Visual Reconstruction Of Played Music

MASTER OF SCIENCE DEGREE IN GEOINFORMATICS ENGINEERING

Author: **Jianwei Deng**

Lesson：Image Analysis and Computer vision

Student ID: 10973015 / 245188
Advisor: Vincenzo Caglioti
Academic Year: 2025/6/15

Abstract

This project presents a lightweight system for detecting piano key presses from finger movements captured in a 2D video. Using MediaPipe for hand landmark detection, the system extracts fingertip trajectories across video frames and classifies each frame as a "press" or "no press" event via a trained machine learning model(Random Forest classification). To accurately identify which key is pressed, a custom keyboard mapping is created based on the physical layout of the keyboard in the input video. A Random Forest classifier is trained on normalized fingertip coordinates, achieving over 90% accuracy in distinguishing press events. Finally, a visualization module overlays detected key presses and finger positions on the video, providing real-time feedback. This approach enables efficient and low-cost piano interaction analysis without requiring depth sensors or wearable devices.

Key-words: Finger Detection; MediaPipe; Video-based Interaction; Random Forest Classifier

Contents

1 Chapter one

In this chapter all useful information about the project are reported.

1.1. Introduction

1.1.1. Background and Motivation

In recent years, gesture-based human-computer interaction (HCI) has attracted widespread attention due to its intuitive interaction method. In particular, finger-level gesture recognition provides important support for applications such as virtual instrument interfaces, sign language recognition, and fine robot control. Among them, applying gesture recognition to piano playing is a task with broad application prospects, which requires high-precision and low-latency detection and recognition of the interaction between finger movements and piano keys.

Existing key detection systems mostly rely on dedicated hardware devices, such as depth cameras (such as Leap Motion, Intel RealSense) or sensor gloves. Such systems are expensive, complex to use, and not user-friendly. Therefore, this project aims to explore whether it is possible to achieve a low-cost and easily accessible piano key detection system using only standard RGB videos (such as those taken by smartphones) in combination with computer vision and machine learning methods, thereby providing technical support for virtual performance and motion analysis to a wider range of users.

1.1.2. Objective of the Project

The goal of this project is to develop a lightweight but complete pipeline for detecting piano key events based on RGB video input. Specifically, the goals are to:

- Track fingertip motion using MediaPipe from RGB video.
- Extract structured features of finger positions.
- Manually annotate a small video dataset to label key events.
- Train a supervised machine learning model to classify key presses.
- Detect which fingers press which piano keys in a test video.
- Visually annotate and output a new video showing the key detection results.

The system is designed to be low-cost, accessible, and easily reproducible without the need for any specialized sensors or external hardware.

1.2. Problem Formulation

1.2.1. Task Definition

The goal of this project is to detect piano key press events from an input RGB video by analyzing frame-by-frame hand movements. Specifically, the task can be formulated as a binary classification problem:

For each video frame, predict whether the piano key is pressed (1) or not (0) based on the 2D/3D position of the fingertip.

After a press is detected, further spatial matching with a predefined keyboard layout is used to identify the finger performed the action and corresponding key pressed.

1.2.2. Challenges

1. No depth images

Because the input images only use standard RGB videos, it is difficult to extend from 2D images to accurate 3D inference

2. Distinguishing between fingers and keys

Distinguishing between a finger hovering near a key and actually pressing a key is visually ambiguous, especially from a single view.

3. Frame-level ground truth annotation

Accurate manual annotation of "pressed" or "not pressed" at the frame level is susceptible to subjective factors and prone to errors.

4. Limited generalization ability

Keyboard key positions are set to be fixed, so the trained model may not generalize well to other keyboards or camera angles.

5. No temporal modeling

Current methods evaluate each frame independently and lack motion patterns that unfold over time (such as pressing and releasing).

1.3 Related Work:Finger Tracking &Tapping Detection

1.3.1.Reference Thesis description

A representative Thesis in this field is the study "Barehanded Music: Real-time Hand Interaction for Virtual Piano".
This system utilizes an RGB-D sensor (DepthSenser 325) to track finger motion and detect piano key presses in real time. The problem is formulated as a two-stage machine learning pipeline:

Stage 1: Fingertip Tracking (Joint Regression)
A Random Regression Forest is trained to predict 3D positions of 7 hand joints from depth images. The input to the model is depth context descriptors extracted from the hand region, and each leaf node of the forest votes for the joint offset.

