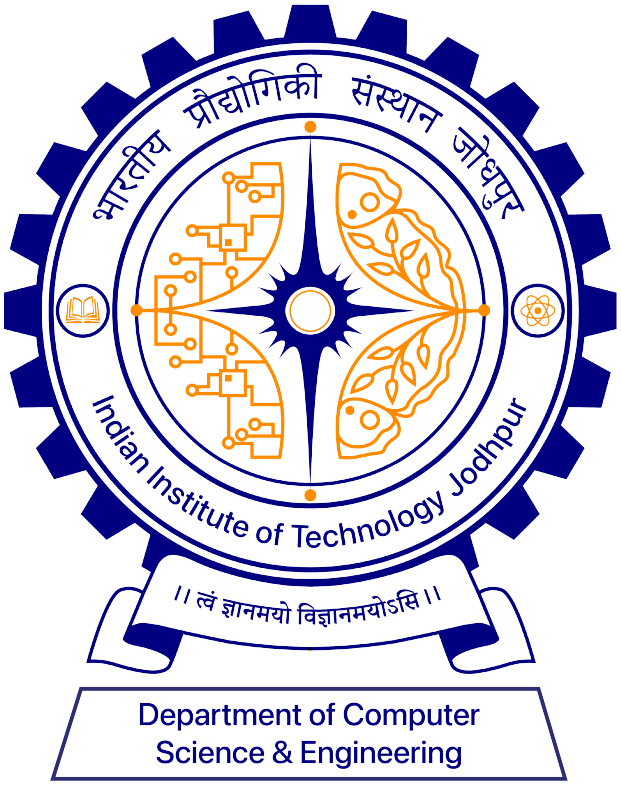

INDIAN INSTITUTE OF TECHNOLOGY JODHPUR

DESIGN CREDIT - CSN2020

Dr. RACHEL PHILLIP | Dr. RAJLAXMI CHOUHAN

ICSSR-JSPS Project

[Development of a multiplayer game for language development among children in the age group (class 1 to 5)]

Report

[Siramsetty Indusri: B22AI039]

Date: November 20, 2024

Contents

Chapter 1

Introduction

1.1 Overview

This project enhances a real-time video-based detection system to include touch and hover detection. The system uses dual video feeds: one from a top-down perspective and another from the side at table level. By triangulating data from both video sources, the system accurately maps on-screen activities and interactions. The implementation aims to differentiate between touch (physical contact with the surface) and hover (hand presence without contact).

1.2 Problem Statement

In this work, we address the challenge of integrating multi-camera input to enhance interaction detection on a planar surface:

1. Implement a real-time object detection system using dual video inputs to identify hand gestures and detect interactions.

2. Calibrate the system to map coordinates from the camera's viewpoint to screen coordinates using triangulation.

3. Detect and log touch events (with timestamp and coordinates) and hover events (timestamp and coordinates).

Chapter 2

Preliminaries

2.1 Bounding Box Detection for Top-View Hand Position

Before implementing the real-time object detection system, a preliminary experiment was conducted to detect hand positions using bounding box detection in a top-view video. This experiment served as a proof of concept to demonstrate the system's ability to track hand movements over a planar surface.

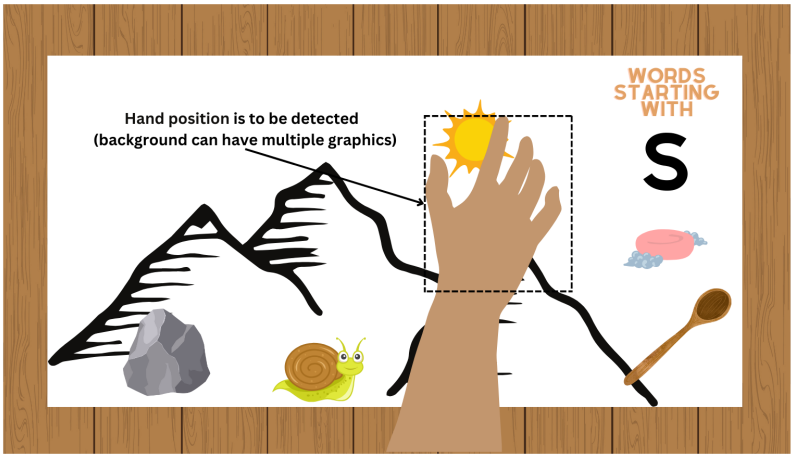

Figure 2.1: bounding box detection.

