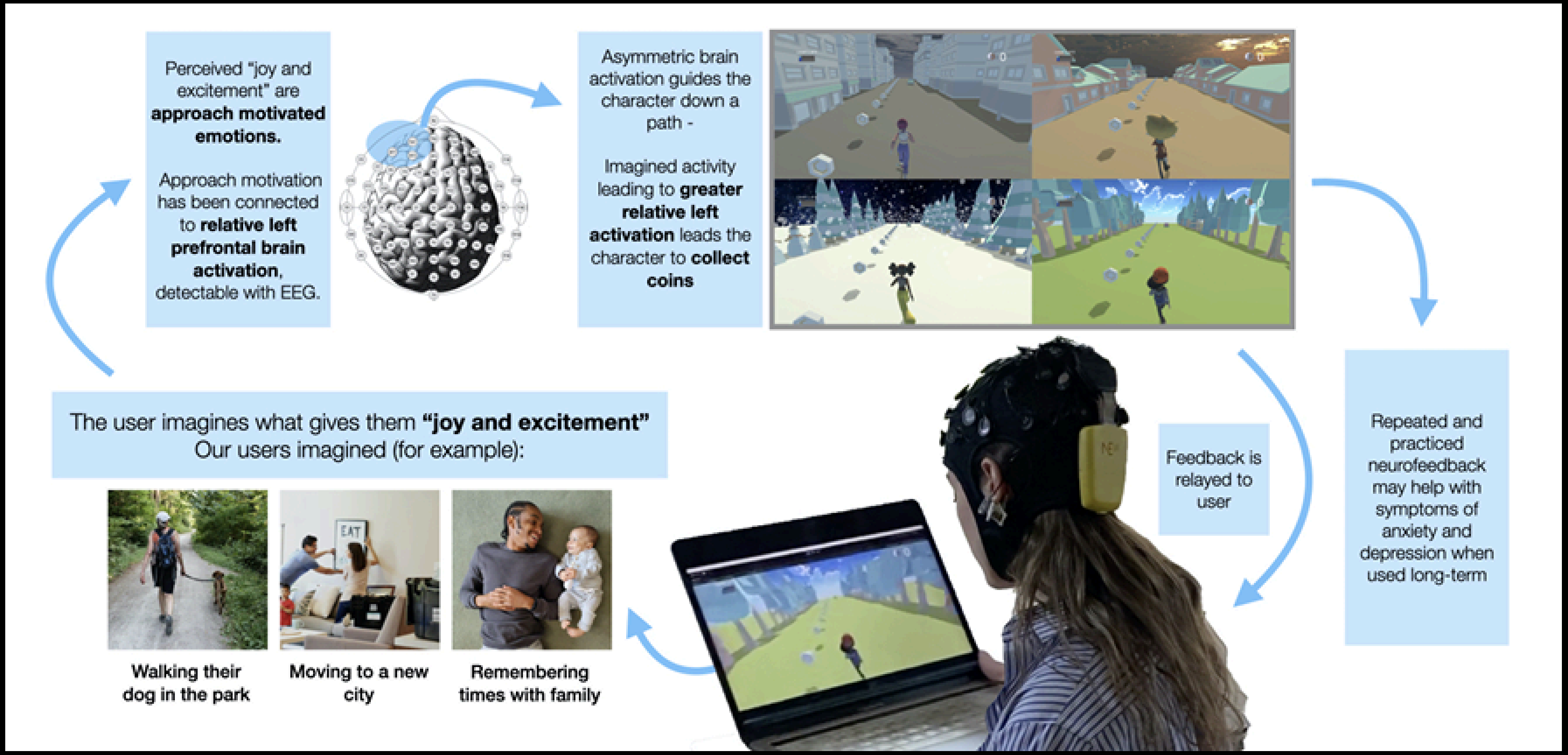
- Pixels store color information
- RGB values (0-255 range)
- Organized in grid format