Stage 2: Tapping Detection (Temporal Classification)
Based on fingertip height trajectories over a sliding time window (N=5), two SVM classifiers are trained per finger to detect:

Tap Down (key press)

Tap Up (key release)

The combination of both stages allows the system to determine which finger pressed which key at which moment. Visual and audio feedback are provided accordingly.

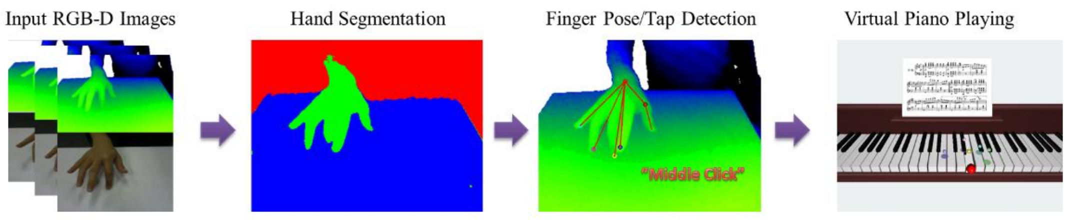


Figure 2: *Fingertip tracking and tapping detection for virtual piano playing.*

1.3.2.Methods Features

- Hand Segmentation

1. Use skin color detection to obtain the hand area in the RGB image.
2. Use the depth map for enhancement, fit the desktop plane through RANSAC, remove background points, and improve segmentation accuracy.

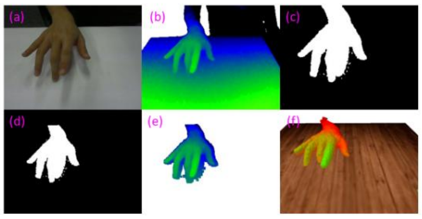


Figure 3: *Hand segmentation. (a) and (b): input RGB-D images; (c) skin mask in RGB image; (d) skin mask mapped to depth image; (e) hand segmentation without 3D plane fitting; (f) fitted 3D plane and final hand segmentation.*

• Hand Pose Estimation

1. Pixel sampling is performed on the hand area, and each pixel is used as a "voter" to predict the relative position of 7 key points through the Random Regression Forest.

2. Each tree outputs the predicted offset and weight at the leaf node. Combined with the Gaussian kernel weighted density estimation method, the 3D position of each joint point is obtained.

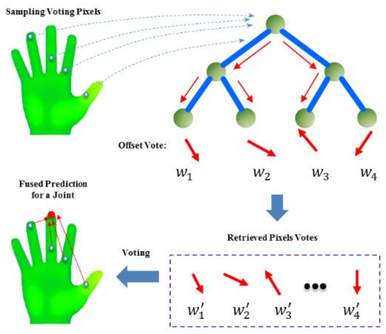


Figure 4: *Hand pose prediction with the random forest. The red arrows represent the votes of the relative offset pointing to the joint to be predicted. The red circle in the left-bottom denoted the predicted middle fingertip position.*

3. Finger motion trajectory analysis and detection (Motion Feature Extraction)

Project the trajectory height of each finger to the desktop normal vector to obtain the height change in the vertical direction of the desktop.

Record the height change of the fingertip in each frame to determine the press (lowest descent) and lift (start of rise) events.

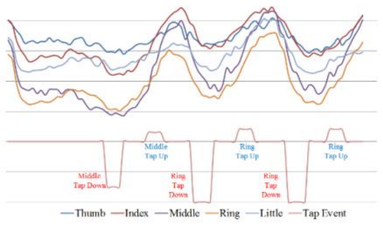


Figure 5: *Finger tapping detection: heights relative to table plane for all finger tips and action labelled*

4. Tap event detection (Tap Detection)

Use a sliding window to extract the features of each fingertip in the last 5 frames, including:

Relative height difference (5x5 combination)

Time series statistical features such as mean, variance, slope, peak range, etc.

Use the SVM model for binary classification: Tap Down / Tap Up / None

Add voting mechanism + slope threshold filtering to eliminate false detection (such as hand shaking, etc.)

5. Virtual Piano Key Interaction and Feedback (Key Press & Feedback)

Determine which key to trigger based on predicted finger position and key area.

Display visual feedback (keys) and audio playback (virtual sound response).

