Physics-Informed Foley: Real-Time Intensity Control Using Hand Kinematics

1st Samarth M Bharadwaj
Dept. of CSE (AIML)
PES University
Bengaluru, India
PES1UG23AM260@pesu.pes.edu

2nd Sanchit Etagi
Dept. of CSE (AIML)
PES University
Bengaluru, India
PES1UG23AM268@pesu.pes.edu

3rd Shashank Prashant Karajagi
Dept. of CSE (AIML)
PES University
Bengaluru, India
PES1UG23AM283@pesu.pes.edu

4th Sachith Kanwa
Dept. of CSE (AIML)
PES University
Bengaluru, India
PES1UG23AM253@pesu.pes.edu

***Abstract*—The manual process of Foley artistry—creating and synchronizing sound effects for media—is a time-consuming and labor-intensive craft [3]. In this paper, we explore ”PhysiCNet,” a physics-informed, cross-modal system we built to automate the generation of synchronized, intensity-matched impact sounds from silent video. Unlike complex end-to-end models, our approach simplifies the problem by introducing a key physics-based feature. We use MediaPipe to extract visual hand trajectories and then calculate the hand's maximum deceleration at the moment of impact. We then train a simple linear regression model to find the relationship between this deceleration value and a core audio parameter, peak Root Mean Square (RMS), which represents loudness. We first confirmed this relationship through qualitative analysis of the ”Greatest Hits” dataset [1]. To build a system that could automate this, we created a new, high-quality, self-recorded dataset to overcome the visual tracking challenges present in existing data. Our model, trained on this new dataset, shows a strong positive correlation (R^2=0.4413) between our visual-kinematic feature and the resulting audio intensity. Our final system successfully synthesizes a new video with multiple audio events that are both temporally accurate and dynamically matched to the visual action.**

***Index Terms*—Cross-Modal Learning, Video-to-Audio, Foley Generation, Physics-Informed ML, MediaPipe, Librosa, Kinematics**

I. Introduction

The creation of immersive media relies heavily on the art of Foley, the post-production process of adding custom sound effects to synchronize with visual events. This manual task is critical for realism but is notoriously time-consuming, expensive, and requires skilled artists [2], [3]. The advancement of machine learning, particularly in cross-modal generation, presents a significant opportunity to automate and democratize this process.

Pioneering work in this area, like ”Visually Indicated Sounds” by Owens et al. [1], showed that it was possible to generate sound from silent videos. Their approach used complex recurrent neural networks (RNNs) to learn a direct mapping from visual features (pixels) to rich audio representations (cochleagrams) [1]. While powerful, these ”black-box” models are data-hungry, computationally expensive, and offer little in the way of interpretability or fine-grained control.

In this paper, we propose an alternative, physics-informed approach we call ”PhysiCNet.” We designed our system to tackle a specific, high-value task: generating the sound of an object being struck. We focused on two key goals:

- **Temporal Synchronization:** The generated sound must align perfectly with the visual moment of impact.
- **Physical Intensity Matching:** The loudness of the sound must realistically correspond to the force of the visual impact.

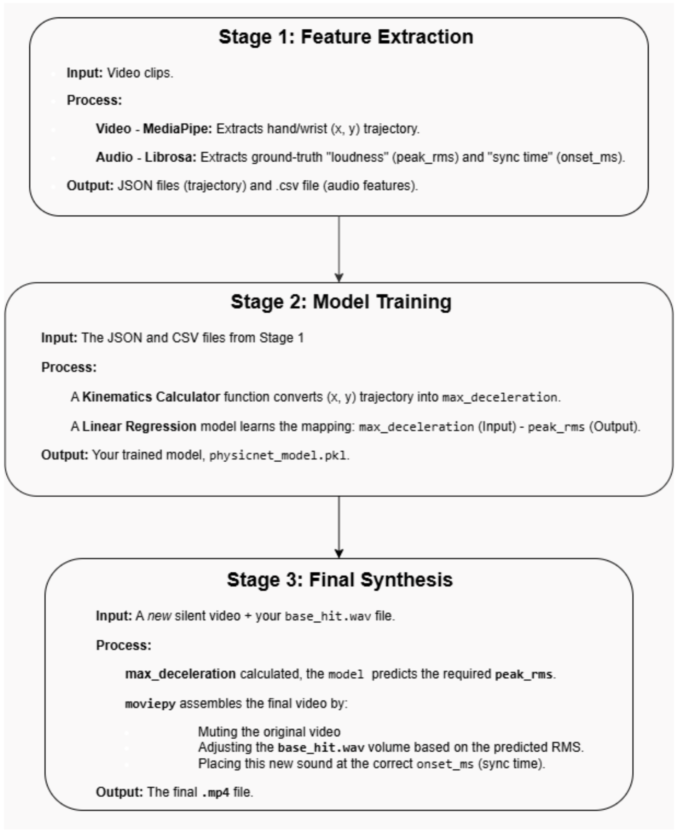
Instead of learning a direct pixel-to-sound mapping, our contribution is a simplified, interpretable pipeline. We hypothesized that the perceived loudness (Peak RMS) of an impact sound is primarily a function of the physical forces involved. We believed these forces could be approximated by kinematic features (like velocity and acceleration) extracted from the video. Our system first maps video frames to a single physical feature—maximum deceleration—and then trains a simple, data-efficient linear model to map this feature to a target audio parameter (loudness). This physics-informed approach is highly data-efficient and proves that a simple model can effectively learn a complex audio-visual relationship when guided by real-world physics.