Sample 2×2 Image Data:

```
# RGB color values per pixel
image_data = [
    [(255,0,0),   # Red
     (0,255,0)],  # Green
    [(0,0,255),   # Blue
     (255,255,0)] # Yellow
]
```

Physiology

Interpreting Biomedical Data



Perceived "joy and excitement" are **approach motivated emotions.**
Approach motivation has been connected to **relative left prefrontal brain activation,** detectable with EEG.
Asymmetric brain activation guides the character down a path -
Imagined activity leading to **greater relative left activation** leads the character to **collect coins**
Repeated and practiced neurofeedback may help with symptoms of anxiety and depression when used long-term
Feedback is relayed to user
The user imagines what gives them **"joy and excitement"**
Our users imagined (for example):
Walking their dog in the park
Moving to a new city
Remembering times with family

Physiology

Interpreting Biomedical Data

Brain-Computer Interfaces:

1. EEG sensors capture brain activity
2. Data collected across multiple channels
3. Time-series readings gathered

Applications:

- Neural control of robots
- Adaptive interfaces
- Assistive technologies

Sample EEG Data:

```
# Multi-channel EEG readings
eeg_data = {
    'time': [0.0, 0.01, 0.02],
    'channel_1': [5.1, 5.3, 5.2],
    'channel_2': [3.2, 3.3, 3.1]
}
```

Transition

From Data Collection to Data Storage

Data Storage

Memory vs. Files in Data Storage

Memory (RAM):

- Temporary storage
- Data lost when program ends
- Fast access speed

Variables Example:

```
name = "Robot"
data = [1, 2, 3]  # Lost after
                 # program ends
```

Files (Disk):

- Permanent storage
- Data persists after program
- Slower access speed

Files Example:

```
sensor_log.txt
config.json     # Data remains
data.csv        # after shutdown
```

Data Reminder

Data Structures in Python

Data Structures

Python's Built-in Collections

Lists and Tuples:

- Lists
  - Ordered, mutable collections
  - Element access and slicing
  - Methods: append, remove, etc.
- Tuples
  - Ordered, immutable collections
  - Faster than lists
  - Perfect for fixed data

Examples:

```
# List example
sensors = ['camera', 'lidar', 'eeg']
sensors.append('gps')

# Tuple example
coordinates = (51.5074, -0.1278)
# coordinates[0] = 50  # Error!
```

Data Structures

Advanced Collections

Sets and Dictionaries:

- Sets
  - Unordered unique elements
  - Fast membership testing
  - Set operations: union, intersection
- Dictionaries
  - Key-value pairs
  - Fast lookups
  - Flexible key types

```
# Set operations
detected = {'car', 'tree', 'person'}
known = {'car', 'sign', 'tree'}
new = detected - known  # {'person'}

# Dictionary example
sensor_data = {
    'temp': 25.5,
    'humidity': 60,
    'pressure': 1013
}
```

File I/O

File Input/Output in Python

File I/O

What is File I/O?

Key Concepts:

- Reading from files
- Writing to files
- File operations workflow
- Data persistence

Basic File Operations:

- Opening files
- Reading data
- Writing data
- Closing files

Example Workflow:

```
# Opening and reading a file
file = open('data.txt', 'r')
content = file.read()
file.close()

# Writing to a file
file = open('log.txt', 'w')
file.write('Hello, World!')
file.close()
```

File I/O

Basic File Operations

File Modes:

- Read Mode ('r')
  - Read existing files
  - Error if file doesn't exist
- Write Mode ('w')
  - Create new file
  - Overwrite existing files
- Append Mode ('a')
  - Add to existing file
  - Create if doesn't exist

Mode Examples:

```
# Reading a file
with open('data.txt', 'r') as file:
    content = file.read()

# Writing to a file
with open('new.txt', 'w') as file:
    file.write('New data')

# Appending to a file
with open('log.txt', 'a') as file:
    file.write('New entry\n')
```