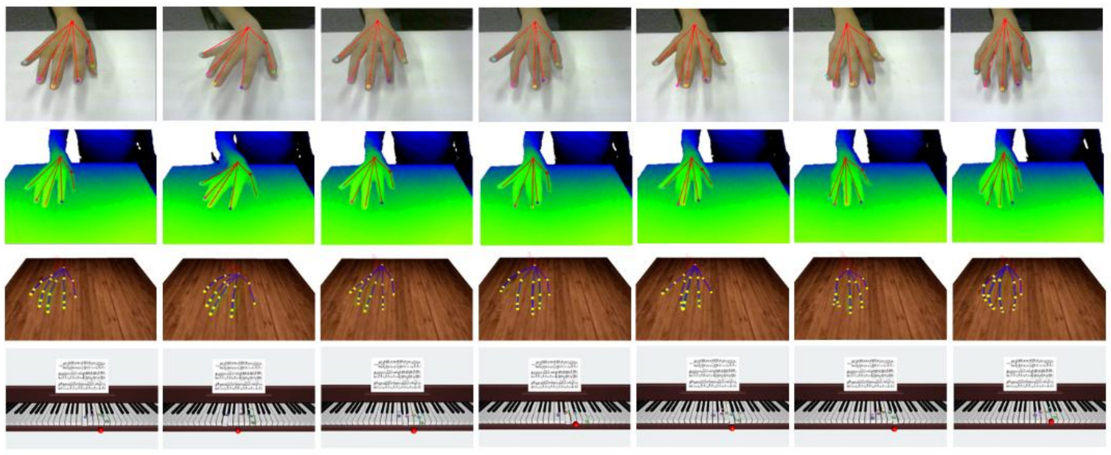


Figure 8: *Exemplar frames of hand pose prediction. Rows 1 & 2: input RGB and depth images overlaid with predicted 7 hand joints. Row 3: reconstructed hand skeleton via applying inverse kinematic to the joints. Row 4: playing virtual piano. Zoom in to see the details.*

The system fully integrates image processing (skin color + depth segmentation), regression forest (joint detection), time series classification (SVM + sliding window) and visual-audio feedback system to form a real-time hand-virtual piano interaction system.

1.4. Proposed Method

Figures, Tables and Algorithms must contain a Caption that describe their content and must be properly referred in the text.

1.4.1 Overview Methodology

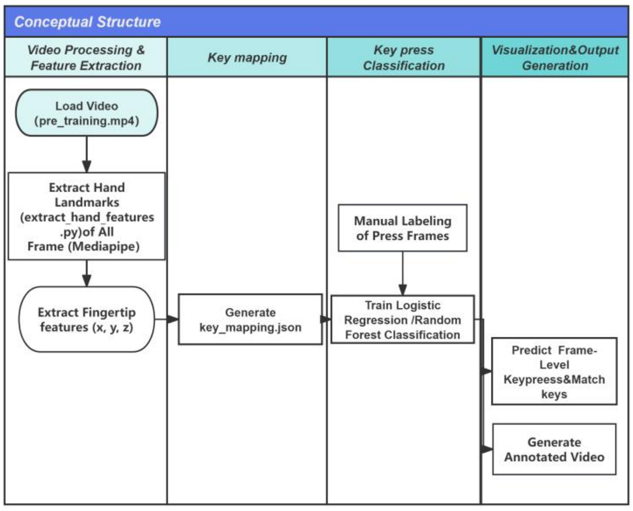

Conceptual Structure
Video Processing & Feature Extraction
Key mapping
Key press Classification
Visualization&Output Generation
Load Video (pre_training.mp4)
Extract Hand Landmarks (extract_hand_features .py)of All Frame (Mediapipe)
Extract Fingertip features (x, y, z)
Generate key_mapping.json
Manual Labeling of Press Frames
Train Logistic Regression /Random Forest Classification
Predict Frame-Level Keypreess&Match keys
Generate Annotated Video

Figure 2: Lane_Diagram of Methodology.

