# ECE 379K Project Report
# Movement to Music: Mapping Dance to Sound

Jackson Lightfoot
jaxon.lightfoot@utexas.edu

Nathan Paull
napaull@utexas.edu

Sydney Thompson
sydneythompson@utexas.edu

Jinze Zhao
jz24694@utexas.edu

## Abstract

*In recent years, electronic instruments have transformed the world of music by allowing for more flexibility and creativity. Our project involves the development of an accessible electronic instrument that relies only on a camera and computer vision techniques to generate sounds. After using OpenPose [3, 4, 19, 21] to generate pose estimates on a custom dataset of dance videos, impact points of limbs were detected and mapped to sounds. Although prior work such as RhythmicNet [20] used neural networks for this application, our algorithm relies on less-computationally expensive operations such as linear interpolation, Gaussian smoothing, gradient computation, peak detection, and thresholding. By quantizing impact points to reference beats, we were able to achieve perceptually high quality audio output that both sounds on beat to the music and matches the dance movements.*

## 1. Introduction

In the era of modern technology, electronic instruments and digital audio workstations (DAWs) have reshaped the world of music. A core component making such technology possible is the Musical Instrument Digital Interface (MIDI), which has standardized the communication protocol of music equipment and software [18]. Electronic instruments known as MIDI controllers use this protocol to digitally store musical notes that are played. The advantage of this format is that it stores the timing of musical notes digitally, allowing for modifications to be made to the recording in post. This enhances the ability of musicians to be creative when composing music electronically.

While MIDI has made music creation much more flexible and creative, MIDI controllers and other electronic instruments can be expensive and less accessible to those who don't know how to play instruments. Our project details the development of a virtual instrument that generates musical sounds with only a camera and computer vision techniques. The application uses pose estimation [4] to extract body movements from human dance and generates MIDI notes based on rhythmic impact points of the dance. Dance was used as the musical medium to hopefully be more intuitive to those without prior knowledge of how to play a musical instrument. Additionally, since most people have access to a basic camera on their phone or computer webcam, this project could increase the accessibility of music creation to those without MIDI controllers. We hope to have developed the beginnings of a fun and intuitive virtual instrument that will bring the creativity of music composition to a wider audience.

## 2. Related Work

### 2.1. Computer Vision and Music

The fields of computer vision and music have many intersections, such as musicians who prefer the analog response of acoustic instruments but require the robustness of electronic instruments. This motivation has led to the development of software that transforms hand movements into music as a way to bridge the gap between these two styles of music creation [9]. One of these gesture-controlled programs allows the musical effects of a keyboard to be controlled by hand speed and position, which are recorded whenever the player's hands enter the region of the instrument [1]. There are even programs that allow the user to customize their instrument to a specific preference, such as selecting which plates may exist on a virtual xylophone [14].

A common problem for virtual instruments is latency, as it takes time to estimate and translate features present in motion such as pose and velocity. Fast gesture recognition methods have been proposed to combat these expensive computational requirements, allowing these methods to work quicker and even in real-time scenarios [17]. A more direct method for combining acoustic and electronic mu-

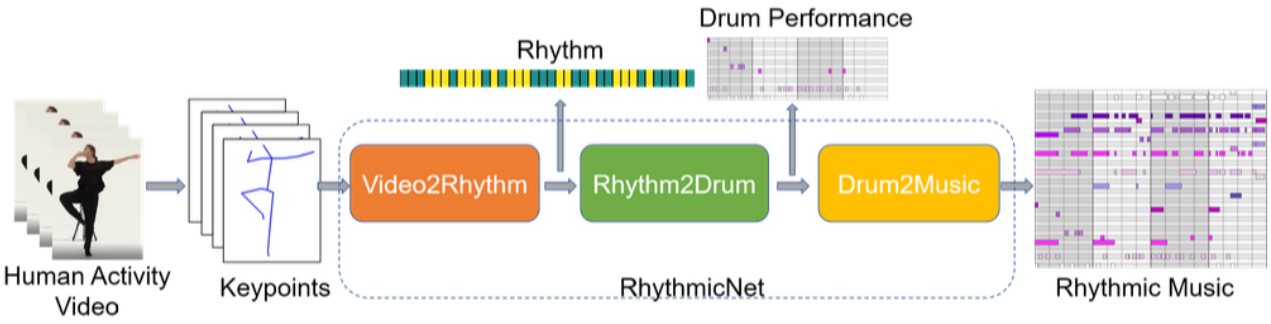


Figure 1. RhythmicNet system overview [20]

sic is the use of computer vision for automatic piano transcription [13]. Rather than generating and interacting with artificial instruments in a virtual space without tactile feedback, automatic transcription allows the player to still use an acoustic instrument.

The movements used to control all of the previously mentioned implementations tend to be rather fine, behaving similarly to traditional forms of instrumentation. Most people lack the training to be able to execute these fine gestures and tend to interact with music through more coarse movements that involve the whole body. By extracting this coarse movement, a more universal set of features can be generated [8]. With these larger movements, it is important to accurately capture the subject, limiting occlusion or clipping from poor video capture. The most ideal method for insuring video quality is environmental control, limiting the subject's range as a method of limiting variables [22]. An additional benefit of more coarse motion is that it is often easier to perform some combination of movements at the same time. In the realm of virtual music, this allows the player to control multiple instruments rather than having to focus on any single instrument [2]. Coarse motion detection methods require segmentation of a subject's body in order to isolate the movement of each limb or body part [16]. Most research that seeks to combine music and movement is bound to traditional methods of music generation, imitating instruments and musicians. Computer vision, however, allows for a much more creative range of inputs, allowing for more free form movements such as dance to be used to generate music.

The prior work most similar to our project is RhythmicNet, a series of networks that creates soundtracks for silent videos of human movement by extracting skeletal information [20]. While this network was tested on dance movements, it also works on "in the wild" human movement. RhythmicNet includes the following three sequential components, visualized in figure 1.

1. A Video2Rhythm block, which can predict the beat of music from rhythmic human movement.
2. A Rhythm2Drum block, which interprets the beat from the previous block into a drum track. The drum track is represented by three matrices: hits, velocity, and offsets. These three matrices are converted to MIDI at the end of this stage.
3. A Drum2Music block, which enriches the drum track into a composition with multiple instruments. It involves an encoder-decoder architecture, which takes the drum track as input and outputs the soundtrack of another instrument.

While our project differs from this work in a few ways, such as using videos that already include music, we still drew inspiration from this work and built upon its ideas.

### 2.2. Real-Time Pose Estimation

Human pose estimation involves approximating the coordinates of body limbs from images or video. Recent pose estimation models provide real-time analysis and can produce 2D and 3D reconstructions. Additionally, multi-person pose estimation is able to accurately analyze videos with multiple people per frame.

One pose estimation model, VNect, can reconstruct a 3D skeletal model from a camera, but results in high computational costs [15]. Using a single RGB camera, this model can map an image to a human skeleton in real-time. VNect uses a convolutional neural network (CNN) and skeletal fitting to compute pose estimation. VNect can also improve performance by estimating body proportions using the CNN predictions of the beginning frames. However, VNect requires a large amount of resources to run in real-time, with live applications of the system running at 30 GHz. Furthermore, the lack of 3D pose datasets limits the accuracy of 3D pose estimation [15].

Instead of more traditional pose estimation techniques like pictorial structures and hierarchical models, convolutional architectures have emerged as a promising source for real-time accurate pose estimation. Convolutional Pose Machines (CPMs) use image features and belief maps to deter-

mine the relationship between segments of the image [21]. With this knowledge, CPMs can provide real-time analysis of images and identify key points. One model that uses convolution networks achieves more precise results when testing with standard datasets, such as MPI, LSP, and FLIC [21]. This CPM can also learn from new datasets to improve accuracy or create a new identification model. However, this implementation experiences errors when multiple people are in close proximity.

Multi-person pose estimation attempts to identify and reconstruct 2D skeletons when multiple people are in an image. One implementation uses Part Affinity Fields (PAFs) to reduce the computation cost of the analysis, leading to real-time accurate results [4]. Whereas other multi-person models' runtimes exponentially increase with the number of people, this model stays efficient using PAFs and a bottom-up approach. Although this model can provide real-time estimations, it loses accuracy and scores lower on the COCO dataset due to imprecise localization [4].

While many advancements in pose estimation have been made, there are still many areas available for growth. These techniques are still limited by their computational costs, the accuracy of keypoints, and reconstruction methods. For our project, we used a publicly available pose estimation library, OpenPose [3, 4, 19, 21].

## 3. Methods

### 3.1. Custom Dataset Collection

Since our methodology involves using a pre-trained pose estimation model and heuristic techniques, a large, labeled dataset was not required for our project. We collected a small, custom dataset of 10 human dance videos for our project, with an example screenshot in figure 2. 4 voluntary subjects were recruited (including one of the authors), and each picked 2-3 songs from a variety of genres to dance to for approximately a minute each. Videos were recorded using an iPhone 13 Pro Max at 30 fps and 1920 by 1080 pixels. A tripod was used to stabilize the video, as a still camera is one of the requirements for our algorithm. A `.csv` file was also created that contains details for each video, such as the song title, song tempo, and a reference beat timestamp used for impact quantization.

### 3.2. Efficient Pose Estimation

Pose estimation is a well-researched task with many open source libraries and algorithms that provided a foundation for our research. We conducted a survey of pose estimation libraries to evaluate their accuracy, estimation speed, and ease of setup. The first key decision we made from this survey was to select an 18-point (or fewer) 2D pose estimation algorithm. Because our movement analysis focuses on the locations of extremities such as the wrist and ankle,


Figure 2. Example screenshot from custom dataset

additional key points in the torso, head, and fingers are not necessary, providing much more detail than we need. Our survey consisted of the following libraries:

- OpenPose [3, 4, 19, 21]
- AlphaPose [5, 6, 12]
- OpenPifPaf [10, 11]

From these libraries, our analysis of accuracy and estimation speed showed that AlphaPose and OpenPose were the best options available. Both of these libraries offer real-time 17/18-point 2D pose estimation algorithms that operate in the realm of 20 frames per second, meaning these libraries would be feasible for a real-time implementation of our project. From our analysis, OpenPose provides slightly higher speed while AlphaPose provides slightly higher accuracy and fewer dropped frames. The increased accuracy initially made AlphaPose more attractive, as a more accurate pose estimation would provide a much better foundation for further analysis. However, OpenPose provided a much quicker installation and setup routine which allowed us to start generating keypoints and in turn meet our research deadlines much more quickly. Thus, we used OpenPose to generate offline pose estimations to seed our impact point detection research. An example pose skeleton generate by OpenPose on a video frame from our dataset can be seen in figure 3.

### 3.3. Detecting Impact Points

Impact point detection was one of the core components and contributions of our project. Inspired by traditional computer vision algorithms learned in class such as SIFT

Figure 3. OpenPose estimate example

and optical flow, we propose a non-deep-learning-based pipeline that can extract impact points given pose estimates. We believe that our methods could also support impact point detection in real-time scenarios, although this would require modifications to our signal processing techniques and has yet to be tested. In this section, we discuss preprocessing steps and our overall impact detection algorithm.

#### 3.3.1 Preprocessing

We first realized that OpenPose is not able to obtain a prediction for every video frame. It occasionally fails to detect the pose of some parts of the human body, typically due to reasons such as low-luminance and occlusion. OpenPose outputs coordinates of $(0, 0)$ for missing frames, which causes the point to jump if not handled properly. Inspired by the Horn–Schunck optical flow algorithm [7], we assumed that human movements are smooth in consecutive frames. Thus, we used linear interpolation to interpolate missing values as an initial preprocessing step.

Secondly, we realized OpenPose suffers from noisy predictions. Despite a limb staying relatively still in space, the position estimates will vary slightly over time. This noise heavily corrupts gradient computation, so our second preprocessing step involves Gaussian filtering the pose estimates over time to smooth the signals. Doing so provides more stable gradient estimates and improved the effectiveness of our algorithm.

#### 3.3.2 Algorithm

Of the returned pose estimates from OpenPose, our algorithm only considers the left hand, right hand, left foot, and right foot. For each limb, we have a time series of its pixel coordinates $(x, y)$ across frames of the input video. With these coordinate signals as input, our algorithm is summarized below. Additionally, pseudocode can be seen in algorithm 1.

1. Interpolate pixel coordinates using linear interpolation.
2. Gaussian filter the interpolated signals to smooth it with $\sigma = 5$.
3. Take the gradient of the smoothed coordinate signals and compute the overall gradient magnitude.
4. Gaussian filter the gradient magnitude to smooth it with $\sigma = 3$.
5. Take the gradient again to obtain a 2nd order gradient.
6. Apply peak detection to find minima in the 2nd order gradient, with a minimum distance of 10 and thresholds of $-0.4$ and $-0.1$ for hands and feet.
7. Impacts are detected at these minima if the 1st order gradient exceeds thresholds of 3 and 1 for hands and feet.

**Algorithm 1** Impact Detection

Input: coordinate arrays $x, y$
$x_i, y_i = \texttt{interp}(x), \texttt{interp}(y)$
$x_s, y_s = \texttt{gaus\_smooth}(x_i), \texttt{gaus\_smooth}(y_i)$
$x', y' = \texttt{gradient}(x_s), \texttt{gradient}(y_s)$
$\|\nabla\| = \sqrt{x'^2 + y'^2}$
$\|\nabla\|_s = \texttt{gaus\_smooth}(\|\nabla\|)$
$\|\nabla\|' = \texttt{gradient}(\|\nabla\|_s)$
$impact\_idx = \texttt{find\_peaks}(-\|\nabla\|')$
$impact\_idx = impact\_idx\big[\|\nabla\|_s \geq thresh\big]$
Output: $impact\_idx$

Intuitively, impact points should occur during a dramatic limb motion, but we don't want to have multiple consecutive impacts detected across one continuous motion. Instead, we want to obtain the precise time a limb has an impact, typically characterized by an abrupt stop of motion. Based on this logic, we detect minima of the 2nd order gradient of the pose estimates, which correspond to a dramatic decrease in the speed of the limb. To avoid detecting impacts too close together, a minimum distance of 10 points between minima is set. Additionally, thresholds of $-0.4$ and $-0.1$ are used in minima detection for hands and feet, respectively. Different thresholds are used since hands typically experience more dramatic, faster movement in human dance. These hyperparameters were tuned to optimize performance on our custom dataset, but different hyperparameters may be optimal for different subjects, camera distances, and genres of dance.

Solely detecting impacts based on the minima of the 2nd order gradient does work, but too many impacts are detected for small movements. To improve the robustness of our impact detection algorithm, these minima are further refined by thresholding based on the smoothed 1st order gradient magnitude. These thresholds were set to be 3 and 1 for hands and feet, respectively. This additional condition ensures that a dramatic motion is indeed detected. The final output of the algorithm denotes which limbs have impacts on each frame via a boolean array, along with corresponding timestamps of the video.

### 3.4. Impact Quantization

Digital audio workstations (DAWs) typically have the ability to temporally quantize MIDI notes to the beat. A MIDI quantization example in a typical DAW is shown in figure 4. Since our implementation did not format output as true MIDI notes, the pipeline itself implements impact quantization to match impacts to the nearest beat or beat subdivision. Quantization can be set to any level of subdivision, such as quarter notes, eighth notes, or sixteenth notes. Note that this implementation requires the song in the video to have a consistent and steady tempo, otherwise quantization will slowly diverge from the true beat. Equations 1 and 2 define how impact quantization was performed with the following variable definitions:

- $dt$: time delta between beat subdivisions
- $bpm$: tempo in beats per minute
- $subdiv$: number of subdivisions per beat
- $q_n$: quantized timestamps of impacts
- $t_n$: original timestamps of impacts
- $t_{ref}$: beat reference timestamp

$$dt = 60/(bpm * subdiv) \tag{1}$$

$$q_n = \texttt{round}((t_n - t_{ref})/dt) * dt + t_{ref} \tag{2}$$

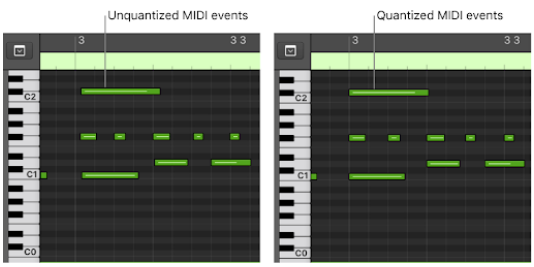


Figure 4. MIDI quantization example

### 3.5. Mapping Impact Points to Sound

To construct the final output audio, we take the quantized impact points of all four limbs and create sounds at those timestamps using the `pydub` Python library. To start, a baseline audio track is created. In our case, this is the original audio from the video, but any audio such as a silent track could be used. Additionally, each limb is assigned a short soundbite stored as a `.wav` file. For our project and the demo, we used a hi-hat and cowbell sound for hands, and a kick drum and snare drum sound for feet. However, any instrument sounds could be used, including melodic instruments.

After impact quantization, each limb has an array of quantized impact points, represented as the video timestamp in seconds that the impact was detected. For each impact point, a track is created that contains silence up until the impact timestamp, at which the relevant soundbite is added. After this audio track containing one impact point is created, it is overlayed with the base audio. Creating audio one impact at a time prevents the audio of one limb from overriding the audio of another. Additionally, this allows longer soundbites to be used without obscuring or covering up other impact points. The overlaying process is repeated for every impact point across all limbs. The constructed `.wav` file is then saved. Finally, the audio is recombined with the original video using the `moviepy` Python library.

### 3.6. Experiments and Results

In the development of our impact detection algorithm, we performed multiple experiments to refine its details. Our three primary impact detection attempts are summarized as follows:

1. Impacts are detected when the 1st order gradient exceeds a certain threshold.
2. Impacts are detected at minima of the 2nd order gradient.
3. Both conditions combined.

Method 1 suffered from detecting far too many impact points. Although multiple detections were pruned in impact quantization, a low threshold would result in constantly detecting impacts, and a high threshold would result in impacts only being detected for a few extreme, dramatic motions. Method 2, on the other hand, performed much better. Specifically determining when movements end by looking at minima of the 2nd order gradient proved to be much more effective. However, this method suffered from detecting impacts during small, abrupt movements where impacts should not occur. Method 3 solves this issue by adding back the 1st order gradient condition. This allows impacts to only be detected at the end of large movements.

Figure 5 displays the impact detection results visualized for the right foot of video "j2." The plot on the left displays $(x, y)$ pose coordinates, the plot in the center displays the 1st order gradient, and the plot on the right displays the 2nd order gradient. Detected impacts are overlayed to show the effectiveness of the algorithm. 5 dramatic motions can be seen in both the 1st order and 2nd order gradient plots, and impacts are detected for all 5 of them. The benefit of Gaussian smoothing can also be seen in these plots.

While these particular results are successful, the algorithm struggles to detect meaningful impacts in more smooth, flow-like styles of dance. Additionally, the hyperparameters used are dependent on the distance of the camera from the subject. Performance would likely degrade if the camera was closer or farther away, as distance values would change in magnitude. Nonetheless, the performance of our algorithm is comparable to that of RhythmicNet [20], the related work most similar to our project. Unfortunately, numerical metrics have not been defined for this application. However, a perceptual comparison of our results to RhythmicNet reveals that our implementation is much more pleasant musically, primarily due to impact quantization. Additionally, notes appear to be better matched to the motions of the dance for most cases in our implementation. Considering the much lower computational costs of our algorithm, we consider this project to be a huge success for this relatively untouched field.

### 3.7. Demo

A code demo of the project is available in the public GitHub repository. Instructions to run the demo are provided in the readme, and it requires Anaconda to install relevant packages as well as Python 3.8 to run. Additionally, a video demo is available on YouTube.

## 4. Discussion

### 4.1. Team Work Assignment

#### 4.1.1 Jackson Lightfoot

At the start of the project, I conducted much of the initial research of related work that combines computer vision and music. For the proposal and report, I performed the final editing and formatting on top of the sections I individually wrote. Looking towards implementation, I developed the overall system design for the project. This involved writing initial function skeletons for the different components of our project to ensure the work everyone did individually ran smoothly when combined. I also implemented impact quantization and contributed in the development of impact detection. Additionally, I wrote scripts to visualize the impact detection process to verify our algorithm works as intended. Lastly, I organized our individual Python notebooks into `.py` files and setup the demo, including the `demo.py` script itself and a package requirements file.

#### 4.1.2 Nathan Paull

My work centered around research and implementation of efficient and accurate pose estimation. I performed the survey of current impact detection libraries and ultimately made the decision to use OpenPose. It was then my job to use OpenPose to generate accurate keypoints on the sample videos, generating both video and coordinate-based outputs. I also performed some of the literature review shown in the related work section to help guide the initial construction of our project.

#### 4.1.3 Sydney Thompson

For this project, I implemented the audio file construction. I wrote the code that constructed the audio given the quantized impact points, which then exported the audio. Additionally, I made the functions that stripped the original audio from the file and the function that took an audio and video path and combined them. Research-wise, I looked at prior related work for real-time pose estimation. I also researched which python library would fit the project best for creating audio. Additionally, I helped brainstorm and come up with ideas that would make the impact point detection better.

#### 4.1.4 Jinze Zhao

For this project, I did several literature reviews and found several similar projects and GitHub repositories. I realized that neural-network-based impact detection might be very challenging for us, so I chose to use traditional methods inspired by what we learned in class. Later in the implementation, I implemented the impact point detection together with Jackson.

### 4.2. Future Work

There is an abundant amount of future work for this project that would allow for better application of this technology as an actual instrument for creating music. While our method of impact detection took an unsupervised, offline approach, there are many other methods for calculating impact points that are more accurate. A potential alternative is the use a supervised learning approach by first generating a set of pre-labeled impact points that would form the foundation for machine learning algorithms to perform impact detection. It is also important to develop an online, real-time implementation, allowing the artist to hear their music live as they create it, similar to a regular instrument. A real-time approach to impact detection would likely require impacts to be predicted ahead of time to prevent issues with latency. Our methods of using the 1st and 2nd

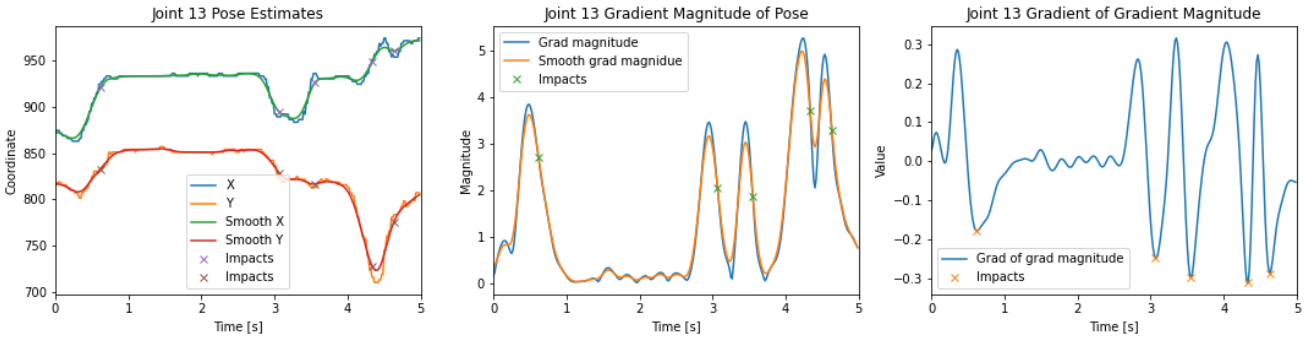


Figure 5. Impact detection results visualized for the right foot of video "j2"

order gradient could be useful for the task of predicting motion a few frames ahead. However, minima detection would likely need to be replaced with an alternative method such as detecting zero-crossings of the 2nd order gradient, which would correspond to a limb's motion slowing down. Although challenging, we believe a real-time implementation is feasible with the groundwork our research has laid. If impact detection could be reliably done in an online scenario, then our initial goal of creating a seamless interaction between the digital and physical world of music would become possible.

Another important addition to a real-time implementation is formatting the algorithms output in true MIDI format. Transcribing the generated music into a readable format such as MIDI would allow for connection to a digital audio workstation (DAW) and fully allow musicians to take advantage of digital tools while using physical expression for music generation. Python libraries do exist for controlling MIDI output, such as the `py-midi` Python library. Configuring a real-time version of our implementation to a DAW would allow for virtually unlimited creativity for music creation.

While impact detection is paramount, music transcription and interpretation is another area of great importance. Since beats are typically equally spaced in time in most music, we were able to quantize beats in a given piece of music using an initialized beat reference timestamp and the song's tempo. However, there are many songs that change time signatures or tempo, which would cause our methods to diverge from the true beat. Our quantization implementation requires each video to be labeled with a given speed and initialized timestamp, which would be incredibly inconvenient for musicians using this application. If beat quantization could instead be done automatically, it would remove the need for human intervention and manual data entry. While algorithms for detecting the pulse of music do exist, we determined this to be outside the scope of our project.

Another potential expansion of the project is further customizing the way impact points are translated to sound. For example, the volume of each soundbite could be determined by how fast the limb was moving before the impact point. Each limb's impact point sound could also be changed based on the location of the impact, such as raising or lowering the pitch of a melodic instrument depending on the vertical coordinate of impact. Another functionality could be playing a different sound when a limb remains stagnant. If using a continuous synth sound, the 1st order gradient magnitude could be directly mapped to volume, creating a unique effect where as limbs move faster, the sound crescendos. If impact point detection is expanded to a group, then each person could have their own instrument sounds, or even different notes to create harmonies. Among these ideas, there are many creative ways to expand the functionality of a virtual instrument controlled by dance and human pose.

### 4.3. Conclusion

In this project, we made numerous contributions to the relatively unexplored field of generating musical sounds from human pose. Firstly, we conducted a thorough literature review of related work, both in pose estimation algorithms and works combining computer vision techniques with music-related applications. We collected a custom dataset for the purpose of our project and used OpenPose [3,4,19,21] to generate pose estimates. The primary contribution of our project was our novel impact detection algorithm, inspired by techniques used in classical computer vision algorithms. We also introduced a simple impact quantization approach that greatly increased the perceptual audio quality of our generated audio compared to RhythmicNet [20]. Lastly, we provided a detailed description of many possibilities for future work that could be done to improve and expand this application. Although there is much left to be accomplished and explored in this field, we believe our work provides a solid foundation for mapping human dance to music.

### 4.4. Acknowledgements

Acknowledgements to Michael Schnebly for some of the ideas behind this project.

## References

[1] AMRR Bandara. *A music keyboard with gesture controlled effects based on computer vision*. PhD thesis, Thesis, University of Sri Jayewardenepura, 2011. 1

[2] Reinhold Behringer. Conducting digitally stored music by computer vision tracking. In *First International Conference on Automated Production of Cross Media Content for Multi-Channel Distribution (AXMEDIS'05)*, pages 4–pp. IEEE, 2005. 2

[3] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, and Y. A. Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019. 1, 3, 7

[4] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *CVPR*, 2017. 1, 3, 7

[5] Hao-Shu Fang, Jiefeng Li, Hongyang Tang, Chao Xu, Haoyi Zhu, Yuliang Xiu, Yong-Lu Li, and Cewu Lu. Alphapose: Whole-body regional multi-person pose estimation and tracking in real-time. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022. 3

[6] Hao-Shu Fang, Shuqin Xie, Yu-Wing Tai, and Cewu Lu. RMPE: Regional multi-person pose estimation. In *ICCV*, 2017. 3

[7] B.K.P. Horn and B.G. Schunck. Determining optical flow. *Artificial intelligence*, 17(1-3):185–203, 1981. 4

[8] Kelly Jakubowski, Tuomas Eerola, Paolo Alborno, Gualtiero Volpe, Antonio Camurri, and Martin Clayton. Extracting coarse body movements from video in music performance: A comparison of automated computer vision techniques with motion capture data. *Frontiers in Digital Humanities*, 4:9, 2017. 2

[9] Chris Kiefer, Nick Collins, and Geraldine Fitzpatrick. Phalanger: Controlling music software with hand movement using a computer vision and machine learning approach, 2009. 1

[10] Sven Kreiss, Lorenzo Bertoni, and Alexandre Alahi. PifPaf: Composite Fields for Human Pose Estimation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 3

[11] Sven Kreiss, Lorenzo Bertoni, and Alexandre Alahi. OpenPifPaf: Composite Fields for Semantic Keypoint Detection and Spatio-Temporal Association. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–14, March 2021. 3

[12] Jiefeng Li, Can Wang, Hao Zhu, Yihuan Mao, Hao-Shu Fang, and Cewu Lu. Crowdpose: Efficient crowded scenes pose estimation and a new benchmark. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10863–10872, 2019. 3

[13] Jun Li, Wei Xu, Yong Cao, Wei Liu, and Wenqing Cheng. Robust piano music transcription based on computer vision. In *Proceedings of the 2020 4th High Performance Computing and Cluster Technologies Conference & 2020 3rd International Conference on Big Data and Artificial Intelligence*, pages 92–97, 2020. 2

[14] Teemu Mäki-Patola, Juha Laitinen, Aki Kanerva, and Tapio Takala. Experiments with virtual reality instruments. In *Proceedings of the 2005 conference on New interfaces for musical expression*, pages 11–16, 2005. 1

[15] Dushyant Mehta, Srinath Sridhar, Oleksandr Sotnychenko, Helge Rhodin, Mohammad Shafiei, Hans-Peter Seidel, Weipeng Xu, Dan Casas, and Christian Theobalt. Vnect: Real-time 3d human pose estimation with a single rgb camera. *ACM Transactions on Graphics*, 36(4), 2017. 2

[16] Joseph Paradiso and Flavia Sparacino. Optical tracking for music and dance performance. *Optical 3-D Measurement Techniques IV*, pages 11–18, 1997. 2

[17] Alejandro Rosa-Pujazón, Isabel Barbancho, Lorenzo J Tardón, and Ana M Barbancho. Fast-gesture recognition and classification using kinect: an application for a virtual reality drumkit. *Multimedia Tools and Applications*, 75(14):8137–8164, 2016. 1

[18] Joseph Rothstein. *MIDI: A comprehensive introduction*, volume 7. AR Editions, Inc., 1992. 1

[19] Tomas Simon, Hanbyul Joo, Iain Matthews, and Yaser Sheikh. Hand keypoint detection in single images using multiview bootstrapping. In *CVPR*, 2017. 1, 3, 7

[20] Kun Su, Xiulong Liu, and Eli Shlizerman. How does it sound? *Advances in Neural Information Processing Systems*, 34:29258–29273, 2021. 1, 2, 6, 7

[21] Shih-En Wei, Varun Ramakrishna, Takeo Kanade, and Yaser Sheikh. Convolutional pose machines. In *CVPR*, 2016. 1, 3, 7

[22] Christopher R Wren, Flavia Sparacino, Ali J Azarbayejani, Trevor J Darrell, Thad E Starner, Akira Kotani, Chloe M Chao, Michal Hlavac, Kenneth B Russell, and Alex P Pentland. Perceptive spaces for performance and entertainment untethered interaction using computer vision and audition. *Applied artificial intelligence*, 11(4):267–284, 1997. 2