Maynooth University

– Department of Music –

Real-Time Generative Music from Object Tracking and Computer Vision

Thesis Submitted in Partial Fulfilment for the Degree of

Master of Science in Sound and Music Computing

by

Ken Kobayashi

Student Number: 24253634

Supervisor : Victor Lazzarini

Declaration

I, Ken Kobayashi, hereby declare that this thesis, entitled *Real-Time Generative Music from Object Tracking and Computer*, is my own original work. It has been composed by myself and has not been submitted, in whole or in part, for any other degree or professional qualification at this or any other institution.

All sources of information and ideas obtained from the work of others have been duly acknowledged in the text and listed in the bibliography.

The word count of this thesis is 11,706 words.

Signed: Ken Kobayashi

Date: August 31, 2025

Acknowledgments

I would first like to express my deepest gratitude to my family for their unwavering support and for providing me with the opportunity to pursue this degree. Their constant encouragement throughout my academic endeavors has been instrumental in bringing me to where I am today.

My journey to Maynooth University is due in large part to the guidance of my undergraduate professor at Berklee College of Music, Dr. Richard Boulanger. Despite my initial reservations about postgraduate study, Dr. Boulanger inspired me to explore the wonderful Music Department at Maynooth. I have no regrets about undertaking this Master's program, and I am forever thankful for his infectious passion and immense talent in electronic music.

I am profoundly grateful to my girlfriend, Ivy Park. My spontaneous obsessions and sudden, drastic decisions undoubtedly presented challenges, yet she was consistently there for me, offering the resources and emotional support essential for me to achieve my goals.

My sincere thanks also go to my colleagues in the Master of Music Technology program. The energy and insight they brought to our classes and discussions provided not only great laughs and joy but also a vital sense of community. Though our time together was short, our camaraderie was often the highlight of my day.

Finally, I must extend my utmost appreciation to my thesis supervisor, Professor Victor Lazzarini, whose wealth of knowledge in audio programming is seemingly endless. Without his expert guidance and patience, this thesis would not have been possible, and I would not have achieved such a deep understanding of the audio world.

Abstract

This thesis presents *VideoCsound*, a novel, open-source performance system that bridges computer vision and algorithmic composition to transform real-time visual input into synchronized, generative audio. Addressing the challenge of audience interpretation in laptop-based electronic music, which often lacks the visual expressiveness of traditional instruments, this work explores the sonification of visual data as a means of creating new, intuitive forms of musical expression. The system leverages the vast and continuously growing amount of publicly available visual data from online sources like live webcams and social media, treating the visual world as a limitless source for data-driven, generative music.

The system is implemented in Python, which serves as the "glue" connecting a modular software pipeline. The core vision component utilizes the *YOLO11* object detection model, accessed via the *Ultralytics* library, which is pre-trained on the 80 object categories of the *COCO* dataset. Video input and output visualization are managed by the robust *OpenCV* library. For audio, the system employs the powerful and highly customizable *Csound* synthesis environment, controlled in real-time through its Python API. The central processing loop captures video frames, performs object detection and tracking to extract bounding box coordinates (position, width, height) and unique track IDs for each object, normalizes this data, and sends it to a running *Csound* instance via named control channels. This architecture allows users to flexibly assign unique, custom-designed *Csound* instruments to specific object classes, enabling highly tailored audiovisual interactions.

A series of four diverse demonstrations were conducted to qualitatively evaluate the system's capabilities and rigorously identify its limitations. These tests, which analyzed footage of dense urban crowds, fast-moving wildlife, vehicle traffic, and a live interactive webcam performance, exposed several key challenges. A primary issue was performance degradation, including significant framerate drops when processing high-resolution video and a consistent audio latency of approximately 500ms, which detracted from the sense of real-time interactivity. Furthermore, the reliance on a general-purpose pre-trained model led to detection inaccuracies (false positives and negatives) when confronted with a domain mismatch, such as tracking specific bird species not well-represented in the training data.

The demonstrations also yielded important insights into the relationship between mapping strategies and sonic outcomes. Stationary objects in dense scenes produced monotonous,

undifferentiated drones, highlighting the need for *Csound* instrument designs with internal modulation that can create auditory interest independent of object movement. The live webcam performance successfully reframed the system as an expressive musical interface, where the performer's manipulation of everyday objects in 3D space provided fluid control over sound. This scenario also revealed emergent performative techniques, such as using physical occlusion to intuitively start and stop sounds.

The results validate *VideoCsound* as a successful prototype for a marker-less, tangible interface that leverages the world itself for musical control. The thesis concludes by outlining critical areas for future work essential for moving the system beyond a prototype. The highest priority is the development of a graphical user interface (GUI) to make the system accessible to artists and musicians without programming expertise. Additionally, future work must focus on expanding the mappable control parameters beyond simple bounding box coordinates to include more musically relevant data, such as object velocity and acceleration, as well as the model's detection confidence score. Integrating these richer data streams will be key to unlocking more nuanced, dynamic, and musically expressive applications in interactive art, education, and data-driven auditory displays.

Contents

1 Introduction

Music and visual arts are complementary art forms, often combined to create immersive and impressive experiences. They reinforce each other; for example, flashing lights can highlight an exciting musical chorus, while ominous music can add mystery to a horror movie scene. The significance of this relationship has been further accentuated in the digital age. However, the increasing prevalence of intricate digital synthesis presents a growing challenge for audiences in interpreting art, especially as laptops, which lack the visual expressiveness of traditional instruments, increasingly supplant them [8].

Multimodal communication, which involves the simultaneous use of two distinct methods, centers on this symbiotic relationship between audio and visual information [52]. When combined, they convey information in two forms, significantly enhancing the receiver's processing. For example, the intelligibility of a speaker can be improved by up to 80% when listeners can both see and hear them [49]. In museum settings, audio integration can enhance the sense of place and immersion for specific exhibits [42].

This thesis investigates the sonification of data, or the representation of data as audio [23], by extracting it from visual media to generate auditory experiences. Data points, gathered from observing objects in video, contain inherent patterns and sequences of movement. These structures can be utilized to generate audio, thereby revealing the hidden artistry embedded within the data itself.

Visual media serves as a rich source of this data, allowing humans to quickly identify objects and perspective from just a glance [43]. Computer vision is the field that enables machines to do the same, analyzing pixel values from images or video to extract information like object type, position, and movement. This technology is rapidly evolving, largely due to advancements in deep learning [6], which allows systems to recognize patterns and detect objects [19]. This technology bridges the gap between pixels and actionable insights. In the realm of digital art, it empowers new forms of creative expression.

Among these advancements, object detection (recognizing the location and type of object in an image) and object tracking (following the movements of an object in a video) have become relatively accessible. Modern computer vision makes it feasible to extract object movement data directly from any video source and use it to generate sound. This results in algorithmically generated music that is synchronized with the source video.

The potential for generating music from video is amplified by the vast amount of publicly available visual data online, a key area of focus in Open Source Intelligence (OSINT). OSINT is the practice of collecting and analyzing publicly available information [40], used by governments to investigate cybercrime [37]. Of particular interest is the abundance of live security camera feeds available from around the globe [41]. Services like *EarthCam* provide free, live webcam footage of locations from New York City to Rio de Janeiro. Furthermore, the popularity of social media has led to an unimaginable quantity of video content; over 20 million videos are uploaded to YouTube alone daily [34].

By developing a tool that harnesses computer vision to analyze object movements in videos, whether from live security feeds, social media, or other digital sources, a near-limitless source of data can be extracted and transformed into generative music. This thesis presents the creation of a performance system, *VideoCsound*, that bridges computer vision and algorithmic composition, converting visual input into synchronized audio output. *VideoCsound* is implemented as an open-source Python application; the full source code, documentation, and demonstration videos are available in the project's GitHub repository: https://github.com/kencul/videoCsound. The system explores applications in interactive art, education, and experimental music, while addressing technical challenges and proposing future directions for research at the intersection of data, vision, and sound.

This thesis is organized as follows. Chapter 2 provides a comprehensive literature review, tracing the history of computer vision from classical algorithms to modern deep learning and examining the theoretical foundations of audio-visual mapping. It reviews prior work in both artistic and analytical applications, highlighting the opportunity for a novel marker-less, semantically-aware musical interface. Chapter 3 introduces the technology stack, detailing the specific software libraries and frameworks chosen for the project (Python, *YOLO11*, *OpenCV*, and *Csound*) and justifies how their combination creates a robust, real-time processing pipeline.

Chapter 4 describes the complete implementation of the system, named *VideoCsound*. It explains the software architecture, the template-based file structure that allows for easy project management, and the critical data flow from video input and object detection to parameter normalization and audio synthesis. Chapter 5 presents a qualitative evaluation of the system through four distinct demonstrations. These case studies rigorously test the system with diverse visual scenarios—dense crowds, fast-moving wildlife, vehicle traffic, and a live interactive performance—to assess its capabilities and expose its limitations regarding performance, accuracy, and musical expressiveness. Finally, Chapter 6 concludes the thesis by summarizing the project's contributions and the key findings from the demonstrations, discussing the primary challenges encountered, and outlining critical directions for future work, including user interface development and the expansion of mappable parameters.

2 Literature Review

This chapter reviews the key historical and technical concepts that form the foundation of this thesis. It begins by tracing the evolution of computer vision, from its origins in hand-crafted algorithms to the current paradigm dominated by deep learning. The discussion then shifts to the theoretical underpinnings of mapping visual data to sound, exploring different strategies and their implications for musical expression. Finally, the chapter examines existing audio-visual applications in both analytical and artistic performance contexts, identifying the opportunity for a novel system that leverages modern, marker-less object detection for creative expression.