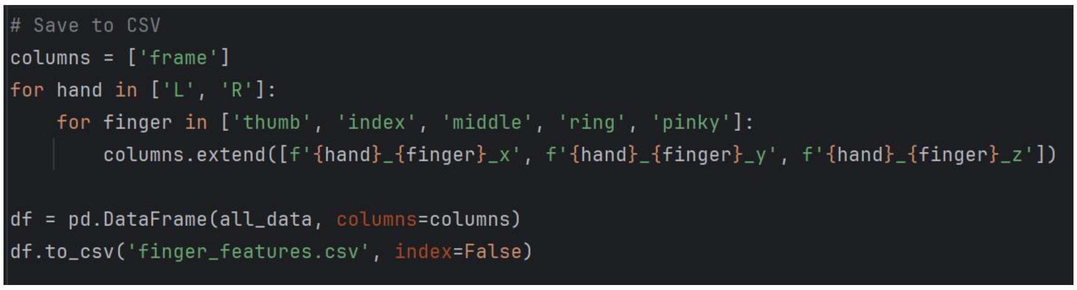
1.4.2 Methodology Details

• Video-Based Fingertip Feature Extraction

Using MediaPipe Hands, each frame of the input video is processed to extract the 3D coordinates of 10 fingertip key points per frame and save them as a structured CSV file.

Code: extract_hand_features.py

Output: finger_features.csv



```
# Save to CSV
columns = ['frame']
for hand in ['L', 'R']:
    for finger in ['thumb', 'index', 'middle', 'ring', 'pinky']:
        columns.extend([f'{hand}_{finger}_x', f'{hand}_{finger}_y', f'{hand}_{finger}_z'])

df = pd.DataFrame(all_data, columns=columns)
df.to_csv('finger_features.csv', index=False)
```

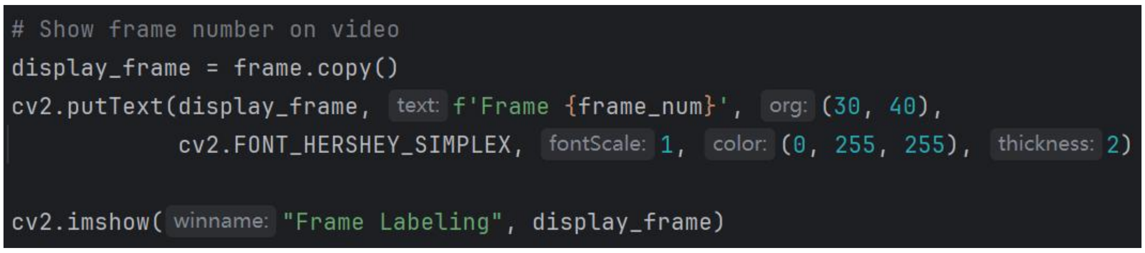
Figure 3: Fingertip Feature Extraction.

• Manual Frame Labeling

Manually label each frame whether it is a key frame (press / no-press) as a supervision signal.

Code: manual_label_tool.py

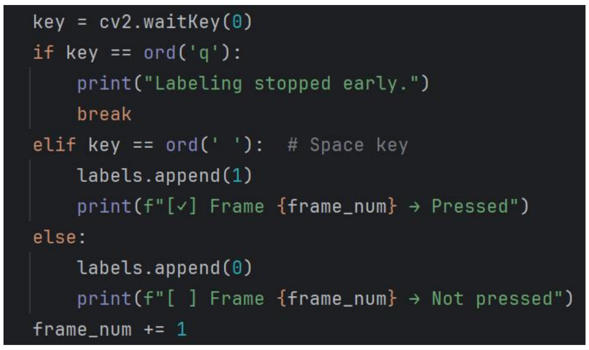
Output: labels.csv



```
# Show frame number on video
display_frame = frame.copy()
cv2.putText(display_frame,  text: f'Frame {frame_num}',  org: (30, 40),
            cv2.FONT_HERSHEY_SIMPLEX,  fontScale: 1,  color: (0, 255, 255),  thickness: 2)

cv2.imshow( winname: "Frame Labeling", display_frame)
```

Figure 4: Read Frame on video.



```
key = cv2.waitKey(0)
if key == ord('q'):
    print("Labeling stopped early.")
    break
elif key == ord(' '):  # Space key
    labels.append(1)
    print(f"[✓] Frame {frame_num} → Pressed")
else:
    labels.append(0)
    print(f"[ ] Frame {frame_num} → Not pressed")
frame_num += 1
```

Figure 5: Manual Lableing Operations

• Train Machine Learning Model(Random Forest)

Use the random forest model to train binary classification based on fingertip features and labels (pressed/not pressed).