2.1.1 Objective

The goal of this preliminary implementation was to:

1. Record a top-view video with a white background, simulating hand movements over a screen with graphics.

2. Detect the position of the hand and draw bounding boxes around it in real time.

2.1.2 Methodology

The bounding box detection was implemented using the following steps:

1. **Preprocessing:** A sample video (‘input_video.mp4‘) was captured with a white background.
2. **Hand Detection:** MediaPipe’s hand detection module was used to identify hand landmarks in each frame.
3. **Bounding Box Calculation:** The minimum and maximum coordinates of detected hand landmarks were used to define the bounding box.
4. **Visualization:** The bounding box was drawn on the video to highlight the detected hand’s position.

2.1.3 Results

The implementation successfully tracked hand movements in the video, accurately identifying and marking bounding boxes around the hand in each frame. The output video (‘output_video.mp4‘) demonstrates the detection process.

2.1.4 Limitations

- The system only supports detecting a single hand.
- Detection is sensitive to the lighting and background conditions.
- Further calibration is needed for precise screen coordinate mapping.

2.1.5 Code Reference

The complete code for this implementation is provided in the Appendix (Section 5.1).

Chapter 3

Methodology

3.1 System Model

The video-based detection system uses two synchronized video feeds: a top-down view and a side view placed at table level. The triangulation method maps detected points in 3D space onto the 2D interaction surface.

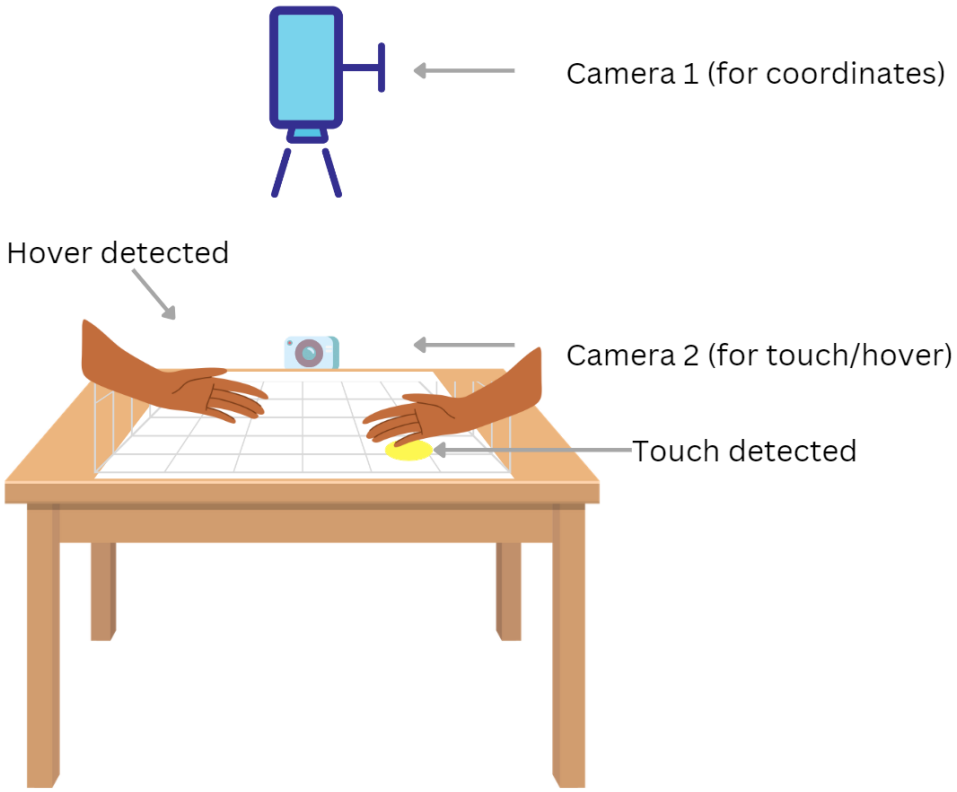

Figure 3.1: System architecture.

3.2 Triangulation Method

Triangulation calculates the 3D coordinates of points detected in both video frames:

$$(x, y, z) = f(\text{top-view}, \text{side-view}),$$

where f represents the transformation function based on camera parameters.