2.1 History of Computer Vision

Computer vision is a rapidly advancing field of AI that enables machines to interpret and analyze visual data, mimicking human visual perception [48]. Its primary objective is to extract meaningful information from images or videos, facilitating automation in tasks such as object recognition [25]. Over the past few decades, computer vision has evolved from simple image processing techniques to sophisticated deep learning-based systems, revolutionizing industries ranging from healthcare [17] to autonomous driving [51] [53]. This section explores the historical development, foundational concepts, and modern advancements that have shaped the field.

2.2 The Classical Era

The foundational work in computer vision was defined by a pipeline of hand-crafted algorithms designed to extract specific, low-level features like edges and corners from images. The first crucial discovery was made by neurologists who observed a cat's brain responses to an array of pictures [26]. Their research shows that the brain initially processes hard edges or lines, indicating that image processing starts with basic shapes like straight edges. Consequently, early efforts concentrated on edge detection and feature recognition. This foundational research in basic image processing led to the Sobel operator, an edge detection algorithm developed in the 1970s [47].

The 1960s also saw advancements in pattern recognition. In 1958, the Perceptron was introduced [44]. It was one of the first models designed for learning patterns from visual data, laying the groundwork for connectionist approaches that would dominate the field later.

The classical era of computer vision was dominated by a pipeline of hand-crafted geometric and statistical algorithms. This approach relied on innovative yet manually designed feature detectors like SIFT [33] and SURF [2] to identify scale-invariant key points. A significant challenge for these methods was their susceptibility to noisy data and outliers. This problem was effectively addressed by robust estimation frameworks like RANSAC [15], which iteratively discarded outliers to find a consensus model. Together, this combination of feature detectors and robust estimation formed the technical backbone for breakthroughs in 3D reconstruction and image stitching [46]. However, this entire paradigm was fundamentally limited by its reliance on human ingenuity to design features; these pipelines were often brittle, failed to generalize to new domains, and struggled with non-rigid objects—limitations that would ultimately lead to the field's paradigm shift towards deep learning.

2.3 The Rise of Machine Learning

The hand-crafted pipeline of classical computer vision systems was brittle and unable to generalize. These limitations led to a paradigm shift towards machine learning, which dominates the field today [35]. Early foundations were laid in 1979 by Kunihiko Fukushima's Neocognitron, a hierarchical multilayered neural network that could decipher handwritten text [16]. Later in 1989, the ideas presented in Fukushima's research were first successfully combined with the back propagation algorithm for practical pattern recognition by LeCun et al. [32], establishing the blueprint for modern deep learning.

The success of deep learning was pivotal for the field, particularly convolutional neural networks (CNN) [35]. The dramatic victory of AlexNet in the 2012 ImageNet competition demonstrated the superior capability of CNNs in image classification [31]. AlexNet led to rapid advancements in areas such as object detection, where the R-CNN framework [18] leveraged CNNs for feature extraction, crushing existing benchmarks.

Today, the representational power of deep learning pioneered by CNNs is the key enabler for sophisticated multi-modal systems. The field continues to evolve beyond its initial CNN-dominated era to include newer architectures like Vision Transformers [11] and generative models like Diffusion Models [24]. The powerful ability to learn features directly from raw data, a hallmark of deep learning, is what allows for the complex audio-visual integration discussed in the following sections.

2.4 Technical Foundations of Audio-Visual Mapping

An important concept to consider when working in computer music is "mapping". Mapping refers to the algorithm used to translate an input control source to parameters of a sound synthesis engine [14]. For instance, physics determines the mapping between the input gestures and resulting sound. The connection between the cause and effect in a violin is intuitive and culturally well understood, allowing ease of learning the violin, as well as ease of understanding as the audience. Electronically generated sounds separate control and sound, leading to difficulties for both the performer in controlling the instrument and for the audience in understanding the relationship between action and sound.

When considering explicitly mapping inputs to outputs of an instrument, three strategies can be employed: *one-to-one*, *one-to-many* or *many-to-one* [45].

- **One-to-one mapping**: One input gesture controls one musical parameter. This direct and simple connection between input and output is conceptually digestible, but lacks expression. The MIDI control architecture is notable example of this mapping.
- **One-to-many mapping**: One input gesture effects multiple musical parameters simultaneously. This method provides a macro-like expression control over the output. However, the approach limits fine access to the sound, so it must be combined with other control methods.
- **Many-to-one mapping**: Multiple input gestures are combined to control one musical parameter. This type of mapping is difficult to use, requiring experience with the system to achieve effective control. However, the approach enables significantly more expression than the others.

An effective instrument properly applies each of the above strategies to be easily understandable to a performer and the audience without sacrificing the expression, and thus the fun, of the instrument [27]. Therefore, it is critical that *VideoCsound* supports all mapping types to enable expressive generative music.

In the majority of visual-audio applications, the movements of the human body are taken optically to be mapped to a musical output. For instance, a computer vision model was to detect emotion from a image of face [1]. The program then plays music based on the detected emotion. Computer vision enables swift and intuitive visual communication of emotion. This is an example of *one-to-one mapping*, as the emotion is assigned to the type of song played. The system functions on a single input and output mapped *one-to-one*, resulting in a tool that is transparent to a user, but has little depth musically.

More intricate examples include gesture based systems [30], where gestures are to assigned to musical parameters. In this system, users teach the machine learning algorithm a gesture through a process of adding recordings of the gesture until the model can detect the gesture

accurately. The gesture can then be assigned to an action through and OSC (Open Sound Control) message. Although the paper's examples primarily employ *one-to-one* mapping, the framework's use of OSC makes it technically capable of *one-to-many* and *many-to-one* mappings. This highlights a common gap between technical possibility and artistic implementation; many systems possess the underlying flexibility for complex mapping but demonstrate only simple applications. This project seeks to explore that very potential for complex, semantically-driven mappings enabled by object detection.

These projects demonstrate that to enable effective mapping, the choice of audio system is critical. It must be able to receive input control data, process the values, then assign them to musical parameters. The user must be able to customize the processing and assignment of values to fine-tune the instrument's transparency and expressiveness. This need for deep customization and powerful sound generation motivates the choice of *Csound* for this project. As a mature and programmable synthesis environment, *Csound* provides the necessary granular control over both mapping strategies and sound design to build expressive audiovisual instruments.

2.5 Audio-Visual Applications in Analysis and Performance

The representational power of modern deep learning models has been the key enabler for sophisticated multi-modal systems that integrate visual perception with other sensory streams, such as audio. Rather than treating vision and audio as separate domains, these systems learn a joint embedding space, leveraging the complementary strengths of each modality to disambiguate information and achieve robustness that is impossible with unimodal approaches. This synergy is exemplified by applications spanning accessibility, human-computer interaction, and scene analysis.

One example is to use computer vision to ground and refine audio signals. For instance, Tavares et al. [50] demonstrate this by using a Kinect camera to track mallet positions on a vibraphone, effectively employing visual tracking as a prior to significantly reduce acoustic transcription errors like false positives and octave miscalculations. This approach highlights a key advantage of multi-modal systems: vision provides a spatial and structural context that pure audio processing lacks. This creates a hybrid analysis system where vision disambiguates the audio, significantly improving transcription robustness.

Beyond providing a spatial prior, computer vision can act as a direct sensory replacement or enhancement for accessibility. Microsoft's Seeing AI application [36] leverages this capability to narrate the visual world for visually impaired users, translating visual information (text, faces, products) into descriptive audio feedback. This transforms the smartphone camera from a simple recorder into an active visual interpreter, whose auditory output serves a clear functional purpose: vision-to-audio translation.

The most profound potential of audio-visual integration lies in self-supervised learning, where models learn cross-modal correlations directly from raw data. The work by Owens and Efros [38] is seminal in this area. By training a model to determine if audio and video streams are synchronized, the system learns to inherently associate visual events (e.g., a person's lip movements) with their corresponding sounds. This learned correlation enables advanced capabilities such as identifying sound sources, visualizing audio origins, and even isolating a single speaker's voice from a mixture based on lip-sync analysis. This demonstrates a technical, deep learning example of cross-modal correlations.

Beyond analytical and assistive applications, the relationship between vision and audio is also a rich domain for artistic performance. Audio-visual artists create musical performances that intertwine audio and visual elements without the use of computer vision. Robert Henke, in his performance series *Lumière*, uses lasers in tandem with a computer music performance [20]. In an interview with Ableton [21], Henke describes *Lumière's* technological structure. MIDI files store control signals for analog voltages for the laser brightness and movements. The same data is sent simultaneously to an audio engine to generate audio, resulting in inherently related audio and visual output for a coherent performance. However, Henke discusses the difficulties he faced in making the two mediums mesh in an interview with the Barbican [22]. The original implementation could not consistently produce satisfying outputs in both mediums at the same time, so the laser and audio were decoupled to only combine specific shapes and sounds that made meaningful sense. Henke's experience with *Lumière* highlights a key limitation of generating both audio and visual art from a single data source, thereby motivating the need for an approach where the visual element independently controls musical parameters.

Computer vision allows the visual art form to act as a control signal for the musical element of a performance. An example of this can be observed in tangible user interfaces (TUI). Audio D-touch [9] is a TUI system that allows users to control digital music software by physically manipulating custom-built blocks on a tabletop. A webcam observes the table surface, which detects special black and white symbols on the blocks and table. An algorithm detects the precise position and orientation of the blocks, which then affect the musical parameter the block type is assigned to. The paper presents three musical applications: a interactive musical stave, a tangible drum machine, and a physical sequencer. The use of computer vision enables fine-grained control over musical parameters, despite being an intuitive, cheap set up. The physical interface was found to be accessible by those without computer expertise and the visually impaired, and it fostered collaboration as multiple users could interact simultaneously. This demonstrates a successful implementation where the visual configuration of physical objects directly drives musical output. It overcomes the limitations of a single source by making the visual element the independent control mechanism.