Code: train_classifier.py

Output: press_classifier.pkl

Main Steps:

1.Load Data

2.Merge Features and Labels

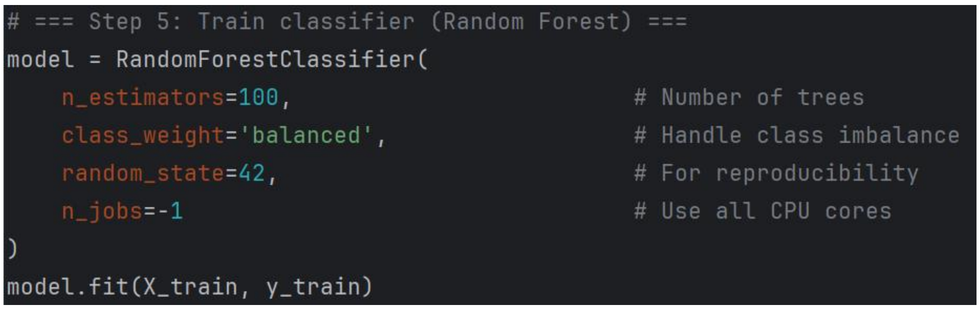
3.Prepare Training Inputs

4.Split Train/Test Sets

5.Train Random Forest Model

6.Evaluate Model Performance

7.Save Trained Model



```
# === Step 5: Train classifier (Random Forest) ===
model = RandomForestClassifier(
    n_estimators=100,                          # Number of trees
    class_weight='balanced',                   # Handle class imbalance
    random_state=42,                           # For reproducibility
    n_jobs=-1                                  # Use all CPU cores
)
model.fit(X_train, y_train)
```

Figure 6: Random Forest Classfier

• Predict Key Presses & Save as JSON

Use the model to predict whether it is a key press frame for each frame, and identify which key is pressed by which finger.

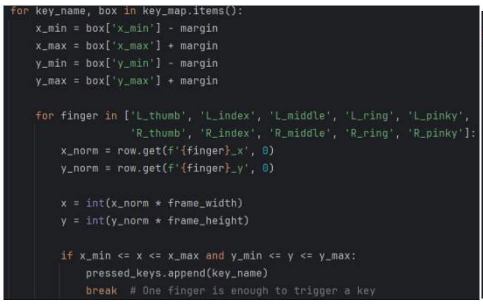
Including：

1.Comparison of the finger position coordinates with the bounding box of the key position

2.Only frames where a key is predicted to be pressed are kept and mapped.

Code: predict_key_presses.py

Output: press_events.json

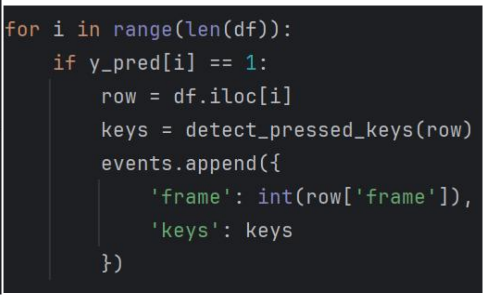


```
for key_name, box in key_map.items():
    x_min = box['x_min'] - margin
    x_max = box['x_max'] + margin
    y_min = box['y_min'] - margin
    y_max = box['y_max'] + margin

    for finger in ['L_thumb', 'L_index', 'L_middle', 'L_ring', 'L_pinky',
                   'R_thumb', 'R_index', 'R_middle', 'R_ring', 'R_pinky']:
        x_norm = row.get(f'{finger}_x', 0)
        y_norm = row.get(f'{finger}_y', 0)

        x = int(x_norm * frame_width)
        y = int(y_norm * frame_height)

        if x_min <= x <= x_max and y_min <= y <= y_max:
            pressed_keys.append(key_name)
            break  # One finger is enough to trigger a key
```



```
for i in range(len(df)):
    if y_pred[i] == 1:
        row = df.iloc[i]
        keys = detect_pressed_keys(row)
        events.append({
            'frame': int(row['frame']),
            'keys': keys
        })
```

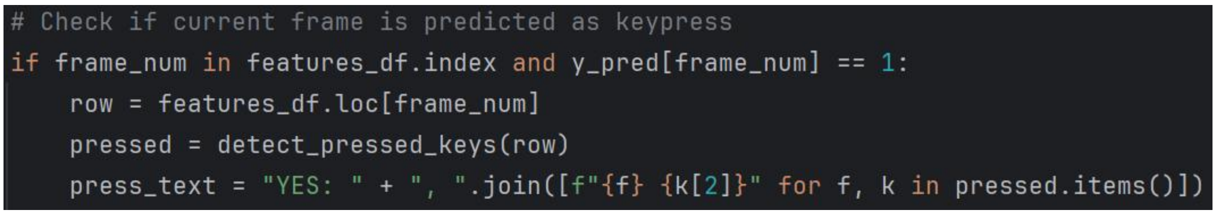
Figure 7、8: Compare finger positions & Save only pressed keys