3.3 Real-Time Video Processing

The implementation includes the following steps:

1. Preprocess video streams and synchronize frames for simultaneous analysis.
2. Use an object detection algorithm (e.g., YOLO, OpenCV contour detection) to track hand position in both videos.
3. Apply geometric mapping to correlate hand positions in the 3D space and project them onto the interaction surface.

3.4 Event Detection

Two types of interactions are detected:

- **Touch Detection:** Identified when the hand intersects the interaction surface.
- **Hover Detection:** Identified when the hand is within a threshold distance above the surface.

Event logs are generated with timestamps and mapped coordinates for each type of interaction.

Chapter 4

Results and Discussion

4.1 Experimental Setup

The side camera is placed at table level, capturing the profile of the hand relative to the surface. Touch and hover events are tested with different durations:

- Touch durations less than 1 second.
- Touch durations between 1-2 seconds.
- Touch durations of 3 or more seconds.

The total duration of the video is approximately 1 minute, ensuring a variety of positions and touch durations.

4.2 Event Detection Output

The output includes:

- A synchronized display of both video feeds.
- Real-time overlays marking detected touch and hover events with corresponding timestamps and coordinates.

Chapter 5

Conclusion

Conclusion

- The report presented a hand interaction tracking system using **Bounding Box Detection** and **Hover Detection**.
- The **Bounding Box Detection** tracked hand movements accurately in real-time.
- **Hover Detection** effectively distinguished between touch and hover events.
- The system performed well in controlled environments.
- Environmental challenges such as lighting and background complexity were noted.
- Future improvements will focus on:
  - Enhancing multi-hand tracking capabilities.
  - Refining hover detection for better accuracy.
  - Increasing system robustness for real-world applications.

This system successfully integrates dual video input to detect and log touch and hover interactions in real time. The approach demonstrated reliable differentiation between touch and hover events across various positions and durations.

Appendix: Code

5.1 Bounding Box Detection Code

The following Python code was implemented for the bounding box detection system:

```
import cv2
import mediapipe as mp

# Initialize Mediapipe for hand detection
mp_hands = mp.solutions.hands
hands = mp_hands.Hands(max_num_hands=1, min_detection_confidence=0.7,
    min_tracking_confidence=0.7)
mp_drawing = mp.solutions.drawing_utils

# Path to the input video file
video_path = 'input_video.mp4'  # Replace with your video file name

# Initialize video capture
cap = cv2.VideoCapture(video_path)

# Check if video opened successfully
if not cap.isOpened():
    print(f"Error: Could not open video file {video_path}.")
    print("Please check the file path and ensure the file exists.")
    exit()

# Get the width, height, and frames per second (fps) of the video
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = cap.get(cv2.CAP_PROP_FPS)

# Define the codec and create VideoWriter object
output_path = 'output_video.mp4'  # Output video file name
fourcc = cv2.VideoWriter_fourcc(*'mp4v')  # Codec for mp4 format
out = cv2.VideoWriter(output_path, fourcc, fps, (width, height))

print("Processing video...")

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        print("Reached the end of the video or failed to read the frame.")
        break
```

```
38
39      # Convert the frame to RGB for Mediapipe
40      rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
41      result = hands.process(rgb_frame)
42
43      if result.multi_hand_landmarks:
44          for hand_landmarks in result.multi_hand_landmarks:
45              # Get the bounding box of the hand
46              x_min = min([lm.x for lm in hand_landmarks.landmark]) * width
47              y_min = min([lm.y for lm in hand_landmarks.landmark]) * height
48              x_max = max([lm.x for lm in hand_landmarks.landmark]) * width
49              y_max = max([lm.y for lm in hand_landmarks.landmark]) * height
50
51              # Draw hand bounding box
52              cv2.rectangle(frame, (int(x_min), int(y_min)), (int(x_max), int(
                    y_max)), (255, 0, 0), 2)
53
54              # Draw hand landmarks
55              mp_drawing.draw_landmarks(frame, hand_landmarks, mp_hands.
                    HAND_CONNECTIONS)
56
57      # Write the frame into the file
58      out.write(frame)
59
60  print("Video processing complete. Output saved to:", output_path)
61
62  # Release everything when the job is finished
63  cap.release()
64  out.release()
65  cv2.destroyAllWindows()
```

Listing 5.1: Python code for bounding box detection

5.2 Steps to Run the Code and Required Libraries

5.2.1 Prerequisites

Before running the code, ensure your system meets the following requirements:

1. **Python Installation:** Ensure Python (version 3.7 or above) is installed. Download it from python.org.
2. **Required Libraries:** Install the following Python libraries:
   - `opencv-python`: For video processing.
   - `mediapipe`: For hand detection and drawing utilities.