File I/O

Reading from Files

Reading Methods:

- file.read()
  - Reads entire file
  - Returns single string
- file.readline()
  - Reads one line
  - Includes newline character
- file.readlines()
  - Returns list of lines
  - Good for line-by-line processing

```
# Reading entire file
with open('data.txt', 'r') as file:
    content = file.read()

# Reading line by line
with open('data.txt', 'r') as file:
    for line in file:
        print(line.strip())
```

File I/O

Writing to Files

Writing Methods:

- file.write()
  - Writes string to file
  - No automatic newline
- file.writelines()
  - Writes list of strings
  - No separators added

Remember:

- Add \n for new lines
- Close files when done
- Use with statement

```python
# Writing single string
with open('output.txt', 'w') as file:
    file.write('Hello\n')
    file.write('World\n')

# Writing multiple lines
lines = ['Line 1\n', 'Line 2\n']
with open('output.txt', 'w') as file:
    file.writelines(lines)
```

File I/O

Write Modes: Overwrite vs. Append

Mode Comparison:

• Overwrite Mode ('w')
  - Clears existing content
  - Creates new file if needed
  - Starts writing at beginning

• Append Mode ('a')
  - Preserves existing content
  - Creates new file if needed
  - Adds to end of file

```
# Overwrite mode
with open('log.txt', 'w') as file:
    file.write('New log starts here\n')

# Append mode
with open('log.txt', 'a') as file:
    file.write('Adding new entry\n')
    file.write('Another entry\n')
```

File I/O

Ensuring Data is Properly Formatted

Common Issues:

- Line Endings
  - Add \n for new lines
  - Platform differences
- Data Separation
  - Use consistent delimiters
  - Format data clearly

```
# Good formatting
with open('data.txt', 'w') as file:
    file.write('Name: Robot 1\n')
    file.write('Sensor: Camera\n')
    file.write('Status: Active\n')

# Poor formatting (no separators)
with open('data.txt', 'w') as file:
    file.write('Robot 1')
    file.write('Camera')
    file.write('Active')
```

File I/O

Using File Context Managers

with Statement:

- Advantages
  - Automatic file closing
  - Better error handling
  - Cleaner code
- Best Practices
  - Always use with
  - Properly indent code
  - Handle exceptions

```
# Using with statement
with open('data.txt', 'w') as file:
    file.write('Safely written\n')
# File automatically closes

# Without with (not recommended)
file = open('data.txt', 'w')
file.write('Unsafe writing')
file.close()  # Might be forgotten!
```

File I/O

Handling Exceptions During File I/O

Common Exceptions:

- FileNotFoundError
  - File doesn't exist
  - Wrong file path
- PermissionError
  - No access rights
  - File locked
- IOError
  - General I/O problems
  - Disk errors

```python
try:
    with open('data.txt', 'r') as file:
        content = file.read()
except FileNotFoundError:
    print("File not found!")
except PermissionError:
    print("No permission!")
except IOError as e:
    print(f"Error: {e}")
```

Data Abundance

Working with Various File Types

Working with CSV Files

CSV Files in Robotics

CSV Structure:

- Plain text format
- Comma-separated values
- Rows as records
- Common Use Cases:
  - Sensor data logs
  - Experimental results
  - Configuration data

Additional Resources:
Python CSV Documentation

```
# Example CSV content
timestamp,sensor,value
2024-02-09,temp,25.5
2024-02-09,humidity,60
2024-02-09,pressure,1013

# Reading with csv module
import csv
with open('data.csv', 'r') as file:
    reader = csv.reader(file)
    for row in reader:
        print(row)
```

Working with CSV Files

Built-in Functions for Data Access

Common Functions:

- csv.reader()
  - Basic row iteration
  - Returns lists
- csv.DictReader()
  - Column name access
  - Returns dictionaries
- pandas.read_csv()
  - Advanced data handling
  - DataFrame operations

```
# Using csv.reader
import csv
with open('data.csv', 'r') as file:
    reader = csv.reader(file)
    headers = next(reader)  # Skip header
    for row in reader:
        print(row[0])  # First column

# Using pandas
import pandas as pd
df = pd.read_csv('data.csv')
print(df['column_name'])
```

Working with CSV Files

Understanding and Handling NaN Values

Common Causes of NaNs:

1. Missing Source Data
   - Empty cells in CSV
   - Incomplete records
2. Data Type Issues
   - Type mismatches
   - Incorrect parsing
3. Structure Problems
   - Inconsistent columns
   - Wrong delimiters

```
# Handling NaN values with pandas
import pandas as pd

df = pd.read_csv('data.csv',
    na_values=['', 'NA', 'null'],
    skip_blank_lines=True)

# Check for NaNs
print(df.isna().sum())

# Fill NaN values
df.fillna(0, inplace=True)
```

Working with JSON Files

JSON in AI and Robotics

JSON Features:

- Human-readable format
- Structured data
- Common Uses:
  - Configuration files
  - Data exchange
  - API responses

```
import json

# Reading JSON
with open('config.json', 'r') as file:
    config = json.load(file)

# Writing JSON
data = {
    'name': 'Robot1',
    'sensors': ['camera', 'lidar'],
    'active': True
}
with open('data.json', 'w') as file:
    json.dump(data, file, indent=2)
```

Binary Files

Handling Binary Files and Images with PIL

Binary Files:

- What are Binary Files?
  - Non-text data format
  - Images, audio, executables
- Using PIL
  - Image processing
  - Format conversion
  - Image manipulation

```python
from PIL import Image

# Opening an image
image = Image.open('photo.jpg')

# Basic operations
image = image.resize((800, 600))
image = image.rotate(90)

# Save in different format
image.save('photo.png')
```

Binary Files

Generating Animated GIFs with PIL

GIF Creation:

- Animation Components
  - Frame sequence
  - Duration settings
  - Loop control
- Applications
  - Visualize robot movements
  - Show state changes
  - Demonstrate behaviors

```python
from PIL import Image

# Load frame images
frames = [
    Image.open(f'frame_{i}.png')
    for i in range(5)
]

# Create animated GIF
frames[0].save(
    'animation.gif',
    save_all=True,
    append_images=frames[1:],
    duration=200,
    loop=0
)
```

File Formats

Exploring Additional File Types

Common Formats:

- Audio Files
  - WAV: Uncompressed audio
  - MP3: Compressed audio
- Video Files
  - MP4: Standard video
  - AVI: Raw video data
- Special Formats
  - ROS bag files
  - Custom binary formats

Key Takeaway:

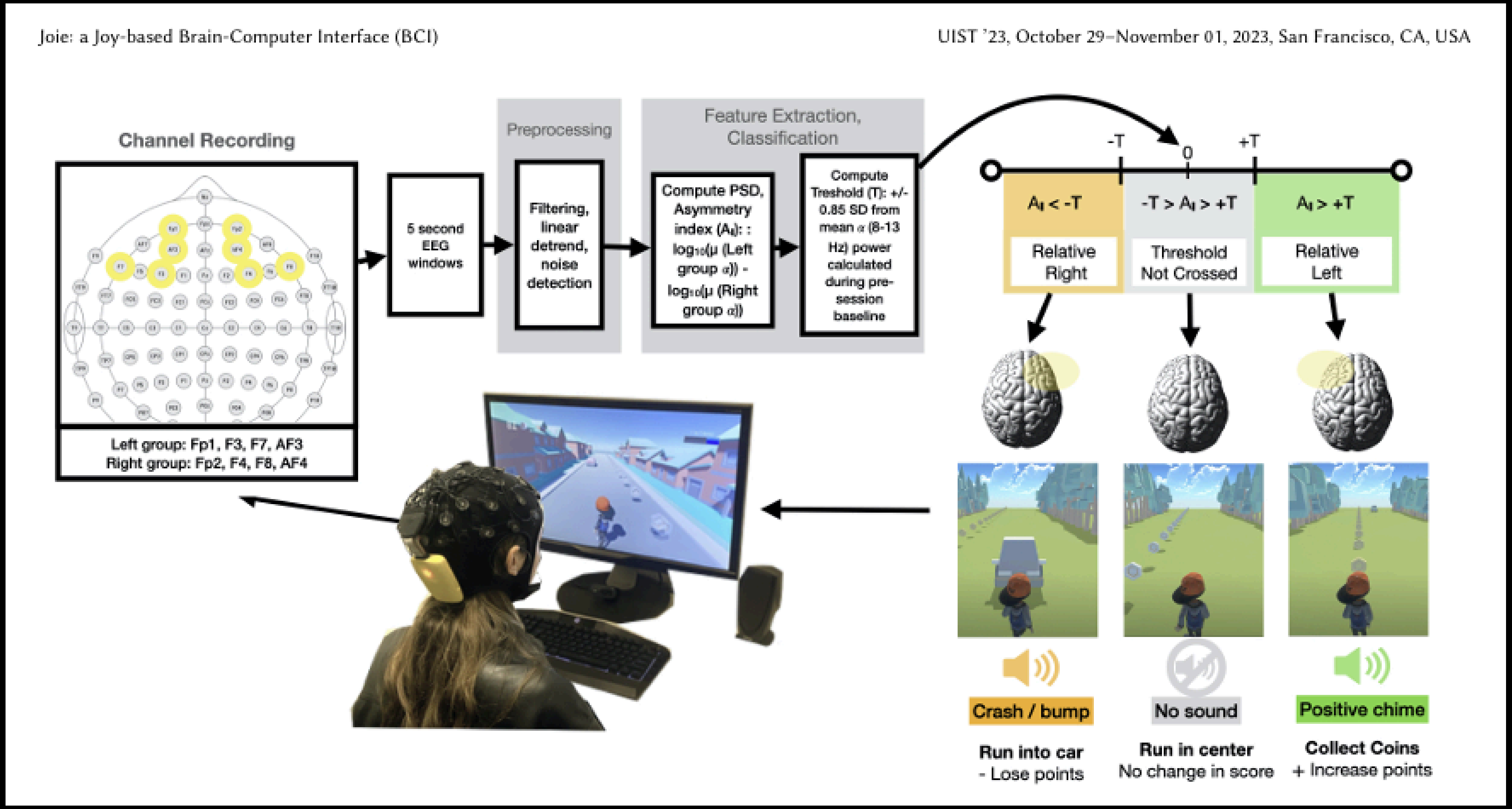
- Understand format purposes
- Choose appropriate libraries
- Consider data requirements
- Handle format-specific issues

Data Processing

Signals == Understanding?

Data Processing

Transforming Raw Data into Actionable Insights



Joie: a Joy-based Brain-Computer Interface (BCI)
UIST '23, October 29–November 01, 2023, San Francisco, CA, USA
Channel Recording
Left group: Fp1, F3, F7, AF3
Right group: Fp2, F4, F8, AF4
5 second EEG windows
Preprocessing
Filtering, linear detrend, noise detection
Feature Extraction, Classification
Compute PSD, Asymmetry index (A_I): : log10(μ (Left group α)) - log10(μ (Right group α))
Compute Treshold (T): +/- 0.85 SD from mean α (8-13 Hz) power calculated during pre-session baseline
-T
0
+T
A_I < -T
-T > A_I > +T
A_I > +T
Relative Right
Threshold Not Crossed
Relative Left
Crash / bump
No sound
Positive chime
Run into car
- Lose points
Run in center
No change in score
Collect Coins
+ Increase points

Source: Angela Vujic, Shreyas Nisal, and Pattie Maes. 2023. Joie: a Joy-based Brain-Computer Interface (BCI). In Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST '23). Association for Computing Machinery, New York, NY, USA, Article 82, 1–14. https://doi.org/10.1145/3586183.3606761