• Final Video Visualization

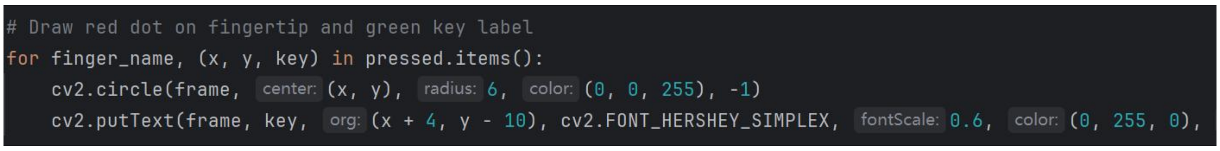
Visualize the prediction results into an annotated video, display the key action and corresponding finger of each frame and output.

Code: visualize_dynamic_keys.py

Output: output_annotated_with_keys.mp4



```
# Check if current frame is predicted as keypress
if frame_num in features_df.index and y_pred[frame_num] == 1:
    row = features_df.loc[frame_num]
    pressed = detect_pressed_keys(row)
    press_text = "YES: " + ", ".join([f"{f} {k[2]}" for f, k in pressed.items()])
```



```
# Draw red dot on fingertip and green key label
for finger_name, (x, y, key) in pressed.items():
    cv2.circle(frame, center: (x, y), radius: 6, color: (0, 0, 255), -1)
    cv2.putText(frame, key, org: (x + 4, y - 10), cv2.FONT_HERSHEY_SIMPLEX, fontScale: 0.6, color: (0, 255, 0),
```

Figure 9、10: Compare finger positions & Save only pressed keys

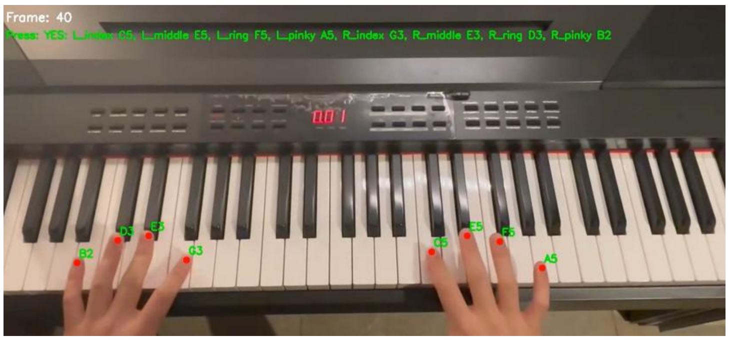
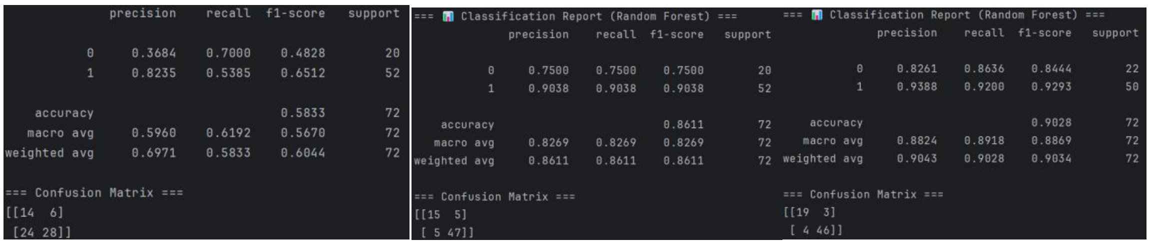

Frame: 40
Press: YES: L_index C5, L_middle E5, L_ring F5, L_pinky A5, R_index G3, R_middle E3, R_ring D3, R_pinky B2
B2
D3
E3
G3
C5
E5
F5
A5

Figure 11: Final result output

1.4.3 Result Visualization

To evaluate the accuracy of finger pressure detection, we conducted quantitative tests using manually labeled datasets. We also used trained models (logistic regression, random forest classifier) to evaluate the performance on the test set.



```
              precision    recall  f1-score   support

           0     0.3684    0.7000    0.4828        20
           1     0.8235    0.5385    0.6512        52

    accuracy                         0.5833        72
   macro avg     0.5960    0.6192    0.5670        72
weighted avg     0.6971    0.5833    0.6044        72

=== Confusion Matrix ===
[[14  6]
 [24 28]]
```

```
=== 📊 Classification Report (Random Forest) ===
              precision    recall  f1-score   support

           0     0.7500    0.7500    0.7500        20
           1     0.9038    0.9038    0.9038        52

    accuracy                         0.8611        72
   macro avg     0.8269    0.8269    0.8269        72
weighted avg     0.8611    0.8611    0.8611        72

=== Confusion Matrix ===
[[15  5]
 [ 5 47]]
```

```
=== 📊 Classification Report (Random Forest) ===
              precision    recall  f1-score   support

           0     0.8261    0.8636    0.8444        22
           1     0.9388    0.9200    0.9293        50

    accuracy                         0.9028        72
   macro avg     0.8824    0.8918    0.8869        72
weighted avg     0.9043    0.9028    0.9034        72

=== Confusion Matrix ===
[[19  3]
 [ 4 46]]
```

Figure 12: Comparison of machine learning classifier results

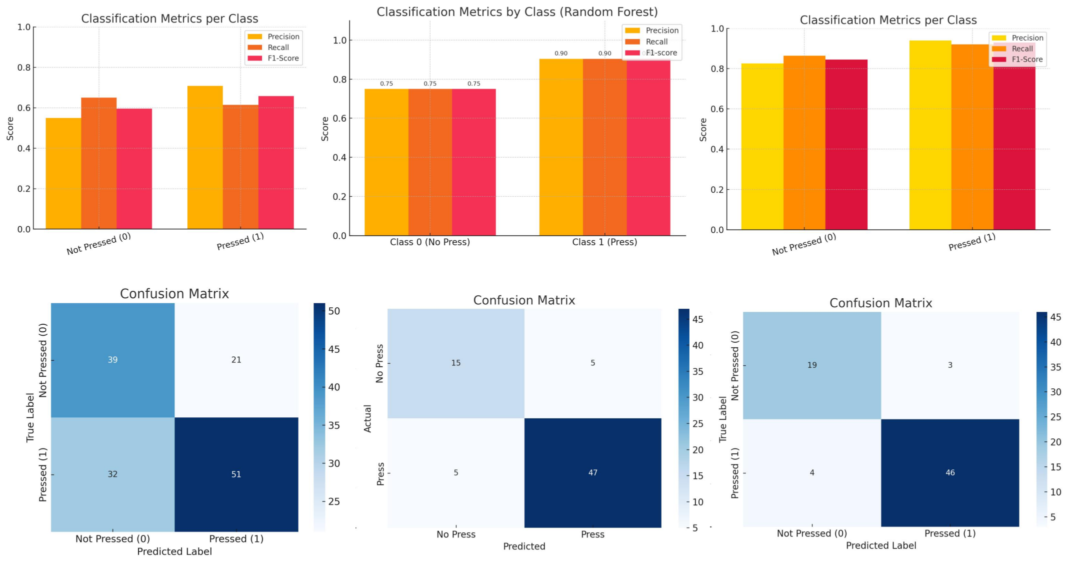


Figure 13: Comparison of ML classifier precision(Bar and Confusion Metrics)

In Figure 13, the classifier model on the far right performs best. It achieves higher precision and recall when detecting piano key press events, and has the highest number of correct classifications (true positives and true negatives) and the lowest number of false positives in the confusion matrix.

Thus, the model on the far right outperforms the random forest model in the middle in terms of overall accuracy and stability.

1.5. Implementation Details

1.5.1 Environment Setup

It is recommended to use a virtual environment to manage dependencies. For example, using Python 3.10

1.# Create a virtual environment named "venv310"

```
python3.10 -m venv venv310
```

2.# Activate the environment

# On Windows:

```
venv310\Scripts\activate
```

# On macOS/Linux:

```
source venv310/bin/activate
```

1.5.2 Software Tools and Libraries

This project was developed using the following software tools and libraries:

• **Python 3.10**
The main programming language used for the entire project, due to its flexibility and extensive ecosystem for computer vision (Mediapipe) and machine learning.

• **OpenCV (cv2)**
Used for video processing, frame-by-frame image handling, display, and writing annotated output videos.

• **MediaPipe**
A Google open-source library for real-time hand and finger landmark detection, used to extract 3D coordinates of 21 hand keypoints per frame.

• **Pandas**
For structured data manipulation and storage of finger position features (finger_features.csv) and frame-level labels (labels.csv).