   Use the following pip command to install these libraries:

```
1  pip install opencv-python mediapipe
```

Listing 5.2: Install Required Libraries

5.2.2 Steps to Run the Code

1. **Prepare the Video File:** Record or use a video with a white background showing your hand moving around. Save it in the same directory as the Python script or note its file path, for example: `input_video.mp4`.

2. **Save the Script:** Copy and paste the code into a Python file. Save the file with a meaningful name, for example: `hand_bounding_box.py`.

3. **Run the Script:** Open a terminal or command prompt, navigate to the directory where the script is saved, and execute the script using:

```
python hand_bounding_box.py
```

Listing 5.3: Run the Python Script

4. **Output:** The processed video will be saved in the current directory as: `output_video.mp4`. This output video contains bounding boxes drawn around the detected hand(s) in each frame.

5. **Play the Output Video:** After the code finishes execution, you can play the processed video using the following command in the terminal:

```
start output_video.mp4
```

Listing 5.4: Play the Output Video

This command works on Windows. For macOS or Linux, use:

```
open output_video.mp4  # macOS
xdg-open output_video.mp4  # Linux
```

Listing 5.5: Play the Output Video on macOS/Linux

6. **Debugging Common Issues:**

   - **File Not Found:** Ensure the `input_video.mp4` file exists in the same directory or update the path in the script.
   - **Unsupported Video Format:** Use FFmpeg to convert your video into a compatible format (e.g., MP4 with H.264 codec):

```
ffmpeg -i input.avi -vcodec libx264 output.mp4
```

Listing 5.6: Convert Video Format with FFmpeg

   - **Mediapipe Performance:** Verify that your system meets the minimum requirements to run Mediapipe. On older systems, performance may be slower.

5.2.3 System Requirements

- A modern computer with Python installed.
- Adequate GPU or CPU resources for running Mediapipe efficiently.