Data Processing

Transforming Raw Data into Actionable Insights

Processing Pipeline:

1. Acquisition
   - Raw signal capture
   - Sensor readings
2. Digitization
   - Analog to digital
   - Sampling process
3. Processing
   - Feature extraction
   - Signal transformation
4. Interpretation
   - Pattern recognition
   - Decision making

```python
# Example processing pipeline
def process_signal(raw_data):
    # 1. Acquisition
    signal = acquire_data(raw_data)

    # 2. Digitization
    digital = convert_to_digital(signal)

    # 3. Processing
    features = extract_features(digital)

    # 4. Interpretation
    result = analyze_features(features)
    return result
```

Data Processing

Essential Libraries for Data Processing

Processing Libraries:

- pandas
  - Data manipulation
  - CSV processing
- OpenCV
  - Computer vision
  - Image processing
- librosa
  - Audio analysis
  - Music processing

```python
# Using pandas for data analysis
import pandas as pd
df = pd.read_csv('data.csv')
result = df.groupby('category').mean()

# Using OpenCV for images
import cv2
img = cv2.imread('image.jpg')
processed = cv2.GaussianBlur(img, (5,5), 0)

# Using librosa for audio
import librosa
y, sr = librosa.load('audio.wav')
```

Sample Application

Processing EEG Data for BCI-Controlled Games

t	F3	F4	Fp1	Fp2	F7	F8	AF3	AF4
1.816789e+06	-24110.509766	-24186.947266	-24320.699219	-24311.974609	-24288.417969	-24319.841797	-24239.113281	-24229.910156
1.816789e+06	-24108.365234	-24185.611328	-24319.412109	-24310.447266	-24287.273438	-24318.125000	-24237.349609	-24228.146484

Example of EEG data stored in CSV format. The columns represent different electrode channels (F3, F4, Fp1, etc.) capturing electrical activity across different regions of the brain. Time stamps are in the leftmost column.

Sample Application

Processing EEG Data for BCI-Controlled Games

Processing Pipeline:

1. Data Collection
   - Raw EEG signals
   - Multiple channels
2. Pre-processing
   - Noise filtering
   - Artifact removal
3. Feature Extraction
   - Frequency bands
   - Signal patterns
4. Classification
   - Machine learning
   - Real-time processing

```
import os
import logging
import pyxdf
import sys
from math import log
import matplotlib.pyplot as plt
import mne
import numpy as np
import copy
import pandas as pd
import warnings
from scipy.signal import butter, filtfilt
import re

# Turn off verbose output for optimization
mne.set_log_level('ERROR')
warnings.simplefilter(action='ignore',
                      category=FutureWarning)

# matplotlib default plot size
plt.rcParams['figure.figsize'] = (15, 6)
```

Data Visualization

Enhancing Data Usability Through Visualization

Key Aspects:

- Explainability
  - Clear data representation
  - Intuitive interfaces
- User Experience
  - Interactive dashboards
  - Real-time updates
- Trust Building
  - Transparent processes
  - Accessible insights

Visualization Best Practices:

1. Choose appropriate chart types
2. Maintain consistent styling
3. Use clear labels and legends
4. Consider color accessibility
5. Include interactive elements
6. Provide context and explanations

Ethics in Data Processing

Responsible Data Handling and Storage

Key Considerations:

- Privacy
  - Data protection
  - User consent
- Security
  - Access control
  - Encryption
- Compliance
  - GDPR requirements
  - Industry standards
- Transparency
  - Clear documentation
  - User awareness

Best Practices:

1. Implement robust data protection measures
2. Obtain explicit user consent
3. Regular security audits
4. Clear data usage policies
5. Proper documentation
6. Regular updates and maintenance

Tutorial Time!

Sonia Litwin & Rohit Ramesh Thampy

08/11/2024