The systems discussed thus far demonstrate the powerful synergy of audio and vision,

from analytical disambiguation (Tavares et al. [50]) and functional accessibility (Seeing AI [36]) to learned cross-modal correlations (Owens et al. [38]) and intuitive physical control (Audio D-touch [9]). Notably, D-touch establishes a successful paradigm for translating visual configuration into musical structure.

However, a limitation of vision systems like D-touch is their reliance on fiducial markers. This approach, while robust and precise, requires a prepared environment with custom physical objects. The semantic understanding of the scene is limited to the pre-defined meaning of these markers; the system knows a block is a "crotchet" or a "snare drum" because it was explicitly designed to do so, not because it understands the object itself.

This presents a clear opportunity: to move from marker-based to marker-less interaction, leveraging the capabilities of modern deep learning. By integrating state-of-the-art object detection models like *YOLO11*, a system can perceive a scene using high-level semantic information (e.g., identifying a 'cup', a 'book', or a 'person') without any specially prepared objects. This shift opens the door to using everyday items and environments as intuitive controllers for sound synthesis.

This project investigates this integration, specifically exploring the combination of the *YOLO11* object detection model with the programmable sound synthesis environment *Csound*. It addresses the technical challenge of building a low-latency pipeline between these frameworks and explores novel mapping strategies between detected object properties (class, position, size) and sonic parameters. The goal is to contribute a practical toolkit for audiovisual performance that leverages the world itself as a tangible interface, moving beyond predefined blocks with fiducial markers to a system of open-ended, semantically-aware musical control.

2.6 Conclusion

This review has traced the trajectory of computer vision from its classical, hand-crafted origins to the powerful deep learning models of today. It established the theoretical foundations of audio-visual mapping, emphasizing the importance of flexible strategies to achieve both transparency and expressiveness. By examining prior work in analysis and performance, a clear distinction emerged between systems that use vision to disambiguate audio, provide accessibility, or learn cross-modal correlations, and those designed for artistic expression. While tangible interfaces like Audio D-touch have successfully translated physical object configurations into musical control, they remain dependent on fiducial markers. This analysis reveals a significant opportunity to build upon this paradigm by replacing marker-based systems with modern, marker-less object detection. The integration of a semantically-aware model like *YOLO11* with a programmable synthesis environment like *Csound* promises a new form of musical interaction—one that uses the everyday world as

an open-ended, tangible interface.

3 Technology Stack

This chapter details the specific software libraries and frameworks chosen to build the *VideoCsound* system. The selection of each component was guided by the need for performance, accessibility, and suitability for real-time interactive applications. The following sections will justify the choice of Python as the integrating language, the *YOLO11* model via the *Ultralytics* library for computer vision, *OpenCV* for video handling, and *Csound* for powerful audio synthesis, explaining how these technologies combine to form a cohesive and effective processing pipeline.

3.1 System Integration: Python

The entire system is implemented in the Python programming language. This language was chosen as the foundation for three key reasons:

- **Ecosystem for Machine Learning**: Python is the de facto standard language for machine learning prototyping and computer vision applications. Its vast ecosystem of scientific computing libraries (e.g., NumPy, *OpenCV*) and deep learning frameworks (e.g., PyTorch, TensorFlow) provides unparalleled support for the project's core task of object detection.
- **Library Availability**: The specific, high-performance implementation of the *YOLO* model required for this project is available through the *Ultralytics* library, which offers a native Python API.
- **API Support**: Critically, the *Csound* audio synthesis environment provides a stable and well-documented Python API (CsoundAPI), allowing a Python script to control a running instance of *Csound* in real-time. Python serves as the ideal "glue" to seamlessly connect the vision and audio modules.

3.2 Computer Vision: YOLO11 via the Ultralytics Library

Given the choice of Python, the *Ultralytics* library was selected as the most efficient way to implement a state-of-the-art object detector [28]. It provides an intuitive interface to YOLO11 [29], the most modern version of the *YOLO* (You Only Look Once) architecture

[43] renowned for its excellent balance between speed and accuracy. For this project, the system utilizes a pre-trained *YOLO11n* model (*yolo11n.pt*) provided by *Ultralytics*. This model was trained on the Common Objects in Context (COCO) dataset, a large-scale dataset containing over 330,000 images annotated with 80 common object categories such as 'person', 'car', 'bird', and 'cup' [28]. This pre-trained model provides robust, general-purpose detection capabilities out-of-the-box, circumventing the significant overhead of curating a custom dataset and the computational expense of training a model.

The library abstracts away the complexities of model implementation, providing a simple methods for live video inference, object tracking, and returning structured results (bounding boxes, class IDs, confidence scores). This allowed for rapid development and a focus on processing the results for audio integration, rather than machine learning infrastructure.

3.3 Video I/O and Visualization: OpenCV

While the *Ultralytics* library provides the object detection model, the system handles video input and output using the *OpenCV* library (Open Source Computer Vision Library) [5]. *OpenCV* was chosen for its robust and widely-supported tools for reading video streams from files and live webcams, as well as for drawing visualization overlays like bounding boxes and labels onto the video frames.

The library's simple API, for example using *cv2.VideoCapture()* for input and *cv2.imshow()* for output, provided a straightforward method to manage the video pipeline. This allowed the development effort to focus on the integration between object tracking data and audio synthesis, rather than on low-level video processing.

3.4 Audio Synthesis: Csound and the Python API

The sound generation component is handled by *Csound* [3], a powerful language for audio synthesis with historical roots in computer music. This system leverages the *Csound* Python API [39], a wrapper for the *Csound* API written in C, to enable real-time audio synthesis and processing [7]. Specifically, the *ctcsound* binding, designed by François Pinot, facilitates direct integration with Python, allowing complete musical control.

While pure-Python synthesis libraries exist, *Csound* offers unparalleled low-level control over sound design through its vast array of opcodes. The Python API was the decisive factor, enabling a client-server model where the Python application sends control data to named software buses (channels) in a running *Csound* instance. The channels can directly control musical parameters, ensuring robust communication between the object tracking and *Csound*.

3.5 Conclusion

The technology stack for *VideoCsound* was strategically assembled to create a robust and modular pipeline. Python serves as the essential "glue," leveraging its extensive machine learning ecosystem to connect the vision and audio components seamlessly. The *Ultralytics* library provides a high-performance, accessible implementation of the state-of-the-art *YOLO11* object detector, while *OpenCV* offers reliable tools for managing video input and visualization. Finally, the *Csound* audio synthesis engine, controlled via its Python API, delivers the power and flexibility required for creating complex, responsive sonic environments. Together, these technologies form a complete system capable of capturing, analyzing, and sonifying visual data in real-time.

4 Implementation

This chapter details the software architecture and real-time processing pipeline of VideoCsound. The system's core functionality is built around a sequential data flow, where visual information from a video source is captured, analyzed for object data, and then mapped to control parameters within the Csound audio engine. Figure 4.1 provides a high-level illustration of this pipeline, from initial video input to final audiovisual output. The subsequent sections will provide a detailed breakdown of each stage in this process, beginning with the project's file structure and configuration.

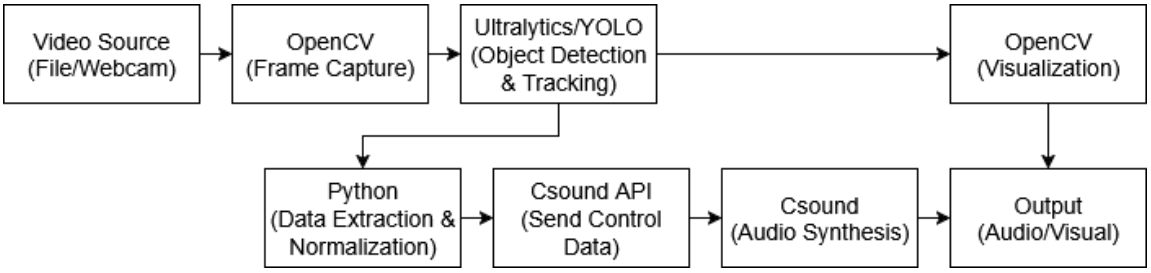

Figure 4.1: Data Flow Diagram of *VideoCsound*.

4.1 File Structure

The directory of VideoCsound is structured as follows:

```
YTCV2/
├── accessYaml.py
├── constants.py
├── csound.py
├── downloadYT.py
├── genYaml.py
├── processMP4.py
├── projectDir.py
└── src/
    └── config.yaml
```

```
└── csound.csd
```

The *VideoCsound* project is organized into a modular directory structure to separate core application logic from user-generated project files. The root directory contains the main application code and a source (*src/*) directory containing default configuration templates.

- **Python Modules:**
  - **processMP4.py**: The main application entry point and processing loop.
  - **csound.py**: A custom wrapper class for the *Csound* API.
  - **constants.py**: Defines global constants, such as the list of COCO class names.
  - **accessYaml.py and genYaml.py**: Utilities for reading from and generating the project YAML configuration files.
  - **downloadYT.py**: Handles YouTube URL validation and video downloading via the *yt-dlp* library.
  - **projectDir.py**: Manages project directory creation, validation, and template initialization.
- **Template Directory (*src/*)**:
  - **config.yaml**: The default configuration template for video, detection, and audio parameters.
  - **csound.csd**: This file contains the *Csound* orchestra, where instrument definitions are implemented to interface with the Python application by retrieving real-time control data, such as object type and position.
  - **objects.yaml**: The default mapping template for *YOLO* object classes to *Csound* instrument numbers.

These template files in the *src/* directory are not used directly at runtime. Instead, they are copied into a user-specified project directory during its initialization phase, serving as a starting point for user configuration.