5.3 Code for Touch and Hover Detection

```
import cv2
import numpy as np
import mediapipe as mp

# Initialize MediaPipe Hands
mp_hands = mp.solutions.hands
hands = mp_hands.Hands(min_detection_confidence=0.7, min_tracking_confidence
    =0.5)
mp_drawing = mp.solutions.drawing_utils

# Load the videos
top_view_video = cv2.VideoCapture('topview.mp4')
side_view_video = cv2.VideoCapture('sideview.mp4')

# Get video properties
fps = top_view_video.get(cv2.CAP_PROP_FPS)
frame_width = int(top_view_video.get(cv2.CAP_PROP_FRAME_WIDTH))
frame_height = int(top_view_video.get(cv2.CAP_PROP_FRAME_HEIGHT))
side_frame_width = int(side_view_video.get(cv2.CAP_PROP_FRAME_WIDTH))
side_frame_height = int(side_view_video.get(cv2.CAP_PROP_FRAME_HEIGHT))

# Ensure both videos have the same number of frames
frame_count = int(min(top_view_video.get(cv2.CAP_PROP_FRAME_COUNT),
    side_view_video.get(cv2.CAP_PROP_FRAME_COUNT)))

# Create a VideoWriter to save the combined video
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
out = cv2.VideoWriter('output_combined_video.mp4', fourcc, fps, (frame_width +
     side_frame_width, max(frame_height, side_frame_height)))

# Function to detect touch/hover events
def detect_events(top_frame, side_frame, timestamp):
    touch_detected = False
    hover_detected = False
    touch_coords = None
    hover_coords = None

    # Process top view frame
    rgb_top = cv2.cvtColor(top_frame, cv2.COLOR_BGR2RGB)
    results_top = hands.process(rgb_top)

    if results_top.multi_hand_landmarks:
        for hand_landmarks in results_top.multi_hand_landmarks:
            # Get bounding box of the hand
            x_min = int(min([landmark.x for landmark in hand_landmarks.
                landmark]) * frame_width)
            x_max = int(max([landmark.x for landmark in hand_landmarks.
                landmark]) * frame_width)
            y_min = int(min([landmark.y for landmark in hand_landmarks.
                landmark]) * frame_height)
```

```
            y_max = int(max([landmark.y for landmark in hand_landmarks.
                landmark]) * frame_height)

            touch_coords = (x_min + x_max) // 2, (y_min + y_max) // 2
            cv2.rectangle(top_frame, (x_min, y_min), (x_max, y_max), (0, 255,
                0), 2)
            cv2.putText(top_frame, f"{timestamp:.2f}s", (x_min, y_min - 10),
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
            cv2.putText(top_frame, f"Coords: {touch_coords}", (x_min, y_max +
                20), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

            if y_max > frame_height - 30:  # Adjust threshold based on your
                setup
                touch_detected = True

    # Process side view frame
    rgb_side = cv2.cvtColor(side_frame, cv2.COLOR_BGR2RGB)
    results_side = hands.process(rgb_side)

    if results_side.multi_hand_landmarks:
        for hand_landmarks in results_side.multi_hand_landmarks:
            x_min = int(min([landmark.x for landmark in hand_landmarks.
                landmark]) * side_frame_width)
            x_max = int(max([landmark.x for landmark in hand_landmarks.
                landmark]) * side_frame_width)
            y_min = int(min([landmark.y for landmark in hand_landmarks.
                landmark]) * side_frame_height)
            y_max = int(max([landmark.y for landmark in hand_landmarks.
                landmark]) * side_frame_height)

            hover_coords = (x_min + x_max) // 2, (y_min + y_max) // 2
            cv2.rectangle(side_frame, (x_min, y_min), (x_max, y_max), (0, 0,
                255), 2)
            cv2.putText(side_frame, f"{timestamp:.2f}s", (x_min, y_min - 10),
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
            cv2.putText(side_frame, f"Coords: {hover_coords}", (x_min, y_max +
                20), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)

            if y_max < side_frame_height - 30 and y_min > 30:  # Adjust
                threshold based on your setup
                hover_detected = True

    return top_frame, side_frame, touch_detected, hover_detected, touch_coords
        , hover_coords

# Release resources
top_view_video.release()
side_view_video.release()
out.release()
cv2.destroyAllWindows()
```

Listing 5.7: Python code for touch and hover detection

5.4 How to Run the Code for touch and hover detection

To run the provided Python file, follow these steps:

5.4.1 Prerequisites

1. **Install Python:** Ensure Python is installed on your system. You can download it from python.org.

2. **Install Required Libraries:** Use pip to install the necessary libraries:

```
pip install opencv-python mediapipe numpy
```

Listing 5.8: Command to install required Python libraries

5.4.2 Prepare Video Files

Ensure you have the 'topview.mp4' and 'sideview.mp4' video files in the same directory as the script. Alternatively, update the file paths in the script to match their location.

5.4.3 Save the Script

Save the provided code into a file, for example, 'hand$_d$*etection.py*'.

5.4.4 Running the Script

1. **Open a Terminal or Command Prompt:** Navigate to the directory where your script is saved.

2. **Run the Script:** Use the following command to execute the script:

```
python hand_detection.py
```

Listing 5.9: Command to execute the Python script

5.4.5 Debugging Common Issues

- **File Not Found:** If you encounter a `FileNotFoundError`, verify the paths for 'topview.mp4' and 'sideview.mp4'.

- **Video Compatibility:** Ensure the video files are in a compatible format (e.g., '.mp4' encoded with H.264). Convert them using tools like FFmpeg if necessary:

```
ffmpeg -i input.mp4 -vcodec libx264 output.mp4
```

Listing 5.10: Command to convert video format using FFmpeg

- **OpenCV Issues:** If OpenCV throws errors, verify its installation or upgrade to the latest version:

```
pip install --upgrade opencv-python
```

Listing 5.11: Command to upgrade OpenCV

- **MediaPipe Performance:** If detection is slow, ensure your system meets the minimum requirements for running MediaPipe efficiently (preferably a machine with GPU support).

5.4.6 Output

The processed video with combined frames and annotations will be saved as 'outputcombinedvideo.mp4' in the current directory. Ensure you monitor the terminal for any logs or errors during execution.