DeepLabCut-Analysis-Jupyter-Scripts

Augustine, Farhan [1]

1. **University of Maryland, Baltimore County**

Abstract

This white paper describes the Motion Analysis Toolkit for DeepLabCut, DeepLabCut-Analysis-Jupyter-Scripts, a Python-based analytical tool designed to process and visualize motion data from behavioral experiments. The toolkit offers a comprehensive approach to calculating distances and velocities and detecting significant movements or "jumps" in the tracked data of animal subjects. This Python-based analytical tool leverages CSV output files from the DeepLabCut framework to provide researchers with advanced capabilities for motion analysis, including filtered distance calculations, jump detection, and velocity analysis with Savitzky-Golay smoothing.

Introduction

The DeepLabCut-Analysis-Jupyter-Scripts for DeepLabCut represent a significant advancement in behavioral neuroscience, particularly in motion analysis. Traditional methods of motion analysis often require manual tracking and subjective interpretation, which can be time-consuming and prone to human error. With the advent of machine learning tools like DeepLabCut and SLEAP, researchers have been able to automate the tracking process, yet the analysis and visualization of this data remain a challenge.

The toolkit and workflow introduced here bridge this gap by providing an automated, robust, and user-friendly solution for the post-processing and visualization of motion data. Unlike other tools, which may offer limited functionality or require extensive programming knowledge, this toolkit is designed with the end-user in mind, offering a suite of functions that can be easily customized and integrated into existing workflows.

Advantages Over Existing Tools

- Ease of Use: The toolkit is built on a Python-based platform, making it accessible to researchers with varying levels of programming expertise. It simplifies complex analytical tasks into manageable functions that can be executed with minimal setup.
- Customization: While other tools might offer a complex set of tools, this toolkit recognizes the need for simplicity in face of unique behavioral experiments. It allows users to define specific parameters, such as body parts and regions of interest, tailored to their individual research needs.
- Comprehensive Analysis: The toolkit goes beyond simple tracking, offering features for detecting jumps, calculating filtered distances, and analyzing velocities with advanced smoothing techniques. This level of detail is often complicated and hard to understand with other tools.
- Integration: Designed to complement DeepLabCut's output, the Jupyter Notebook seamlessly integrates into the existing ecosystem of motion analysis tools.

Multi-Body Part Tracking Analysis

A distinctive feature of the Motion Analysis Toolkit for DeepLabCut is its ability to analyze tracks using more than two body parts. This multi-point tracking capability is crucial for comprehensively understanding complex behaviors and movement patterns. Researchers can gain insights into the coordination and relative motion between different points of interest on an animal subject by analyzing multiple body parts simultaneously.

Advantages of Multi-Body Part Tracking:

- Enhanced Accuracy: Tracking multiple body parts reduces reliance on a single point of data, thereby minimizing errors and improving the overall accuracy of the motion analysis.
- Complex Behavior Analysis: Certain behaviors can only be characterized by the movement of several body parts in relation to each other. Multi-body part analysis allows for the analysis of such complex behaviors.
- Dynamic Interaction: Understanding how different body parts move in relation to one another provides a dynamic view of the subject's behavior, essential for studies involving interaction with objects or other subjects.
- Robust Data Set: Collecting data from multiple points provides a more robust data set, leading to more reliable conclusions and supporting a wider range of research questions.

The toolkit's flexibility in handling data from multiple body parts makes it an invaluable resource for researchers who require detailed and dynamic motion analysis. It stands out from other tools that may be limited to single-point tracking or require complex setups for multi-point analysis. With this toolkit, researchers can easily configure their analysis to include any body parts, making it adaptable to various experimental designs and research objectives.

- Open Source: In the spirit of scientific collaboration, the toolkit is open-source, encouraging contributions and improvements from the community. This contrasts proprietary tools, which are often closed to external development.

The analysis script(s) provided here for DeepLabCut CSV analysis address the limitations of current tools and empower researchers to conduct more efficient, accurate, and visually compelling motion analysis, ultimately advancing our understanding of animal behavior.