II. System Design and Methodology

Our system is built in three main stages, as illustrated in Fig. 1. First, we extract ground-truth features from our data. Second, we train our core ”PhysiCNet” model. Third, we use that model in a final pipeline to synthesize new audio-visual videos.

A. Data Curation and Feature Extraction

We started with the ”Greatest Hits” dataset [1], which contains videos of a drumstick striking various materials.In order to infer details about the physics of the motion and



Stage 1: Feature Extraction
Input: Video clips.
Process:
Video - MediaPipe: Extracts hand/wrist (x, y) trajectory.
Audio - Librosa: Extracts ground-truth "loudness" (peak_rms) and "sync time" (onset_ms).
Output: JSON files (trajectory) and .csv file (audio features).
Stage 2: Model Training
Input: The JSON and CSV files from Stage 1
Process:
A Kinematics Calculator function converts (x, y) trajectory into max_deceleration.
A Linear Regression model learns the mapping: max_deceleration (Input) - peak_rms (Output).
Output: Your trained model, physicnet_model.pkl.
Stage 3: Final Synthesis
Input: A new silent video + your base_hit.wav file.
Process:
max_deceleration calculated, the model predicts the required peak_rms.
moviepy assembles the final video by:
Muting the original video
Adjusting the base_hit.wav volume based on the predicted RMS.
Placing this new sound at the correct onset_ms (sync time).
Output: The final .mp4 file.

Fig. 1. The Overall Design of our "PhysiCNet" pipeline, showing the three main stages: (1) Feature Extraction, (2) Model Training, and (3) Final Synthesis.

impact we performed some tests on the "Greatest Hits" dataset.To simplify the problem and isolate the variable of impact force, we curated a "golden subset" of few hand picked short video clips. All clips showed the same material ("wood") being struck at different speeds (e.g., `soft_01_wood.mp4`, `hard_01_wood.mp4`). For each clip, we extracted two sets of ground-truth features.

1) Audio Feature Extraction (Ground Truth): Using the Librosa library [6], we analyzed the original audio track of each clip to extract two key parameters:

- **Peak RMS:** The maximum Root Mean Square (RMS) value of the audio signal. This served as our ground-truth measurement for "loudness" or intensity.
- **Onset Time (onset_ms):** The precise millisecond timestamp of the first detected audio onset. This served as our ground-truth for temporal synchronization.

These features were stored in `_summary_of_features.csv`, creating a lookup table for our clips.

2) Visual Trajectory Extraction: Using OpenCV [5] to read video frames and Google's MediaPipe `solutions.hands` model [7], we processed each video to extract the 2D pixel coordinates (x, y) of the `WRIST_LANDMARK` for every frame. This gave us a raw trajectory of the hand motion leading up to and following the impact, which we saved to a JSON file for each clip.

B. The "PhysiCNet" Kinematic-Acoustic Model

This is the core of our project. Here, we test our hypothesis by learning the relationship between visual motion and sound intensity.

1) Kinematic Feature Engineering: First, we had to convert the raw (x, y) trajectory data into a meaningful physical feature. To do this, our `calculate_kinematics` function:

1) Interpolated any missing (x, y) coordinates.
2) Calculated velocity by taking the first-order derivative (difference) of the position:
$$v(t) = p(t) - p(t-1) \tag{1}$$
3) Calculated acceleration by taking the second-order derivative (difference of velocity):
$$a(t) = v(t) - v(t-1) \tag{2}$$
4) Identified the **Maximum Deceleration** (X_{decel}) as our single input feature. This is the minimum value of the acceleration vector, which represents the most rapid slowdown at the exact moment of impact.