• **NumPy**
For numerical operations and feature transformations.

• **Scikit-learn (sklearn)**
Used for:

Splitting training and testing sets.

Training a Random Forest Classifier for key press detection.

Generating classification reports and confusion matrices.

• **Joblib**
For efficient saving and loading of trained machine learning models (press_classifier.pkl).

• **JSON (built-in)**
To store and load key mapping information (key_mapping.json) and final detected key press events (press_events.json).

1.5.3 File Structure and Functionality

```
Project/
├── extract_hand_features.py        # Extract fingertip positions using MediaPipe
├── manual_label_tool.py             # Manually label frames as keypress or not
├── train_press_classifier.py         # Train classifier based on finger features
├── python predict_pressed_keys.py    # Predict keypress events from features
├── python visualize_dynamic_keys.py # Generate annotated output video
```

```
├──── pre_training.mp4                  # Input video
├──── finger_features.csv               # Extracted features per frame
├──── labels.csv                        # Frame-wise labels (keypress or not)
├──── press_classifier.pkl              # Saved trained model
├──── key_mapping.json                  # Piano key region mapping
├──── press_events.json                     # Output detected keypress events
├──── output_annotated_with_keys.mp4    # Final annotated video
└──── keyboard.jpg                          # Used for manual key mapping
```

1.5.4 Runtime Instructions (based on README.MD)

Step 1: Create Virtual Environment (Recommended)

```
python -m venv venv310          # Install python
source venv310/bin/activate     # On Windows: .\venv310\Scripts\Activate
```

Step 2: Install Required Libraries

```
pip install -r requirements.txt
pip install opencv-python mediapipe pandas numpy scikit-learn joblib
```

Step 3: Feature Extraction
Run the script to extract fingertip 3D coordinates from video:

```
python extract_hand_features.py
```

Step 4: Manually Label Pressed Frames
Label each frame as press (1) or non-press (0) by pressing space or other keys:

```
python manual_label_tool.py
```

Step 5: Train Classifier
Train a Random Forest model to distinguish pressed vs. non-pressed frames:

```
python train_press_classifier.py
```

Step 6: Predict Pressed Keys
Combine the classifier and geometric rules to determine which keys were pressed:

```
python predict_pressed_keys.py
```

This creates:
- `press_events.json`: predicted press events

Step 7: Visualize Output
Overlay predictions on the video:

```
python visualize_dynamic_keys.py
```

Result saved to: `output_annotated_with_keys.mp4`

2.Conclusion and future developments

This chapter mainly summarizes the content presented in the project and provides areas for future improvement.

2.1 Conclusion

2.1.1 Summary of Contributions

This project successfully developed a lightweight yet complete pipeline for detecting piano key presses based on fingertip motion analysis from video. The key contributions are as follows:

• A simple and effective data collection method using MediaPipe to track fingertip 3D positions from RGB video input;

• A feature extraction pipeline that converts per-frame hand keypoints into structured numerical data for learning;

• Manual labeling of video frames to build a supervised dataset for the binary classification of press/no-press events;

• A trained Random Forest classifier that achieves over 90% accuracy on detecting piano key press events;

• An automated video annotation system that shows which finger presses which key, frame by frame;

• The entire system operates using only a standard RGB webcam (e.g., smartphone camera) without the need for depth sensors or expensive hardware.

2.1.2 Limitations

• Although this project has basically established a video-based piano key detection system, there are still several limitations:

• Press judgment is based only on a binary classification model: the current classifier can only determine whether there is a press event in a certain frame, but cannot accurately distinguish the difference between finger "hovering" and "real press", resulting in a certain misjudgment rate.

• Keyboard mapping assumes static boundaries: this system relies on predefined static keyboard boundaries, which makes the system less adaptable to different piano models or different camera angles and lacks universality.

• The annotation method is purely manual and the data scale is small: the current training data relies entirely on manual frame-by-frame annotation, and the data volume is limited, which affects the robustness and generalization ability of the classifier.

• No time modeling method is used: although finger movements are continuous and time-dependent, this project does not introduce time series modeling methods such as LSTM or HMM, resulting in the model failing to fully utilize time context information.

2.2 Future Improvements

• Introduce temporal models (e.g., recurrent neural networks, transformers) to more robustly capture motion patterns over time.

• Collect larger, more diverse datasets, including users of different ages, hand sizes, and focus angles.

• Add more detailed finger classification to identify which finger is active, not just that "some" partitions occurre