Methodology

This analysis toolkit for DeepLabCut is engineered to process CSV files containing x, y coordinates and likelihood values of tracked body parts. It has a suite of functions that users can customize to fit their research needs. The toolkit's capabilities include:

- ***Jump Detection***: Utilizes a user-defined maximum distance threshold to identify significant movements, or "jumps," between frames. This threshold can be adjusted based on the expected movement range of the animals being studied.

Define a Threshold for Max Distance a Body Part Can move prior to calculating Distance Moved by body parts

In [ ]:
```python
# Define the maximum distance a body part can move per frame (this should be based on your specific data)
max_distance_per_frame = 100  # Example threshold value for the distance the mouse's bodyparts can move, adjust based on your data. Units are pixels.
frame_rate = 30  # Frames per second, adjust based on your data

def detect_jumps(x_coords, y_coords, max_distance):
    jumps = []
    for i in range(1, len(x_coords)):
        distance = np.sqrt((x_coords[i] - x_coords[i-1])**2 + (y_coords[i] - y_coords[i-1])**2)
        if distance > max_distance:
            jumps.append(i)
    return jumps

def calculate_filtered_distance(body_part_data, part_name, max_distance_per_frame):
    total_distance = 0
    x_coords = body_part_data[part_name]['x']
    y_coords = body_part_data[part_name]['y']
    jumps = detect_jumps(x_coords, y_coords, max_distance_per_frame)

    for i in range(1, len(x_coords)):
        if i not in jumps:
            distance = np.sqrt((x_coords[i] - x_coords[i-1])**2 + (y_coords[i] - y_coords[i-1])**2)
            total_distance += distance

    return total_distance, jumps

# Calculate the filtered total distance moved for each body part
body_parts = ['Head', 'Nose']  # Add 'TailBase' or other parts as needed
distances = {}
jumps_detected = {}

for part in body_parts:
    distances[part], jumps_detected[part] = calculate_filtered_distance(body_part_data, part, max_distance_per_frame)

# Sum the distances moved by all tracked body parts in units
total_filtered_distance_moved_in_units = sum(distances.values())

# Print the filtered total distance moved for each body part and the total in units
for part, distance in distances.items():
    print(f"Filtered total distance moved by the mouse's {part}: {distance:.0f} units")
    if jumps_detected[part]:
        print(f"Jumps detected for {part} at frames: {jumps_detected[part]}")

print(f"Filtered total distance moved together by the mouse's {' and '.join(body_parts)}: {total_filtered_distance_moved_in_units:.0f} units")
```

- ***Filtered Distance Calculation***: Computes the total distance moved by each body part, excluding distances flagged as jumps. This provides a more accurate representation of the subject's movement.

Filter the Data with Savitzky-Golay Filter

In [ ]:
```python
# Assuming body_part_data is a dictionary with keys as body parts
body_parts = list(body_part_data.keys())

# Parameter for frame rate
frame_rate = 30  # Replace with your actual frame rate

# Initialize the dictionary for velocities
velocities = {}

# Calculate velocities and apply Savitzky-Golay filter for smoothing
for part in body_parts:
    # Ensure there are enough data points for velocity calculation
    if len(body_part_data[part]["x"]) > 1:
        # Calculate velocity as the difference in position over time
        velocities[part] = np.diff(body_part_data[part]["x"]) / frame_rate

        # Determine the window length for the Savitzky-Golay filter
        # It must be odd and less than the size of the data
        window_length = min(15, len(velocities[part]) // 2 * 2 + 1)

        # Apply the Savitzky-Golay filter if the window length is valid
        if window_length > 2:  # At least 3 points are needed to apply the filter
            velocities[part] = savgol_filter(velocities[part], window_length=window_length, polyorder=3)
        else:
            print(f"Not enough data points to apply Savitzky-Golay filter for {part}.")
    else:
        print(f"Not enough data points to calculate velocity for {part}.")
```

- ***Velocity Computation:*** Calculates the velocity of each body part between frames and applies a Savitzky-Golay filter for data smoothing. The filter parameters, such as window length and polynomial order, are customizable.