The execution of the main script, *processMP4.py*, can be delineated into four primary phases:

- Project directory creation and validation, including the initialization of configuration files from templates
- Initialization of video input/output, the object tracking model, and the audio engine
- The main frame processing loop

- Program termination and resource cleanup

The subsequent sections detail the critical components of each phase.

4.2 Project Folder

A primary design challenge was facilitating the management of multiple, distinct audio-visual projects. Each project requires a unique set of configurations: a video source, a tailored *Csound* orchestra, and specific object-to-instrument mappings. To address this, a template-based project system was implemented.

A *VideoCsound* project is defined by a user-created directory. When a new directory is specified, the system initializes it by copying the three default configuration templates from the *src/* directory. This ensures every project starts with a valid and complete set of configuration files that the user can then modify. The core configuration files are:

- **config.yaml**: This file serves as the central configuration point, allowing users to define the video source (YouTube URL, local file path, or webcam), object detection parameters (confidence and IOU thresholds), video output resolution, and global audio settings (base frequency and amplitude).
- **csound.csd**: This file contains the *Csound* orchestra code. Users can define an arbitrary number of instruments, leveraging the full power of the *Csound* language to design sounds that respond to the data provided by the object tracker.
- **objects.yaml**: This file facilitates a high-level audiovisual mapping. It contains a list of all 80 object classes detectable by the pre-trained *YOLO11n* model. Users assign a list of *Csound* instrument numbers to each object class. Upon detection, an instrument is selected randomly from this list. A default entry provides a fallback instrument for unassigned classes.

This template-based configuration system allows users to easily create and switch between projects by specifying a different project directory, with each directory containing a self-contained set of settings and assets, thus enabling rapid experimentation with different audiovisual mappings.

4.3 Basic Usage

Running *VideoCsound* requires specifying a project directory. The program is executed from the command line using the following syntax:

```
python processMP4.py <directoryName>
```

Listing 4.1: Running the program in bash

The program first checks the provided directory for all necessary files using the *check_project_dir()* function.

```
# Check directory
if projectDir.check_project_dir(sys.argv[1]):
        sys.exit(1)  # If the directory was incomplete, exit
           the program
```

Listing 4.2: processMP4.py: 28-30

This function verifies the directory's existence and confirms it contains the three essential configuration files (*config.yaml*, *csound.csd*, *objects.yaml*). If any are missing, they are initialized from the *src/* templates, and the program terminates. This allows the user to review and customize the new project files before execution. If the check passes, the program proceeds to load the configuration and initialize the system components.

4.4 Initialization

The program requires initialization of four elements: video output, video input, object tracking, and audio.

4.4.1 Video Output

The program initializes video output using the resolution specified by the user in *config.yaml*. Since the input video source can vary widely in aspect ratio and quality, determining an appropriate output resolution presents a challenge.

The chosen solution is to adjust the output resolution from the input resolution only if both output width and height values are provided by the user. These values define the maximum screen space available to the program. The *findScale()* function within *processMP4.py* then calculates the largest possible output size that fits within this allocated space while preserving the original aspect ratio. This prevents visual distortion of the source video and the annotated bounding boxes when altering the size of the input source.

4.4.2 Video Input

The program accommodates three distinct video input sources: YouTube URLs, local video files, and a live webcam feed. Selection of the desired source type is managed within the

config.yaml file, which also necessitates the provision of the YouTube URL or video file path when those sources are chosen.

For YouTube video integration, the *YT-DLP* library is employed to facilitate both video download and metadata retrieval. Upon accessing a YouTube URL, its unique video ID is extracted. The system subsequently inspects the project directory for a video file corresponding to this ID. If no such file is located, the video is downloaded and saved using its unique ID as the filename. This entire process, encompassing ID verification and video download, is encapsulated within the *downloadYT.py* script. This script leverages the *YT-DLP* library through three dedicated methods: *check_id_of_url()*, *check_local_file()*, and *download_video()*.

Once a YouTube video is downloaded, it can be treated identically to local video files. The *OpenCV* library provides easy access to video files by passing the file path to its *VideoCapture()* method:

```
1 cap = cv2.VideoCapture(str(video_file))
```

Listing 4.3: *processMP4.py*: 53

Similarly, the *VideoCapture()* method can be used to access the computer's default webcam by passing in 0:

```
1 cap = cv2.VideoCapture(0) # Use the default webcam
```

Listing 4.4: *processMP4.py*: 62

With these steps, the video input is properly initialized, and *OpenCV* efficiently handles the video processing.

4.4.3 Object Tracking

The *Ultralytics* library's *YOLO* class handles all underlying model loading and configuration, allowing for initialization with a single line of code.

```
1 # Load the model
2 yolo = YOLO('yolo11n.pt')
```

Listing 4.5: *processMP4.py*: 34-35

If the "*yolo11n.pt*" file is not present locally, the *YOLO* class will automatically download it. Once the .pt file is located, it is loaded and prepared for use within the processing loop.

4.4.4 Csound

To streamline the initialization of *Csound* and simplify its API, a wrapper class for the *Csound* API was created in *csound.py*.

The *Csound* class is initialized by passing the program directory path, which grants access to the *csound.csd* file within that directory. The *initialize()* method then handles the compilation and creation of a *Csound* performance thread, and the *start()* method begins the performance. This sequence of initializing the class and executing these two methods provides a streamlined approach to *Csound* initialization:

```
# Initialize Csound
cs = csound(directory)
res = cs.initialize()
if res == 1:
        print("Csound initialization failed.")
        sys.exit(1)
cs.start()
```

Listing 4.6: *processMP4.py*: 132-138

If the *csound.csd* file fails to compile, the *initialize()* method returns an error code (1), causing the program to print an error message and exit.

4.5 Frame Processing

Once program initialization is complete, the script enters the main processing loop. For each frame captured from the video source, the system performs a sequence of steps: detecting and tracking objects, passing the extracted data to *Csound*, triggering and managing *Csound* instruments, and finally visualizing and displaying the results. This loop continues in real-time until the video ends or the user terminates the program.

The core of the processing begins with object detection and tracking. For each video frame, the *YOLO* model's track() method is invoked:

```
result = yolo.track(frame,
        persist=True,
        verbose=False,
        conf=config_data.get('YOLO_CONF_THRESHOLD', 0.3),
        iou=config_data.get('YOLO_IOU_THRESHOLD', 0.5),
        max_det=config_data.get('MAX_DETECTIONS', 3))[0]
```

Listing 4.7: *processMP4.py*: 152-157

This method returns a results object containing data for all detected objects in the frame, including bounding box coordinates (center x, y, width, height), a unique track ID for maintaining identity across frames, and a class ID. The *persist=True* parameter is crucial, as it enables the model to maintain these unique IDs for objects between frames, which is essential for continuous sonic output.

The data for each detected object is subsequently processed for communication with *Csound*. Bounding box coordinates are normalized to ratios of the screen dimensions using helper functions *x_ratio()* and *y_ratio()*, which ensures that control values are consistent regardless of video resolution.

Notably, the result of *y_ratio* is inverted before being passed to *Csound*. This is necessary because the *YOLO11* model outputs coordinates where the origin is at the top-left, meaning larger y-values are lower on the screen. The inversion remaps this to a more conventional system where a higher y-value corresponds to a higher screen position, a common paradigm for musical mapping.

The normalized values are sent to *Csound* via named control channels. The unique track ID is embedded in each channel name (e.g., "x_123", "y_123") to provide each tracked object with its own set of independent control signals.

```
# Iterate through the boxes and track IDs
for box, track_id, class_id in zip(boxes, track_ids, class_ids):
        # ... code omitted for brevity

        # Get the coordinates of the bounding box
        x, y, w, h = box

        #...

        # send x, y ratios to Csound
        cs.set_control_channel(f"x{track_id}", x_ratio(x))
        cs.set_control_channel(f"y{track_id}", 1 - y_ratio(y))
            # Invert y-axis
        cs.set_control_channel(f"w{track_id}", w /
           original_width)
        cs.set_control_channel(f"h{track_id}", h /
           original_height)
```

Listing 4.8: *processMP4.py*: 169-170, 177-178, 182-186

Upon an object's initial detection, a *Csound* instrument must be triggered. The object's

class ID is used to look up a list of assigned instrument numbers in the *objects.yaml* configuration file. An instrument is selected randomly from this list, promoting variety in the sonic output.

```
instrNum = random.choice(yaml.access_data(constants.CLASSES[
    class_id]))
# Save chosen instrument number of track ID to turn off later
active_ids[track_id] = (instrNum)
# Formats a Csound score statement: i<instrNum>.<trackID>
    start duration p4 p5 ...
cs.event_string(f"i {instrNum}.{track_id} 0 -1
        {track_id}
        {config_data.get('BASE_FREQ', 440)}
        {config_data.get('AMP', 0.5)/config_data.get('
            MAX_DETECTIONS', 5)}
        {x} {y} {w} {h}")
```

Listing 4.9: *processMP4.py*: 198-200

A critical design decision is to use the track ID as part of the instrument number in the event string ("*i*{*instrNum*}.{*track_id*}"). For instance, when an object is no longer detected, its corresponding sound must be terminated. The program checks for tracked objects that have disappeared and sends a *Csound* event to turn off the specific instrument instance associated with that object's track ID.

```
# Remove the track from active_ids
instrNum = active_ids.pop(track_id)
cs.event_string(f"i -{instrNum}.{track_id} 0 0")
```

Listing 4.10: *processMP4.py*: 207-209

Finally, the visualization is updated. The annotated frame with bounding boxes and labels is displayed, and the loop checks for user input to exit.

```
# Visualize the result on the frame
frame = result.plot()

# ... omitted code

# Resize frame for better display
if needResizing:
frame = cv2.resize(frame, (width, height))

```

```
# Show the image
cv2.imshow('frame', frame)
```

Listing 4.11: *processMP4.py*: 166-167, 225-230

The loop breaks if the 'q' or 'escape' key is pressed, providing a responsive way for the user to end the performance.

```
# Break the loop if 'q' is pressed
key = cv2.waitKey(1) & 0xFF
if key == ord('q') or chr(key) == '\x1b':  # 'q' or ESC key
        break
```

Listing 4.12: *processMP4.py*: 232-235

4.6 Exiting the Program

Exiting the program includes only 2 simple tasks: close all CV2 windows and close the Csound thread.

```
# release the video capture and destroy all windows
cv2.destroyAllWindows()
# Close the Csound thread
cs.close_thread()
```

Listing 4.13: *processMP4.py*: 238-241

4.7 Conclusion

This chapter has detailed the complete implementation of the *VideoCsound* system, from its modular file structure to its real-time frame processing loop. The architecture is designed for both functionality and usability, with a template-based project system that allows users to easily manage different configurations. Key design decisions, such as the use of an object's unique track ID to manage its corresponding *Csound* instrument instance, ensure that sounds are triggered and terminated reliably. The data pipeline (capturing a frame, tracking objects, normalizing coordinates, and sending control data to *Csound*) forms the core of the application, successfully translating visual events into synchronized audio. This implementation provides a solid foundation for the practical explorations and evaluations presented in the following chapter.

5 Demonstrations

This chapter presents a qualitative evaluation of the *VideoCsound* system through a series of four demonstration videos, supported by empirical observations on system performance. Each demonstration is designed to assess the system's performance under different conditions and with diverse visual inputs, thereby illustrating its capabilities and exposing its limitations. The selected scenarios range from analyzing pre-recorded footage of dynamic urban environments and natural settings to a live, performative application using a webcam.

The primary objectives of these demonstrations are to:

- **Demo 1:** Temple Bar Webcam Footage tests the system with high-density, semi-predictable pedestrian movement.
- **Demo 2:** Bird Video evaluates performance with rapid, erratic motion and challenges the limits of the pre-trained model's accuracy.
- **Demo 3:** Mulberry Street Webcam Footage focuses on tracking a vehicle and examines issues arising from camera perspective and confidence thresholds.
- **Demo 4:** Webcam Performance shifts the focus to intentional, human-in-the-loop interaction, demonstrating the system's use as a novel musical interface.

Each subsection will describe the video source, introduce the audio mapping, present observations on the system's behavior, and provide a critical analysis of the results.

5.1 Temple Bar Webcam Footage

Demo Video

5.1.1 Source Description

This demonstration utilized a live webcam feed of Temple Bar, a bustling cultural quarter in Dublin, Ireland, sourced from *EarthCam* [12]. The video captures dense, continuous pedestrian traffic moving through a narrow street, providing a stream of numerous and predictable visual objects for tracking. The source video's resolution of 1920x1080 pixels

presented a significant initial test case for the system's processing capabilities.

5.1.2 Audiovisual Mapping Configuration

The demonstration employed a minimal configuration to evaluate the system's core functionality with a complex visual input. Within the project's *objects.yaml* file, only the default instrument entry was defined, assigned to instrument number 2. All specific object class mappings were left undefined. Consequently, every object detected by the *YOLO11* model, regardless of its class (e.g., 'person', 'umbrella'), triggered the same *Csound* instrument.

Instrument 2 was designed as a frequency modulation (FM) synthesizer using the foscil opcode. The parameters of the bounding box were mapped to various aspects of the sound to create a dynamic and spatially-aware output. The *foscil* opcode generates an audio signal through frequency modulation, following the form *[foscil kamp, kcps, kcar, kmod, kndx [, iphs]]*.

The relevant *Csound* code is as follows:

```
aFM foscil kAmp * kEnv, kFreq * (1 + 10 * kh), kCarrier, ky *
    20, pow(2, kw * 10)
aFilt = butterlp:a(aFM, 1000 + (kw * 1000))
aOutL, aOutR pan2 aFilt, kx
out aOutL, aOutR
```

Listing 5.1: *Examples/templeBar/csound.csd*: 66-69. FM synthesis instrument with mappings for bounding box height (*kh*), Y-position (*ky*), width (*kw*), and X-position (*kx*).

- **Base Frequency**: The height of the bounding box (*kh*), normalized to a ratio of the screen height, was mapped to the base frequency. It acts as a multiplier on the base frequency (kFreq), creating a range of 1 to 11 times the base value. This value then proportionally affects the carrier and modulator frequency. This meant taller objects (larger kh) produced higher-pitched sounds.
- **Modulator Frequency**: The modulator frequency is determined by multiplying the base frequency by the *kmod* parameter of the opcode. This parameter was driven by the vertical position (*ky*) of the bounding box's center point, scaled by exponentially by *ky*. Therefore, the modulator frequency was exponentially proportional to the object's vertical position in the frame, with objects higher on the screen producing higher modulator frequencies and thus more complex harmonics.
- **Modulation Index**: The modulation index (*kndx*), which controls the intensity of the modulation and thus the harmonic complexity of the timbre, was controlled by the

width of the bounding box (*kw*) via an exponential function *(pow(2, kw*10))*. Wider objects (larger *kw*) produced a higher index, leading to brighter, more dissonant, and more complex spectra.

- **Filtering**: The same width parameter (*kw*) was also used to control the cutoff frequency of a low-pass filter (*butterlp*), adding a timbral dimension to the object's size.
- **Spatialization**: The horizontal position (*kx*) of the bounding box center was directly mapped to the stereo panning position, placing the sound image accordingly within the stereo field.

5.1.3 Observations and System Performance

The system successfully demonstrated its core functionality: objects were detected, tracked with unique IDs, and triggered corresponding audio events in *Csound* in real-time. The visual feedback, comprising bounding boxes and tracking trails, provided an intuitive representation of the model's accuracy and persistence.

However, two primary issues were observed. First, a substantial performance degradation was evident. The program operated at less than half the intended speed of the source video, resulting in a noticeable and a significant and disruptive lag. Second, the sonic output during periods of high object density became overly dense, merging into an undifferentiated mass of sound that obscured the auditory origin of individual events. This was particularly pronounced when multiple stationary objects were detected, generating sustained, unchanging tones from the FM synthesizer.

5.1.4 Analysis of Technical and Sonic Outcomes

The observed latency is directly attributable to the computational overhead of video preprocessing. The *YOLO11* model is optimized for a 640x640 pixel input [28], but the source video exceeded this specification by a factor of over five. Consequently, each 1920x1080 frame required downscaling by the Ultralytics library before object detection could occur. The cumulative cost of this per-frame rescaling, combined with the detection and audio routing logic, overwhelmed the processing pipeline, preventing it from maintaining the original frame rate.

The issue of auditory clutter stems from the mapping strategy and instrument design. While the position parameter (e.g., the y_ratio of a bounding box) effectively modulated audio for moving objects—creating a perceptible glissando effect for pedestrians ascending through the frame, it provided no variation for stationary objects. The FM synthesis instrument, when triggered by a static bounding box, produced a steady-state tone. With a

high number of concurrent detections, these tones amalgamated into a monolithic drone, lacking the articulation and dynamism necessary to reflect the visual scene's complexity meaningfully.

5.1.5 Conclusion and Implications

The Temple Bar demonstration validated the fundamental integration of the *VideoCsound* pipeline but exposed critical limitations in its handling of high-resolution input and its sonic response to static objects. The performance issues related to framerate and latency underscore the necessity for a more optimized video processing workflow, a core technical challenge synthesized in Section 6.2. Furthermore, the results highlight the importance of designing *Csound* instruments that generate engaging audio even in the absence of change in an object's positional coordinates, suggesting a need for more sophisticated parameter mapping that incorporates other data such as object speed, detection confidence.

5.2 Bird Video

Demo Video

5.2.1 Source Description

This demonstration evaluated the system's performance with non-human subjects and rapid, unpredictable motion. The source video, created by Paul Dinning, is representative of a genre of online content intended for feline viewership, often colloquially termed 'CatTV' [10]. The video features a static camera capturing a natural scene centered on a bird feeder, which attracts various species. The video used depicts various bird species and squirrels interacting with the feeder, characterized by quick, darting movements and brief periods of stasis. This source provided an excellent test case for tracking small, fast-moving objects with high variation in appearance.

5.2.2 Audiovisual Mapping Configuration

The configuration for this demonstration was identical to that used in the Temple Bar demo (Section 5.1.2). The same FM synthesis instrument, detailed in Listing 5.1, was triggered for all detected objects. This consistency allowed for a direct comparison of the system's audio output and tracking performance between a scenario with large, slow-moving objects (people) and one with small, fast-moving objects (birds).

5.2.3 Observations and System Performance

The system's behavior diverged significantly from the Temple Bar demonstration. The audio output was immediately more dynamic and transient, characterized by rapid glissandi and sudden timbral shifts that closely followed the erratic flight paths of the birds, as demonstrated at 0:08 in the corresponding demonstration video. This stood in stark contrast to the monolithic drone produced by the stationary crowd in the previous demo, effectively demonstrating the system's capacity for responsive sonic output.

However, these sonic successes were frequently undermined by substantial inaccuracies in object detection, stemming from a domain mismatch. The *YOLO11n* model was pre-trained on the general-purpose *COCO* dataset, which may lack sufficient examples of the specific bird species, orientations, and atypical camera angles present in the source video. For instance, the model failed to detect a prominent bird at the 0:32 mark, resulting in a false negative. At 1:10, a bird facing the camera was sporadically detected and frequently misclassified as a 'sheep' or 'cow' (a false positive). These inaccuracies created a noticeable discrepancy between the visual events and the resulting audio, undermining the perceived synchrony of the audiovisual mapping.

A consistent audio latency of approximately 500ms (15 frames at 30fps) was also measured between the initial visual detection of a bird and the onset of the corresponding sound. This delay was particularly exposed in this video due to birds often entering the frame individually.

5.2.4 Analysis of Technical and Sonic Outcomes

The improved sonic dynamism directly results from the mapping strategy described in Section 5.1.2. The rapid changes in the bounding box parameters (x, y, w, h) caused by the birds' movement led to proportional and rapid changes in the FM synthesis parameters, generating the observed energetic and complex sounds.

The tracking inaccuracies are a known limitation of general-purpose object detection models when applied to specialized domains [18]. A pre-trained model learns universal features (e.g., edges, textures) from a large dataset like COCO. This model is typically then trained on a smaller, specific dataset to adapt the general features to a specific task, a process known as fine-tuning. However, this demonstration utilized the pre-trained model without such domain-specific adaptation. Consequently, its failures highlight a predictable reliance on patterns from its training data, which lacked sufficient examples of the video's specific bird species and orientations. The problem of domain mismatch was exacerbated by the erratic movements of the birds, which presented perspectives of the subjects likely underrepresented in the model's training set.

The cause of the consistent audio latency requires further investigation but suggests a

bottleneck in the procesing pipeline between frame capture (*OpenCV*), model inference (*Ultralytics*), and event scheduling (*Csound*). Potential causes include buffering in the video capture loop, computational overhead in the Python-*Csound* communication layer, or a configuration issue within the *Csound* API itself.

5.2.5 Conclusion and Implications

This demonstration confirmed the system's capacity to generate compelling and responsive audio from rapid visual motion. However, it also revealed critical dependencies on the capabilities of the underlying pre-trained model. The performance is inherently contingent upon the model's accuracy; the system's output is only as semantically meaningful as its detections.

The results strongly suggest that for specialized applications (e.g., bird watching, industrial part inspection), the system's performance would be greatly enhanced by using a domain-specific model fine-tuned on relevant data. For a general-purpose tool, implementing a robust class-filtering system would allow users to ignore classes prone to error in a given context, thereby improving the reliability of the audio output. The persistent latency issue underscores the need for further profiling and optimization of the real-time pipeline, a point addressed in the general technical improvements outlined in Section 6.2.

5.3 Mulberry Street Webcam Footage

Demo Video

5.3.1 Source Description

To evaluate the system's performance tracking vehicles, this demonstration utilized footage of Mulberry Street in New York City, sourced from *EarthCam* [13]. The primary test subject was a slow-moving van, which provided a clear and consistent target for the object detection model. The video also featured other urban elements, such as a fire hydrant and a decorated street background, which presented potential challenges for the detector.

5.3.2 Audiovisual Mapping Configuration

The mapping strategy for this demonstration was designed with two objectives: to create a transparent, easily understandable sonification for the primary target vehicle, and to generate a more complex, internally dynamic sound for incidental or erroneously detected objects. Two new instruments were implemented to achieve this.

Instrument 1, a wavetable synthesizer, was assigned specifically to the car class in *objects.yaml*. It was designed for clarity, with simple and direct parameter mappings using the *poscil* opcode.

- **Frequency and Timbre**: The vertical position of the bounding box (ky) was mapped to both the fundamental frequency and the wavetable index. This created a direct relationship where ascending objects produced a higher pitch while also cross-fading the timbre from a square wave to a saw wave.
- **Spatialization**: Consistent with previous demonstrations, the horizontal position (kx) was mapped directly to the stereo panning position.

```
aSaw poscil kAmp * kEnv, kFreq * (1+ky*4), 24
aSquare poscil kAmp * kEnv, kFreq * (1+ky*4), 23

aOut = aSaw * ky + aSquare * (1-ky)

aOutL, aOutR pan2 aOut, kx
out aOutL, aOutR
```

Listing 5.2: *Examples/mulberrySt/csound.csd*: 35-41

Instrument 3 served as the default for all other detections, sonifying objects like the misidentified fire hydrant. It was implemented as a sample-and-hold based amplitude modulation (AM) synthesizer, adapted from an instrument in Dr. Richard Boulanger's *Csound* piece, "Trapped" [4]. The instrument's key feature is its ability to generate complex, erratic patterns independent of the object's movement, using *randh* and *expsegr* opcodes to create unpredictable modulation.

- **Base Frequency**: The vertical position (ky) was mapped to the synthesizer's base frequency.
- **Spatialization**: The horizontal position (kx) controlled the stereo panning.

```
randAmp1:k  = expsegr:k(1, 3, 80, 2, 1)
randFreq1:k = expsegr:k(2, 3, 20, 2, 2)
amp1:k init 0
amp1 = linsegr:k(i(amp1), 2, p6, 3, 0)
rand1:k = randh:k(randAmp1, randFreq1, 10) ; randomize with
   time seed
a1 = oscil:a(amp1, (iFreq  * (1.01+ky)) + rand1, 24, .3)

aOutL, aOutR pan2 a1, kx
```

Listing 5.3: *Examples/mulberrySt/csound.csd*: 57-64

5.3.3 Observations and System Performance

The system successfully tracked the van for approximately eight seconds as it moved through the frame. However, as the van approached the bottom of the screen, the model's performance degraded, resulting in dropped detections and a misidentification of the object as a 'truck'.

Separately, the system incorrectly identified a stationary fire hydrant as a 'person'. Despite the object's lack of movement, the resulting audio was erratic and engaging. This sonic output stood in contrast to the monotonous drones observed in the Temple Bar demonstration (Section 5.1), which also featured stationary objects.

5.3.4 Analysis of Technical and Sonic Outcomes

The inconsistent detection of the van is likely attributable to two factors. The first is the sharp camera angle, which, as observed in the previous demonstration (Section 5.2), presents subjects from a perspective likely underrepresented in the model's training set. The second factor is a high confidence threshold, a parameter that acts as a gatekeeper by ignoring any detection the model deems uncertain. This created a classic precision-recall trade-off: while the high threshold successfully prevented false positives by filtering out misidentified background decorations, it simultaneously reduced recall by rejecting legitimate, but less confident, detections of the van as its appearance changed.

The successful generation of dynamic audio from the stationary fire hydrant demonstrates the importance of instrument design. Unlike the FM synthesizer in the Temple Bar demo (Section 5.1), whose sonic character was directly tied to static positional data, the sample-and-hold based AM synthesizer possesses its own internal modulation. This allows it to create auditory interest over time, independent of changes in the object's coordinates, thus effectively solving the static drone problem

5.3.5 Conclusion and Implications

This demonstration highlights a critical trade-off between preventing false positives and maintaining consistent tracking. The results suggest that a global confidence threshold is insufficient for scenes with diverse and complex content. To mitigate this issue, the ability to filter detections by specific object classes would be highly beneficial. For instance, limiting detection to only the 'car' and 'truck' classes would have allowed for a lower, more forgiving confidence threshold, likely resulting in cleaner and more consistent tracking of the primary subject.

5.4 Webcam Performance

Demo Video

5.4.1 Source Description

For the final demonstration, the system's video source was switched from pre-recorded footage to a live webcam feed. This shifted the application's focus from a passive analysis of existing scenes to an active, performative context. By assigning *Csound* instruments to small, common household items, such as a cup, a water bottle, scissors, and a TV remote, the system was transformed into a novel musical interface. This setup allows a performer to manipulate everyday objects within the webcam's view to "play" the corresponding *Csound* instruments in real-time.

5.4.2 Audiovisual Mapping Configuration

The audiovisual mapping for this performance was designed to create a multi-layered, interactive ensemble using four distinct instruments assigned to different household objects.

1. **FM Synthesizer (Instrument 2)**: Assigned to the scissors class, this instrument utilized the same FM synthesis patch from the first demonstration (see Listing 5.1). The intention was to leverage a familiar, complex sound source in a new context of fluid, real-time performative control.
2. **AM Synthesizer (Instrument 5)**: Assigned to the remote class, this instrument used the sample-and-hold based AM synthesizer from the Mulberry Street demo (see Listing 5.3). Its internally dynamic nature was chosen to provide rhythmic complexity.
3. **Noise Instrument (Instrument 3)**: Assigned to the cup and bottle classes, this instrument was designed to create an atmospheric base layer. It processes white noise through two filters modulated by the object's properties.
    - **Low-Pass Filter**: The vertical position (ky) controlled the cutoff frequency of a low-pass filter, making the sound brighter as the object was raised.
    - **High-Pass Filter**: The width of the bounding box (kw) controlled the cutoff frequency of a high-pass filter, thinning the sound as the object appeared wider to the camera.
    - **Spatialization**: The horizontal position (kx) controlled the stereo panning.

```
aNoise noise kEnv*iAmp, 0
aLP butterlp aNoise, 100 + (ky * 10000)
```

```
aHP butterhp aLP, 10000 - (kw * 10000)

aOutL, aOutR pan2 aHP, kx
out aOutL, aOutR
```

Listing 5.4: *Examples/webcam/csound.csd*: 89-94

4. **Plucked String Model (Instrument 6)**: Assigned to the person class, this instrument was intended to sonify the performer's hands when they inadvertently entered the frame. It was designed as a physical model of a plucked string to produce a melodic but ephemeral sound that would not obscure the other instruments.
    - **Frequency**: The vertical position (ky) controlled the pitch.
    - **Timbre**: The height of the bounding box (kh) was mapped to a low-pass filter, affecting the brightness of the pluck.
    - **Spatialization**: The horizontal position (kx) controlled the stereo panning.

```
aPluck pluck iAmp * kEnv, kFreq * (1+ky+4), i(kFreq), 0, 1

aLP = butterlp:a(aPluck, 100 + (kh * 10000))

aOutL, aOutR pan2 aLP, kx
out aOutL, aOutR
```

Listing 5.5: *Examples/webcam/csound.csd*: 176-181

5.4.3 Observations and System Performance

The system successfully tracked the various objects, and the resulting performance demonstrated a high degree of interactivity. Moving the scissors freely in front of the camera generated wild and expressive timbres from the assigned FM synthesizer. The person class, which was triggered by the performer's hand entering the frame, produced a melodic yet ephemeral sound that integrated well without obscuring the other instruments.

A key performative mechanic observed was occlusion; when one object covered another, the occluded object was no longer detected, and its sound ceased immediately (as shown at 1:37 in the corresponding demonstration video). This provided a simple and intuitive method for stopping an instrument by physically obstructing it from the camera's view. However, the object detection model struggled with consistent tracking of the TV remote, suggesting that its visual features were less distinct to the model. Furthermore, placing objects on the floor resulted in an unfavorable camera angle that likely contributed to occasional misidentifications.

5.4.4 Analysis of Technical and Sonic Outcomes

The success of the performance hinged on the direct, real-time feedback loop between physical action and sonic response. The varied and expressive audio resulted from the performer's ability to manipulate all of an object's bounding box parameters (position, width, and height) in a fluid, three-dimensional space, which was not possible in the previous 2D video demonstrations.

The tracking inconsistencies with the TV remote and the issues caused by the sharp camera angle reinforce findings from earlier demos. These problems highlight the model's sensitivity to object appearance and perspective, which are inherent limitations of a general-purpose model. The observation that occlusion can be used to control sound introduces a tangible, performative affordance; the physical act of hiding and revealing objects becomes a method for structuring the musical performance.

5.4.5 Conclusion and Implications

This demonstration successfully reframes *VideoCsound* as an innovative instrument for musical expression, highlighting its potential for live, interactive applications. The results suggest that for optimal performance, the physical environment should be considered. A more controlled setup, such as a top-down view of a desktop with bright, even lighting and a high-contrast background, would likely improve the accuracy and consistency of the object detection, leading to more reliable musical control. The use of occlusion as a performance technique also opens a new avenue for intentional musical interaction with the system. For instance, a performer could use a physical barrier to strategically hide and reveal objects, adding a theatrical layer of compositional control.

5.5 Conclusion

The four demonstrations collectively provided a comprehensive qualitative evaluation of the *VideoCsound* system, successfully highlighting its strengths while rigorously exposing its weaknesses. The Temple Bar test confirmed the system's core functionality but revealed critical performance bottlenecks with high-resolution video and sonic monotony with static objects. The bird video demonstration showcased the system's capacity for dynamic, responsive audio from rapid motion, but also underscored the limitations and inaccuracies of the pre-trained model when faced with a domain mismatch. The Mulberry Street footage illustrated the trade-offs associated with confidence thresholds and demonstrated the importance of instrument design in generating interest from stationary objects. Finally, the live webcam performance successfully reframed the system as an expressive musical instrument, revealing its potential for real-time interaction and uncovering emergent performative techniques like occlusion. Together, these findings validate *VideoCsound* as a promising prototype while clearly defining the key challenges—namely latency, model accuracy, and parameter mapping—that must be addressed in future work.

6 Conclusion and Future Work

6.1 Summary of Contributions

This thesis presented *VideoCsound*, a novel, open-source performance system that merges computer vision with algorithmic composition to transform real-time visual input into synchronized, generative audio.

The core contribution is the design, implementation, and evaluation of this system. After establishing the project's context and identifying a gap in existing marker-less musical interfaces in the Chapter 2 literature review, the research detailed the system's software architecture in Chapter 3 and Chapter 4. This architecture integrates the *YOLO11* object detection model with the *Csound* audio synthesis environment via a custom Python pipeline. While previous multi-modal music interfaces relied on fiducial markers, *VideoCsound* leverages modern computer vision technology for a marker-less approach, contributing a practical toolkit that uses the world itself as a tangible interface. Through a series of diverse demonstrations in Chapter 5 (analyzing urban crowds, unpredictable wildlife, vehicle traffic, and a live webcam performance), the thesis validated the system's capabilities while rigorously identifying its technical challenges and performance limitations. The system provides a framework for future development in both artistic and practical applications through computer vision.

Ultimately, *VideoCsound* serves as a successful prototype that demonstrates the artistic and functional potential of mapping high-level semantic information from video to complex sonic parameters, laying the groundwork for future research in data-driven art and interactive musical systems.

6.2 Limitations and Key Challenges

The demonstrations successfully validated the core functionality of *VideoCsound* but also exposed several critical limitations that define the primary challenges for future development. These issues fall into two main categories: real-time performance and the efficiency of the video processing pipeline.

6.2.1 Real-Time Performance Challenges

A primary challenge identified across the demonstrations was maintaining system performance, particularly concerning framerate and audio latency.

- **Framerate Degradation**: High-resolution source videos, such as the 1920x1080 Temple Bar feed, led to a low processing framerate, resulting in a jarring and unpleasant visual experience. Optimizing the processing pipeline to ensure fluid visual feedback, especially under heavy load, is crucial for improving user experience.
- **Audio Latency**: The most critical performance issue was a consistent delay between a visual event and its corresponding audio response. This lag, measured to be approximately half a second (15 frames at 30fps) in the bird video demonstration, significantly detracts from the feeling of direct, real-time interactivity. While the cause requires further investigation, this delay undermines the system's effectiveness as a responsive instrument.

One potential solution to the latency issue is a "rendered approach," where object detection data is collected offline first, and the final audio and video are combined in a post-processing step. This method, used in a previous project, offers several advantages:

- **Improved Video Quality**: Processing time is no longer a limiting factor, allowing for higher-quality video without framerate drops.
- **Eliminated Latency**: The approach bypasses real-time communication issues between Python and *Csound* that contribute to lag.
- **Automatic Output File**: It inherently creates a final, synchronized video file, a feature not currently possible with the live system.

Despite these benefits, the live processing approach was maintained for this prototype primarily to prioritize a quick, preview-like user experience for sound design experimentation, especially with long source videos, and to retain the crucial functionality of the live webcam performance, which relies on real-time processing.

6.2.2 Video Processing Inefficiencies

The current video input method, which relies on *OpenCV*, is not optimal for certain functionalities. Shifting the video handling to the *Ultralytics* library's native methods would enable more advanced and efficient processing techniques. This would facilitate batch processing of frames, allow for the use of "video stride" (processing only every Nth frame to reduce computational load), and support the creation of a more continuous and efficient video stream. Adopting these methods would be a key step in addressing the framerate and latency issues described above.

6.3 Directions for Future Work

Based on the findings from the demonstrations, several key directions for future work have been identified. These can be grouped into three main areas: improving the user interface for accessibility, expanding the system's expressive capabilities through enhanced parameter mapping, and exploring a wider range of applications.

6.3.1 User Interface Development

A primary limitation of the current *VideoCsound* prototype is its reliance on a command-line interface (CLI), which is often intimidating for non-technical users and requires a complex installation process. While a web application was initially considered for its accessibility, the technical hurdles related to client-server architecture, real-time video processing, and high server costs make it an impractical goal for the project.

The most logical and impactful next step is the development of a desktop application with a graphical user interface (GUI). Using a Python GUI toolkit like *Tkinter* or *PySide* would make the system more intuitive and accessible to a broader audience, including artists and musicians without programming expertise. To further enhance the user experience, the project could be consolidated into a single executable file using a packager. This would eliminate the need for users to install Python or any dependencies, making the application trivial to share and run.

6.3.2 Enhanced Parameter Mapping

The demonstrations revealed that the system's musicality is directly tied to the richness of the data mapped to *Csound*. Future work should focus on moving beyond bounding box coordinates to include more nuanced parameters and create a more musically expressive system. The following conceptual enhancements have been identified:

- **Speed and Motion Analysis**: Integrating the velocity and acceleration of objects would allow for a direct mapping between an object's physical energy and the resulting sound's intensity or complexity. This would add a significant dimension of expressiveness, enabling rapid movements to elicit more energetic sonic responses while slower or stationary objects could trigger more ambient textures.
- **Confidence Value Mapping**: Transmitting the model's detection confidence score for each object would allow *Csound* instruments to adapt dynamically to the system's certainty. Using this score as a control signal could introduce sonic variations that reflect the system's certainty, adding a layer of organic instability to the output. This could be mapped to parameters like brightness or timbral stability, adding a layer of nuance to the sonification.

- **Dynamic Voice Management**: Sending the total number of currently detected objects to *Csound* would enable more intelligent resource and audio management. Implementing this would allow for intelligent audio mixing, such as making a lone object sound fuller or reducing the complexity of individual voices in a crowded scene to prevent auditory clutter.

6.3.3 Expansion of Applications

Beyond its use as a tool for experimental music and live performance, the *VideoCsound* framework has potential applications across a wide spectrum of fields. Future development could explore these avenues:

- **Educational Tools**: The system can serve as an accessible and interactive tool for demonstrating the core principles of AI and computer vision, providing immediate auditory feedback that makes the technology's capabilities and limitations tangible to users.
- **Practical and Real-World Scenarios**: The system's core functionality is particularly well-suited for scenarios where auditory feedback is more effective than constant visual monitoring. Future research could explore the system's potential in the following areas:
  - **Kinesthetic Feedback for Training**: The system could provide real-time audio cues for physical training that requires precise form, such as a golfer's swing or a physical therapy exercise. A successful movement could be rewarded with a pleasing sound, allowing the user to focus on their body instead of a screen.
  - **Enhanced Security Monitoring**: In security applications, *VideoCsound* could translate visual alerts into spatially-aware audio cues. A guard could be alerted to a person approaching a restricted area by a distinct sound panned to the direction of the intrusion, enabling faster response times without constant screen monitoring.
  - **Industrial Safety Systems**: The system could improve safety in industrial environments by providing tiered, hands-free audio alerts. For example, it could track a worker's proximity to robotic machinery, generating a low hum in a safe zone that increases in pitch as they enter a warning zone, culminating in a loud alarm if a collision becomes imminent

By pursuing these applications, *VideoCsound* can evolve from a novel performance system into a versatile framework for creating meaningful, data-driven auditory displays.

6.4 Conclusion

In conclusion, this thesis successfully navigated the complex intersection of computer vision and algorithmic music, delivering *VideoCsound* as a functional and expressive prototype. The journey from implementation through demonstration not only validated the core concept but also rigorously defined a clear path for future enhancement. By transforming the ambient visual world into a source for structured sound, *VideoCsound* contributes a novel instrument to the field of data-driven art and stands as a foundation for new forms of interactive musical expression. By providing a bridge between pixel data and musical structure, VideoCsound opens new possibilities for interacting with and understanding the visual world through sound.

Bibliography

[1] Bakariya, B., Singh, A., Singh, H., Raju, P., Rajpoot, R., and Mohbey, K. K. Facial emotion recognition and music recommendation system using cnn-based deep learning techniques. *Evolving Systems 15*, 2 (Apr 2024), 641–658.

[2] Bay, H., Tuytelaars, T., and Van Gool, L. Surf: Speeded up robust features. In *Computer Vision – ECCV 2006* (Berlin, Heidelberg, 2006), A. Leonardis, H. Bischof, and A. Pinz, Eds., Springer Berlin Heidelberg, pp. 404–417.

[3] Boulanger, R., Ed. *The Csound Book: Perspectives in Software Synthesis, Sound Design, Signal Processing, and Programming*. MIT Press, Cambridge, MA, 2000. Includes bibliographical references and index.

[4] Boulanger, R. Trapped, 2025. Original work written in 1979. Csound source code; accessed 2025-08-25.

[5] Bradski, G. *The OpenCV Library*, vol. 25. United Business Media, Nov 2000.

[6] Chai, J., Zeng, H., Li, A., and Ngai, E. W. Deep learning in computer vision: A critical review of emerging techniques and application scenarios. *Machine Learning with Applications 6* (2021), 100134.

[7] Community, T. C. Csound api documentation, 2025. Describes the C API and wrappers for Python, Java, and other languages.

[8] Correia, N. N., Castro, D., and Tanaka, A. The role of live visuals in audience understanding of electronic music performances. In *Proceedings of the 12th International Audio Mostly Conference on Augmented and Participatory Sound and Music Experiences* (New York, NY, USA, 2017), AM '17, Association for Computing Machinery.

[9] Costanza, E., Shelley, S. B., and Robinson, J. Introducing audio d-touch: A tangible user interface for music composition and performance. In *Proceedings of the 6th International Conference on Digital Audio Effects (DAFx03)* (London, UK, 2003), Queen Mary University of London, pp. 118–122.

[10] Dinning, P. Videos for cats to watch : Birds being awesome - watch at home with your cat on tv : Catflix, Feb 2020. Accessed: 2025-08-18.

[11] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR abs/2010.11929* (2020).

[12] EarthCam. Earthcam live: Dublin, Ireland, Dec 2023. Accessed: 2025-08-18.

[13] EarthCam. Earthcam live: Mulberry Street (Little Italy, NYC), Feb 2025. Accessed: 2025-08-20.

[14] Fels, S., Gadd, A., and Mulder, A. Mapping transparency through metaphor: towards more expressive musical instruments. *Organised Sound 7*, 2 (2002), 109–126.

[15] Fischler, M. A., and Bolles, R. C. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM 24*, 6 (Jun 1981), 381–395.

[16] Fukushima, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics 36*, 4 (1980), 193–202.

[17] Gao, J., et al. Computer vision in healthcare applications. *Journal of Healthcare Engineering 2018* (Mar 2018), 5157020.

[18] Girshick, R., Donahue, J., Darrell, T., and Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition* (2014), pp. 580–587.

[19] Guo, Y., Liu, Y., Oerlemans, A., Lao, S., Wu, S., and Lew, M. S. Deep learning for visual understanding: A review. *Neurocomputing 187* (2016), 27–48.

[20] Henke, R. Lumiere, 2017. Artist's official website; description of the audiovisual performance system.

[21] Henke, R. Interview with robert henke about lumière. Ableton Blog, 2018. Accessed: 2025-08-24.

[22] Henke, R., and Eshmade, B. From the archive: Robert henke on Lumiere I and III. Barbican Read, Watch, Listen, Aug 2020. Accessed: 2025-08-24.

[23] Hermann, T., Hunt, A., and Neuhoff, J. G. *The Sonification Handbook*. Logos Publishing House, Berlin, Germany, 2011.

[24] Ho, J., Jain, A., and Abbeel, P. Denoising diffusion probabilistic models. *CoRR abs/2006.11239* (2020).

[25] Huang, S.-C., and Le, T.-H. Chapter 12 - object detection. In *Principles and Labs for Deep Learning*, S.-C. Huang and T.-H. Le, Eds. Academic Press, 2021, pp. 283–331.

[26] Hubel, D. H., and Wiesel, T. N. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology 160*, 1 (1962), 106–154.

[27] Hunt, A., Wanderley, M. M., and Kirk, R. Towards a model for instrumental mapping in expert musical interaction. In *Proceedings of the 2000 International Computer Music Conference* (2000), pp. 209–212.

[28] Jocher, G., and Qiu, J. Ultralytics YOLO11. https://github.com/ultralytics/ultralytics, 2024. Accessed: 2025-08-24.

[29] Khanam, R., and Hussain, M. Yolov11: An overview of the key architectural enhancements, 2024.

[30] Khazaei, M., Bahrani, A., and Tzanetakis, G. A real-time gesture-based control framework, 2025.

[31] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Commun. ACM 60*, 6 (May 2017), 84–90.

[32] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. Backpropagation applied to handwritten zip code recognition. *Neural Computation 1*, 4 (1989), 541–551.

[33] Lowe, D. Object recognition from local scale-invariant features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision* (1999), vol. 2, pp. 1150–1157 vol.2.

[34] Mahesh, V. 20 ways we're celebrating two decades of YouTube, Apr 2025.

[35] McMillan, L., and Varga, L. A review of the use of artificial intelligence methods in infrastructure systems. *Engineering Applications of Artificial Intelligence 116* (2022), 105472.

[36] Microsoft. Seeing AI, 2023. Accessed: 2025-08-21.

[37] Nouh, M., Nurse, J. R., Webb, H., and Goldsmith, M. Cybercrime investigators are users too! understanding the socio-technical challenges faced by law enforcement. In *Proceedings 2019 Workshop on Usable Security* (2019), USEC 2019, Internet Society.

[38] Owens, A., and Efros, A. A. Audio-visual scene analysis with self-supervised multisensory features. *CoRR abs/1804.03641* (2018).

[39] Pinot, F., et al. *The Ctcsound Application Programming Interface*. Csound Community, 2025. Comprehensive documentation for the ctcsound module, the Python wrapper for the Csound API.

[40] Powell, A., and Haynes, C. Social media data in digital forensics investigations. In *Digital Forensic Education: An Experiential Learning Approach*, X. Zhang and K.-K. R. Choo, Eds. Springer International Publishing, Cham, 2020, pp. 281–303.

[41] Pritch, Y., Rav-Acha, A., Gutman, A., and Peleg, S. Webcam synopsis: Peeking around the world. In *2007 IEEE 11th International Conference on Computer Vision* (2007), pp. 1–8.

[42] Raptis, G. E., Kavvetsos, G., and Katsini, C. Mumia: Multimodal interactions to better understand art contexts. *Applied Sciences 11*, 6 (2021).

[43] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. You only look once: Unified, real-time object detection, 2016.

[44] Rosenblatt, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review 65 6* (1958), 386–408.

[45] Rovan, J. B., Wanderley, M. M., Dubnov, S., and Depalle, P. Instrumental gestural mapping strategies as expressivity determinants in computer music performance. In *Kansei, The Technology of Emotion Workshop* (Genoa, Italy, 1997).

[46] Snavely, N., Seitz, S. M., and Szeliski, R. *Photo tourism: exploring photo collections in 3D*, 1 ed. Association for Computing Machinery, New York, NY, USA, 2023, pp. 835–846.

[47] Sobel, I. An isotropic 3x3 image gradient operator. *Presentation at Stanford A.I. Project 1968* (Feb 2014).

[48] Sonka, M., Hlavac, V., and Boyle, R. *Image processing, analysis and machine vision*. Springer, 2013.

[49] Sumby, W. H., and Pollack, I. Visual contribution to speech intelligibility in noise. *The journal of the acoustical society of america 26*, 2 (1954), 212–215.

[50] Tavares, T. F., Odowichuck, G., Zehtabi, S., and Tzanetakis, G. Audio-visual vibraphone transcription in real time. In *2012 IEEE 14th International Workshop on Multimedia Signal Processing (MMSP)* (2012), pp. 215–220.

[51] Viswanath, P., Chitnis, K., Swami, P., Mody, M., Shivalingappa, S., Nagori, S., Mathew, M., Desappan, K., Jagannathan, S., Poddar, D., Jain, A., Garud, H., Appia, V., Mangla, M., and Dabral, S. A diverse low cost high performance platform for advanced driver assistance system (adas) applications. In *2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (2016), pp. 819–827.

[52] Wei, Y., Hu, D., Tian, Y., and Li, X. Learning in audio-visual context: A review, analysis, and new perspective, 2022.

[53] Yurtsever, E., Lambert, J., Carballo, A., and Takeda, K. A survey of autonomous driving: Common practices and emerging technologies. *IEEE Access 8* (2020), 58443–58469.