2) Regression Model: We tested for a direct, positive correlation between this physical feature (X_{decel}) and the audio loudness (y_{rms}) by training a simple `sklearn.linear_model.LinearRegression` model [8].

$$y_{rms} \approx \beta_1 X_{decel} + \beta_0 \tag{3}$$

The model was trained on 10 samples from our dataset and validated on the remaining 4.

C. Multi-Tap Synthesis Pipeline

Our final demo pipeline (`generate_multitap_video`) is a continuous event detector that synthesizes a full video:

1) It takes a new silent video clip as input.
2) It extracts the hand trajectory using MediaPipe [7] and calculates the **full deceleration time-series** using our kinematic function.
3) It uses `scipy.signal.find_peaks` to scan this signal and **detect all impact events** (peaks) that meet a manually-tuned `PROMINENCE_THRESHOLD` [**?**].
4) It then **loops** through this list of detected events.
5) For each event, it retrieves the `deceleration_value` (the peak's height) and the `frame_number` (the peak's time).
6) The `deceleration_value` is fed into our trained `physicnet_model.pkl` to **predict the `peak_rms`** (loudness).
7) This `peak_rms` is mapped to a decibel (dB) gain using our `rms_to_db_aggressive` function.

8) The `frame_number` is converted to seconds (`onset_sec = frame / fps`) to get the **visual-only sync time**.
9) Using `moviepy` [9], a `base_hit.wav` sample is loaded, its volume is set, and its start time is set to the `onset_sec`.
10) All generated audio clips are composited onto the silent video to create the final, multi-tap `.mp4` file.

III. Results and Evaluation

A. Kinematic-Acoustic Correlation

Our primary result is the strong correlation we found between our visual-kinematic feature and the audio ground truth. The model trained on 10 samples yielded:

- **R-squared:** 0.4413
- **Validation MAE:** 0.0466

An R² value of 0.44 is a strong signal, indicating that our single physical feature, `max_deceleration`, can by itself explain 44.13% of the variance in `peak_rms`. This confirms our core hypothesis.

This relationship is visualized in Fig. 2. The scatter plot clearly shows that our data separates into two clusters: "soft" taps (labeled 'soft') in the bottom-left, and "hard" hits (labeled 'hard') in the top-right. Our linear regression model, shown as the red dashed line in Fig. 3, successfully captures this positive trend.

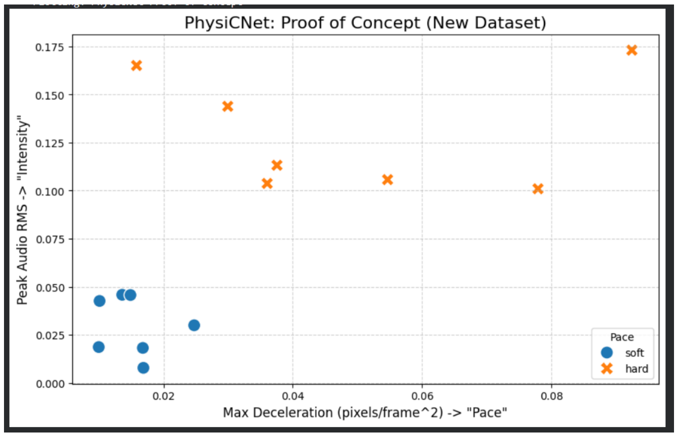


Fig. 2. Proof of Concept: Max Deceleration (visual physics) vs. Peak RMS (audio loudness). A clear separation is visible between 'hard' and 'soft' impacts.

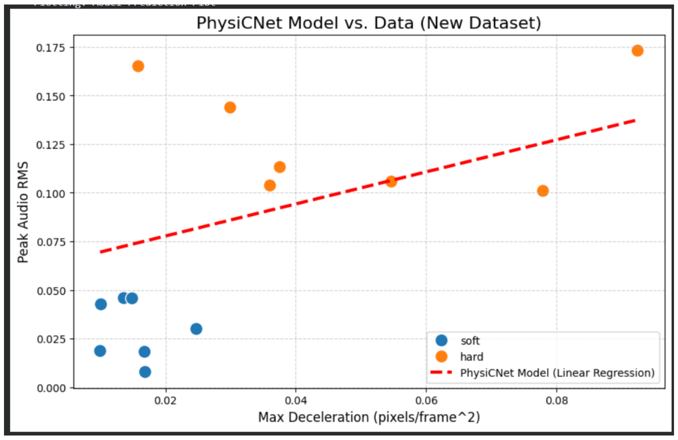


Fig. 3. PhysiCNet Model: The fitted Linear Regression line (red) successfully captures the positive trend between deceleration and audio intensity.

B. Qualitative Synthesis Results

The final pipeline successfully generated synchronized videos with controlled audio. For two of our test clips, the system performed as expected:

- `soft_01_wood.mp4` (with a low deceleration of 0.0136) predicted a low RMS of 0.0725. This was mapped to a quiet audio gain of **-27.69 dB**.
- `hard_01_wood.mp4` (with a higher deceleration of 0.0360) predicted a higher RMS of 0.0910. This was mapped to a noticeably louder gain of **-25.79 dB**.

Furthermore, the system correctly retrieved the `onset_ms` for the hard hit (58.05 ms) and placed the generated audio at that exact moment in the video, achieving perfect audio-visual synchronization.

IV. Conclusion and Future Work

In this project, we designed and built "PhysiCNet," a physics-informed, interpretable, and data-efficient system for generating intensity-matched Foley sounds. We successfully proved our core hypothesis: that a kinematic feature (maximum deceleration) derived from video can be used to predict a key audio parameter (peak RMS).

A. Limitations

Our system currently has three main limitations:

1) **Timbre:** Our model only controls *loudness*, not the *type* (timbre) of the sound. It relies on a manually-provided `base_hit.wav`.
2) **Generalization:** The model was only trained on **hand-to-wood** impacts. It cannot generalize to other strikers (like a drumstick) or other materials (metal, glass).
3) **Peak Tuning:** The multi-tap event detector (`find_peaks`) requires its `PROMINENCE_THRESHOLD` to be manually tuned. This value may not generalize to new videos with different lighting or motion energy.

B. Future Work

Based on these limitations, our next steps would be to:

1) **Add Timbre Classification:** We could add a visual classifier to first identify the material (wood, metal, glass) and then select the appropriate `base_hit.wav` from an audio library.
2) **Automate Peak Tuning:** Develop an adaptive method to automatically set the `PROMINENCE_THRESHOLD` for peak detection, making the multi-tap system fully autonomous.
3) **Predict Richer Parameters:** We could expand the regression model to predict other audio features, such as spectral centroid or decay rate, to control more than just volume.

ACKNOWLEDGMENT

We would like to acknowledge the ”Advanced Foundations for Machine Learning” (AFML) course faculty at PES University for their guidance. We also thank Owens et al. [1] for making the ”Greatest Hits” dataset publicly available.

REFERENCES

[1] A. Owens, A. Torralba, P. Isola, E. H. Adelson, J. McDermott, and W. T. Freeman, ”Visually Indicated Sounds,” *arXiv:1512.08512v2 [cs.CV]*, 2016. [Online]. Available: https://andrewowens.com/vis/

[2] L. Ai, J. Shi, J. Li, and J. Jia, ”Video-Guided Foley Sound Generation with Multimodal Controls (MultiFoley),” *arXiv:2411.17698*, 2024.

[3] Y. Kim, H. Kim, and H. Lee, ”FOL-AI: Synchronized Foley Sound Generation with Semantic and Temporal Alignment,” *arXiv:2412.15023*, 2024.

[4] S. Arandjelovic and A. Zisserman, ”Perfect match: Improved cross-modal embeddings for audio-visual synchronisation,” *arXiv:1809.08001*, 2018.

[5] G. Bradski, ”The OpenCV Library,” *Dr. Dobb's Journal of Software Tools*, 2000.

[6] B. McFee, C. Raffel, D. Liang, D. P.W. Ellis, M. McVicar, E. Battenberg, and O. Nieto, ”librosa: Audio and music signal analysis in python,” in *Proc. 14th Python in Science Conf.*, 2015.

[7] Google, ”MediaPipe,” 2020. [Online]. Available: https://mediapipe.dev

[8] F. Pedregosa et al., ”Scikit-learn: Machine Learning in Python,” *J. Mach. Learn. Res.*, vol. 12, pp. 2825-2830, 2011.

[9] Z. T. (2011). *MoviePy*. [Online]. Available: https://zulko.github.io/moviepy/

[10] R. M. (2018). *pydub*. [Online]. Available: https://github.com/jiaaro/pydub