Calculate Average Velocity of Each Body Parts

```
In [ ]: # Assuming velocities is a dictionary with keys as body parts and values as velocity data
        # Initialize the dictionary for average velocities
        average_velocities = {}

        # Iterate over each body part to calculate and print the average velocity
        for part in body_parts:
            # Calculate the average velocity for each body part
            average_velocity = np.mean(np.abs(velocities[part]))
            # Store the average velocity in the dictionary
            average_velocities[part] = average_velocity
            # Print the average velocity for each body part
            print(f"Average velocity for {part}: {average_velocity:.2f} units/frame")
```

- ***ROI Analysis***: This toolkit allows users to define Regions of Interest (ROIs) and analyze the time subjects spend within these areas. The toolkit can track entries and exits, providing insights into subject behavior within specified zones.

User-Configurable Parameters:

- max_distance_per_frame: Sets the threshold for what constitutes a jump, allowing researchers to define the sensitivity of movement detection.
- frame_rate: Adjusts the temporal resolution of the analysis, which is essential for accurate velocity calculations.
- debounce_frames: Determines the minimum number of consecutive frames required for a subject to be considered as having entered or exited an ROI, reducing noise from momentary fluctuations.
- body_part_pairs: Users can specify which body parts to analyze in pairs, enabling multi-body part tracking for complex behavior analysis.

By offering these adjustable parameters, the toolkit empowers researchers to tailor the analysis to their experimental design, ensuring that the results are both relevant and reliable. The methodology is transparent and adaptable, making it suitable for various motion analysis applications in behavioral neuroscience following pose-estimation with DeepLabCut.

Usage

The Motion Analysis Toolkit for DeepLabCut is designed for user-friendly interaction, ensuring that researchers can focus on their experiments without the need for extensive programming knowledge. The toolkit is structured to guide users through a logical sequence of steps, from initial data import to the final analysis and visualization of results.

2. Step-by-Step Guide:

a. Data Import: Begin by loading your CSV data into the toolkit. The data should include x, y coordinates and likelihood values for each tracked body part.

Load Your "CSV" File

```
In [ ]: # Load CSV data, using the second and third rows as the header
        df = pd.read_csv(r"C:\Users\Farha\Downloads\Mouse9_MetalExposed_T-maze_12-14-23DLC_resnet101_TMZ_Model_V2Dec16shuffle1_157000.csv", header=[1, 2])
        print(df.columns)
```

b. Parameter Configuration: Customize the analysis by setting parameters such as max_distance_per_frame and frame_rate to match the specifics of your experimental data. For example, max_distance_per_frame = 100 (adjust based on

your data), and frame_rate = 30 (adjust frames per second based on your own video recording settings).

c. Body Part Selection: Define the body parts you wish to analyze. The toolkit can handle multiple body parts, providing a comprehensive view of the subject's movement. For example, body_parts = ['Head', 'Nose', 'TailBase'] (Add or remove parts as needed).

Entries and Exits when Two Body Parts Simultaniously are present in the Regions of Interest for a specified number of frames.

In [ ]:
```
# Define the body parts you want to track
parts_to_track = ['Nose', 'Head']  # User can replace with the body parts they're interested in

# debound_frame assings the limit for the minimum consecutive frames that the mouse must have entered the ROI before it is counted as entry.
def track_multi_part_entrances_exits(body_part_data, parts, roi, debounce_frames=30):
    inside_roi = False
    entrances = 0
    exits = 0
    inside_counter = 0  # Counter for consecutive frames inside the ROI
    outside_counter = 0  # Counter for consecutive frames outside the ROI

    # Get the coordinates for the body parts
    coords = {part: body_part_data[part] for part in parts}

    for i in range(len(body_part_data[parts[0]]['x'])):
        # Check if all specified body parts are in the ROI
        currently_inside = all(is_in_roi(coords[part]['x'][i], coords[part]['y'][i], roi) for part in parts)

        if currently_inside:
            inside_counter += 1
            outside_counter = 0  # Reset outside counter
        else:
            outside_counter += 1
            inside_counter = 0  # Reset inside counter

        # Count entrance only if body parts have been inside for enough consecutive frames
        if inside_counter >= debounce_frames and not inside_roi:
            entrances += 1
            inside_roi = True
            inside_counter = 0  # Reset inside counter after counting entrance

        # Count exit only if body parts have been outside for enough consecutive frames
        if outside_counter >= debounce_frames and inside_roi:
            exits += 1
            inside_roi = False
            outside_counter = 0  # Reset outside counter after counting exit

    # Print the number of entrances and exits
    print(f"Entrances: {entrances}, Exits: {exits}")
    return entrances, exits

# Example usage
if roi_coords:  # Ensure that an ROI has been selected
    last_roi = roi_coords[-1]  # Use the last ROI drawn
    entrances, exits = track_multi_part_entrances_exits(body_part_data, parts_to_track, last_roi)
    print(f"The mouse entered the ROI {entrances} times and exited {exits} times with both 'Nose' and 'Head' inside.")
```

d. ROI Definition: Draw Regions of Interest (ROIs) within your data plot to focus the analysis on specific areas of interest.

e. Analysis Execution: Run the provided functions to detect jumps, calculate distances, analyze velocities, and assess time spent in ROIs.

f. Results Interpretations: Review the output, including total distances moved, velocities, and ROI occupancy, to conclude the animal's behavior.

Customization and Flexibility:

The toolkit is highly customizable, allowing users to adjust thresholds for jump detection, define debounce frames for ROI entry/exit analysis, and select body part pairs for multi-point tracking analysis. This level of customization ensures that the toolkit can be adapted to a wide range of experimental setups and research questions.

By following these steps, users can harness the full potential of the toolkit for DeepLabCut CSV analysis to conduct detailed and accurate motion analysis. The toolkit's intuitive design and comprehensive documentation enable researchers to quickly become proficient in its use, making it an indispensable asset in the field of behavioral neuroscience.

Discussion and Conclusion

The provided toolkit for DeepLabCut CSV analysis stands to advance the field of behavioral neuroscience. By providing a comprehensive suite of tools for motion analysis, this toolkit addresses a critical need for researchers to analyze complex behaviors through multi-body part tracking. The ability to customize parameters such as jump thresholds, debounce frames, and ROI definitions allows for a tailored approach to various experimental designs, ensuring that the toolkit remains versatile across different research contexts.

In conclusion, the toolkit not only enhances the precision and efficiency of motion analysis via multi-body part tracking but also democratizes access to advanced analytical techniques. Its integration with DeepLabCut streamlines the research workflow, from data acquisition to in-depth analysis and visualization, facilitating a deeper understanding of animal behavior. As the scientific community continues to embrace data-driven methodologies, tools like this will play an increasingly vital role in uncovering the intricacies of movement and behavior.

Researchers are encouraged to utilize this toolkit to its full potential, contributing to its evolution through feedback and modifications. The hope is that it will not only serve as a valuable resource for individual projects but also inspire further innovation in the realm of motion analysis. With continued development and community support, the Motion Analysis Toolkit for DeepLabCut is poised to remain at the forefront of behavioral research technology.

How to Cite:

To reference this toolkit in your work, please cite as follows:

Augustine, F. (2024) "DeepLabCut-Analysis-Jupyter-Scripts" (v0.1.0). Zenodo. doi: 10.5281/zenodo.10866995. https://github.com/farhanaugustine/DeepLabCut-Analysis-Jupyter-Scripts

Or

Augustine, F. (2024). DeepLabCut-Analysis-Jupyter-Scripts (v0.1.0). Zenodo. https://doi.org/10.5281/zenodo.10866995

License: