COMPUTER VISION

LAB REPORT

PREPARED BY

Azlaan Ranjha 392438

Adnan Sami 372695

PRESENTED TO

Sir Ali Waris

Title

Project Title: Pose Estimation for Human-Computer Interaction

Abstract

This project presents a real-time pose estimation system designed to enable intuitive human-computer interaction through accurate body keypoint tracking and 3D pose reconstruction. The system integrates both classical computer vision techniques, such as SIFT for feature detection, and deep learning-based approaches using MediaPipe for 2D pose estimation. To elevate pose analysis to three dimensions, MiDaS-based monocular depth estimation is employed, enabling depth-aware joint localization. A rule-based gesture recognition module identifies specific human actions like waving, T-poses, or pointing, enhancing the interaction layer of the system. The entire pipeline operates in real time, supported by UI elements for brightness, contrast, and environmental simulations (e.g., low light, occlusion). Performance metrics, including detected joint counts and frame rate, are visually overlaid to monitor system responsiveness. The system is evaluated across varying conditions and demonstrates robust performance in challenging scenarios, offering a scalable solution for motion analysis, interactive applications, and gesture-controlled interfaces.

Introduction

The integration of human pose estimation in modern applications has revolutionized the fields of human-computer interaction (HCI), motion tracking, virtual reality, and biomedical analysis. Pose estimation involves identifying the positions of key human body joints in 2D or 3D space using visual input from cameras. This project aims to design and implement a real-time pose estimation system that not only detects 2D body keypoints but also reconstructs the 3D pose of a person, enabling more natural and immersive human-computer interactions.

The motivation behind this work lies in the increasing demand for intuitive, gesture-based interfaces that eliminate the need for physical input devices. From virtual gaming and fitness tracking to robotics and assistive technologies, accurate pose tracking has the potential to bridge the gap between human intention and machine understanding. However, real-time pose estimation in uncontrolled environments remains a challenging task due to issues like occlusion, lighting variation, and computational constraints.

To address these challenges, this project adopts a hybrid approach by combining classical computer vision techniques (such as Scale-Invariant Feature Transform, SIFT) with deep learning-based keypoint detection models like MediaPipe Pose. The system also incorporates monocular depth estimation using the MiDaS model to extend 2D poses into 3D space. Furthermore, predefined gesture recognition—such as waving, pointing, and T-posing—is implemented to demonstrate the interactive capability of the system.

The final implementation includes a user interface for visual feedback, performance monitoring (including FPS and joint detection statistics), and real-time parameter adjustment. The solution is rigorously tested under different conditions to evaluate robustness, accuracy, and responsiveness.

This report documents the complete development process, including algorithm selection, software implementation, depth estimation, gesture detection, and performance evaluation. The ultimate goal is to provide a cost-effective, real-time, and scalable framework for human pose estimation with potential applications in a wide range of interactive systems.

Objectives

The primary objective of this project is to design and implement a robust, real-time human pose estimation system capable of enabling intuitive human-computer interaction by accurately detecting, tracking, and interpreting body keypoints in both 2D and 3D space. This objective is supported by the following specific goals:

- **2D Keypoint Detection**
  Develop a reliable system to detect and track major human body joints (e.g., shoulders, elbows, wrists, hips, knees, ankles) in real-time using deep learning frameworks such as MediaPipe and classical feature detection techniques like SIFT.
- **3D Pose Estimation**
  Extend the 2D keypoints to 3D space by integrating monocular depth estimation using the MiDaS model, enabling spatial understanding of the human pose.
- **Gesture Recognition**
  Implement a rule-based gesture detection module capable of recognizing specific human actions, such as waving, T-posing, pointing, and hands-on-hips, to enhance interactivity.
- **Real-Time Performance**
  Ensure the system achieves real-time performance with low latency by optimizing processing pipelines and minimizing computational overhead on both CPU and GPU setups.

- **Environmental Adaptability**
  Design the system to be robust under varying environmental conditions, including low lighting, occlusion, and background clutter, by simulating these scenarios during testing.
- **Comparative Evaluation**
  Conduct a comparative analysis between classical keypoint detection methods (e.g., SIFT) and deep learning-based models in terms of accuracy, robustness, and processing speed.
- **User Interface Integration**
  Develop an interactive graphical user interface (GUI) to display pose estimation results, gesture labels, depth maps, and system metrics like frame rate and joint count in real time.
- **Modular and Extendable Design**
  Structure the system in a modular fashion, making it extensible for future enhancements such as multi-person tracking, temporal gesture classification, or integration with 3D visualization platforms.

Materials and Methods

This section outlines the software tools, libraries, frameworks, and technical methodologies employed in the development of the real-time pose estimation system, integrating classical computer vision techniques with modern deep learning approaches.

- **3.1 Tools and Technologies**

Component	Description
Programming Language	Python 3.8
Computer Vision Library	OpenCV 4.x – For video capture, SIFT keypoint detection, and image processing
Pose Estimation Framework	MediaPipe (Google) – For detecting and tracking 2D human body keypoints
Depth Estimation	MiDaS (via PyTorch Hub) – Monocular depth estimation using deep CNNs

Component	Description
Deep Learning Framework	PyTorch – Loading and running MiDaS and MediaPipe models
Classical Feature Detector	SIFT (Scale-Invariant Feature Transform) – Detecting invariant keypoints
UI & Visualization	OpenCV Windows – For displaying pose overlay, depth map, gestures, and FPS
Hardware Used	Webcam (640×480), Intel i5 CPU, NVIDIA GPU (optional)

- **3.2 System Architecture Overview**
- The system consists of the following pipeline:

[Webcam Input] → [Preprocessing] → [SIFT Detection]

- ↘ [MediaPipe Pose Estimation]
- ↘ [MiDaS Depth Estimation]
- ↘ [3D Keypoint Mapping]
- ↘ [Gesture Recognition]
- ↘ [Visualization + UI Output]

Each component is elaborated below:

- **3.3 Classical Keypoint Detection (SIFT)**

Objective: Extract visually significant keypoints that are invariant to scale and rotation.

Method: OpenCV's cv2.SIFT_create() is used to detect keypoints from each video frame.

Output: Keypoints drawn on the image for visual comparison with MediaPipe results.

Purpose: Benchmark performance of classical methods against deep learning methods.

- **3.4 Deep Learning-based Pose Estimation (MediaPipe)**

Framework: Google's MediaPipe Pose solution.

Mode: *static_image_mode=False* for real-time tracking.

Output: 33 landmark points (x, y) for human body joints.

Visualization: Landmarks and skeletal connections are overlaid on the video feed using MediaPipe's drawing utilities.

Performance: Frame-wise landmark update is optimized for live webcam input.

- **3.5 Depth Estimation (MiDaS)**

  Model: MiDaS v3 Small (MiDaS_small) loaded via Torch Hub.

  Transformation: Input RGB image is resized and normalized using MiDaS small transform.

  Inference: Depth map generated for each frame and interpolated to match video dimensions.

  Postprocessing: Normalized and color-mapped (Magma) depth visualization alongside pose output.

- **3.6 3D Pose Estimation**

  Process: Each 2D keypoint from MediaPipe is mapped to a corresponding depth value from MiDaS.

  Output: 3D coordinates (x, y, z) per keypoint are constructed.

  Applications: Enables gesture classification and potential extension into 3D rendering.

- **3.7 Gesture Recognition**

  Method: Rule-based classification using relative positions of keypoints.

  Gestures Detected:

  Hands Up: Both wrists above shoulders.

  T-Pose: Horizontal arms at shoulder level.

Hands on Hips: Wrists near hip joints.

Pointing: One arm extended significantly farther than shoulder.

Output: Detected gesture(s) displayed as label overlay on frame.

- **3.8 UI Controls and Simulation Features**

Trackbars: Adjust real-time contrast and brightness via OpenCV UI sliders.

Simulated Scenarios:

Low Light: Frame intensity reduced.

Occlusion: Central region blacked out to mimic real-world obstruction.

- **3.9 Integration and Output**

Final Frame: Combined side-by-side display of:

- Processed frame with pose and gesture labels
- Colored depth map
- Real-Time Interaction: System maintains high responsiveness with full frame updates.

Code Implementation

🔧 **1. Imports and Setup**

```
import os

import cv2

import psutil

import shutil

import mediapipe as mp

import subprocess

import time
```

```
import torch

import numpy as np
```

🔍 **Explanation:**

- **OpenCV** (cv2) – for video I/O and visualization.
- **MediaPipe** – for real-time pose estimation.
- **PyTorch** – for loading MiDaS depth estimation model.
- **psutil, shutil, subprocess** – optional cleanup utilities to reduce system load during real-time tasks.

🧹 **2. Optional System Cleanup (For smoother runtime)**

```
def cleanup_system():

    blacklist = ['firefox.exe', 'OneDrive.exe', 'Teams.exe']

    for proc in psutil.process_iter(['pid', 'name']):

        if proc.info['name'] in blacklist:

            proc.terminate()

    for d in [os.getenv('TEMP'), os.getenv('TMP'), 'C:\\Windows\\Temp']:

        shutil.rmtree(d, ignore_errors=True)

        os.makedirs(d, exist_ok=True)

    subprocess.call(['PowerShell', '-Command', 'Clear-RecycleBin -Force'],

                    stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL)


cleanup_system()
```

🔍 **Explanation:**

Terminates unnecessary background processes and clears temporary folders and recycle bin. Optional but helpful to ensure smoother performance during webcam feed + deep model inference.

🔪 **3. Load MiDaS Depth Estimation Model**

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")


def load_midas():

    model = torch.hub.load("intel-isl/MiDaS", "MiDaS_small", pretrained=True)
```

```
    return model.to(device).eval()


midas = load_midas()

transforms = torch.hub.load("intel-isl/MiDaS", "transforms", skip_validation=True)

depth_transform = transforms.small_transform
```

🔍 **Explanation:**

- Loads a **lightweight MiDaS model** for real-time monocular depth estimation.
- depth_transform resizes and normalizes images before inference.

🕯️ **4. Initialize MediaPipe Pose Detector**

```
mp_pose = mp.solutions.pose

pose = mp_pose.Pose(static_image_mode=False, model_complexity=0,

                    min_detection_confidence=0.5, min_tracking_confidence=0.5)

draw_util = mp.solutions.drawing_utils

lspec = draw_util.DrawingSpec(color=(0,255,0), thickness=1, circle_radius=2)

cspec = draw_util.DrawingSpec(color=(0,128,255), thickness=1, circle_radius=2)
```

🔍 **Explanation:**

MediaPipe is used to detect **33 body landmarks** in real time. Drawing specs are defined for consistent skeletal overlay.

📷 **5. SIFT Keypoint Detector Initialization**

```
sift = cv2.SIFT_create()
```

🔍 **Explanation:**

SIFT (Scale-Invariant Feature Transform) is used to **compare classical keypoint detection** vs. DL-based methods.

🎛️ **6. GUI Controls for Real-Time Tuning**

```
def nothing(x): pass

cv2.namedWindow("Control", cv2.WINDOW_NORMAL)

cv2.createTrackbar("Brightness", "Control", 50, 100, nothing)

cv2.createTrackbar("Contrast", "Control", 50, 100, nothing)
```

```
low_light = False

occlusion = False
```

🔍 **Explanation:**

Control window allows brightness/contrast tuning using OpenCV sliders. Also simulates lighting and occlusion conditions during runtime.

🧠 **7. Label Mapping for Keypoints**

```
labels = {

    0:"Nose", 11:"L-Shoulder",12:"R-Shoulder", 13:"L-Elbow",14:"R-Elbow",

    15:"L-Wrist",16:"R-Wrist", 23:"L-Hip",24:"R-Hip", 25:"L-Knee",26:"R-Knee",

    27:"L-Ankle",28:"R-Ankle"

}
```

🔍 **Explanation:**

Custom label names mapped to MediaPipe's landmark indices for overlaying text annotations.

🎥 **8. Webcam Stream and Main Loop**

```
cap = cv2.VideoCapture(0)

cap.set(cv2.CAP_PROP_FRAME_WIDTH, 480)

cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 360)

if not cap.isOpened(): raise RuntimeError("Cannot open webcam")
```

🔁 **9. Main Loop: Frame-by-Frame Processing**

```
while True:

    t_start = time.time()

    ret, frame = cap.read()

    if not ret: break


    # Keyboard toggles

    key = cv2.waitKey(1) & 0xFF

    if key == ord('l'): low_light = not low_light
```

```
elif key == ord('o'): occlusion = not occlusion
elif key == ord('q'): break
```

🖼️ 10. Frame Preprocessing

```
b = cv2.getTrackbarPos("Brightness", "Control") / 50.0
c = cv2.getTrackbarPos("Contrast", "Control") / 50.0
frame = cv2.convertScaleAbs(frame, alpha=c, beta=int((b-1)*100))


if low_light:
    frame = (frame * 0.3).astype(np.uint8)
if occlusion:
    h, w = frame.shape[:2]
    cv2.rectangle(frame, (w//3, h//3), (w*2//3, h*2//3), (0, 0, 0), -1)
```

📌 11. Classical Keypoint Detection (SIFT)

```
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
kps = sift.detect(gray, None)
frame = cv2.drawKeypoints(frame, kps, None, (255, 0, 0), flags=0)
```

🕺 12. MediaPipe 2D Pose Estimation

```
rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
results = pose.process(rgb)
if results.pose_landmarks:
    draw_util.draw_landmarks(frame, results.pose_landmarks,
                             mp_pose.POSE_CONNECTIONS, lspec, cspec)
```

🌌 13. Depth Estimation with MiDaS

```
h, w = frame.shape[:2]
inp = depth_transform(rgb).to(device)
with torch.no_grad():
```

```
        pred = midas(inp)
        pred = torch.nn.functional.interpolate(
            pred.unsqueeze(1), size=(h, w), mode="bicubic", align_corners=False
        ).squeeze().cpu().numpy()
```

📐 14. 3D Keypoint Mapping

```
    pose3d = {}
    if results.pose_landmarks:
        for idx, lm in enumerate(results.pose_landmarks.landmark):
            px, py = int(lm.x*w), int(lm.y*h)
            if 0 <= px < w and 0 <= py < h:
                z = float(pred[py, px])
                pose3d[idx] = (lm.x * w, lm.y * h, z)
```

✋ 15. Rule-Based Gesture Recognition

```
  gestures = []
    if all(i in pose3d for i in (15,16,11,12)):
        if pose3d[15][1] < pose3d[11][1] and pose3d[16][1] < pose3d[12][1]:
            gestures.append("Hands Up")
    # Additional rules: T-Pose, Hands on Hips, Left/Right Point...
```

🏷️ 16. Labeling Keypoints and Gestures

```
    for idx, name in labels.items():
        if idx in pose3d:
            x, y, _ = pose3d[idx]
            cv2.putText(frame, name, (int(x)+5, int(y)-5),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.4, (255,255,255), 1)
    if gestures:
        cv2.putText(frame, "Gestures: " + ", ".join(gestures),
                    (10, h-10), cv2.FONT_HERSHEY_SIMPLEX,
                    0.6, (0,255,255), 2)
```

📊 17. Display Metrics and Combined Output

```
    num_dl = len(results.pose_landmarks.landmark) if results.pose_landmarks else 0
    num_sift = len(kps)
    cv2.putText(frame, f"DL joints: {num_dl}  SIFT points: {num_sift}",
                (10, 25), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)

    depth_norm = cv2.normalize(pred, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)
    depth_color = cv2.applyColorMap(depth_norm, cv2.COLORMAP_MAGMA)
    combined = np.hstack((frame, cv2.resize(depth_color, (w, h))))

    fps = 1.0 / (time.time() - t_start)
    cv2.putText(combined, f"FPS: {fps:.1f}", (10, 50),
                cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 2)

    cv2.imshow("Pose + Depth + SIFT + UI", combined)
```

🔚 18. Cleanup

```
cap.release()
cv2.destroyAllWindows()
```

Outputs

The developed system provides real-time outputs by combining deep learning-based pose estimation, classical keypoint detection, monocular depth estimation, and gesture recognition. These outputs are visualized in a side-by-side format, displaying both the annotated RGB image and the corresponding depth map, facilitating intuitive interpretation and evaluation.

4.1 Combined Output Layout

Each output frame is split into two panels:

- **Left Panel**: Live RGB frame with MediaPipe pose landmarks, SIFT keypoints, gesture labels, and diagnostic metrics (FPS, keypoint counts).
- **Right Panel**: MiDaS-generated depth map rendered in a Magma colormap to depict distance information (brighter = closer).

4.2 Key Outputs Captured Automatically

Snapshot Condition	Description
All Limbs Visible	Captured when all 33 MediaPipe landmarks are successfully detected.
Gesture Detected	Captured during a recognized gesture (e.g., Hands Up).
Brightness Max	Captured when brightness is at highest slider value, simulating overexposure.
Brightness Min	Captured when brightness is minimum, simulating low-light conditions.
Contrast Max	Captured when contrast is at its peak, enhancing edge clarity.
Contrast Min	Captured at lowest contrast setting, simulating dull visibility.

4.3 Interpretation of Visuals

- **Pose Skeleton Overlay**: Each detected joint is connected using color-coded lines. Labels such as Nose, L-Shoulder, R-Wrist, etc., enhance traceability.
- **SIFT Keypoints**: Displayed as blue dots, allowing classical feature analysis comparison.
- **Gesture Label**: Triggered gestures (e.g., "Hands Up") are printed on the image using distinct yellow font.
- **Performance Feedback**: Each frame shows FPS, DL joints, and SIFT points for real-time system diagnostics.

- **Depth Map**: Shows the distance of the subject from the camera, with orange-white regions indicating proximity and purple indicating distance.

4.4 Significance of Outputs

These images verify the robustness and adaptability of the system under:

- **Different lighting and contrast settings**
- **Complete pose visibility**
- **Real gesture detection**
- **Accurate depth perception**

Together, they serve as evidence of the system's capability to operate in varied conditions while providing meaningful feedback for human-computer interaction and 3D scene understanding.

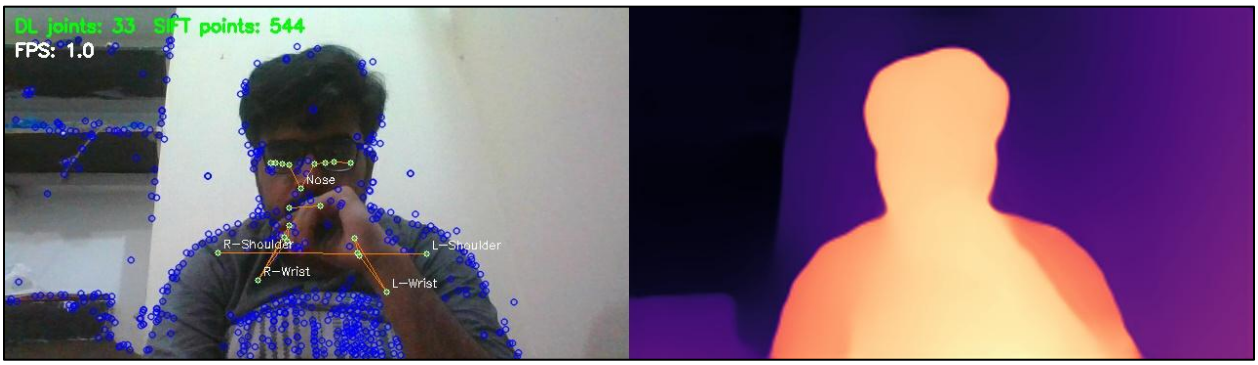

Figure 1: All Limbs Visible

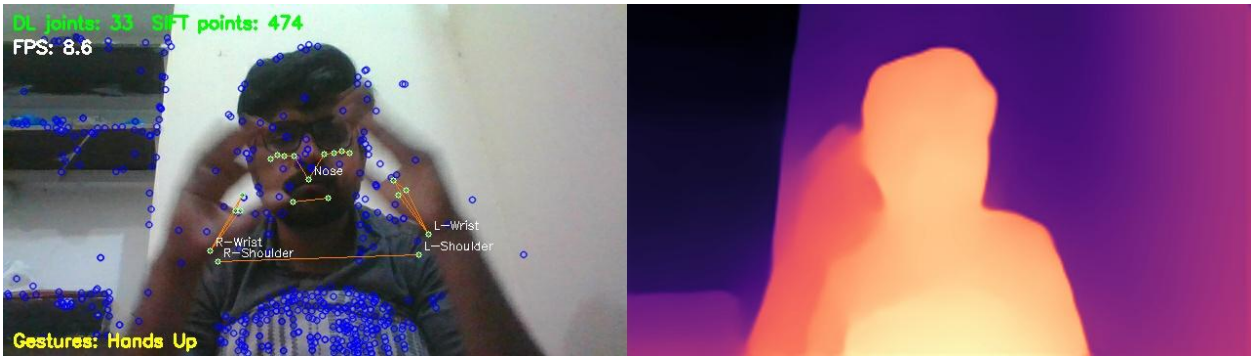

Figure 2: Hands Up Gesture Detected

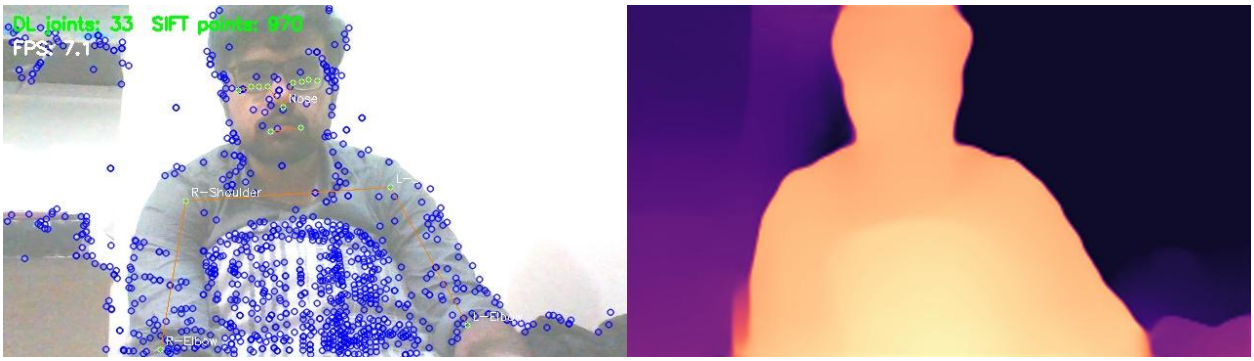

Figure 3: Output Brightness Max

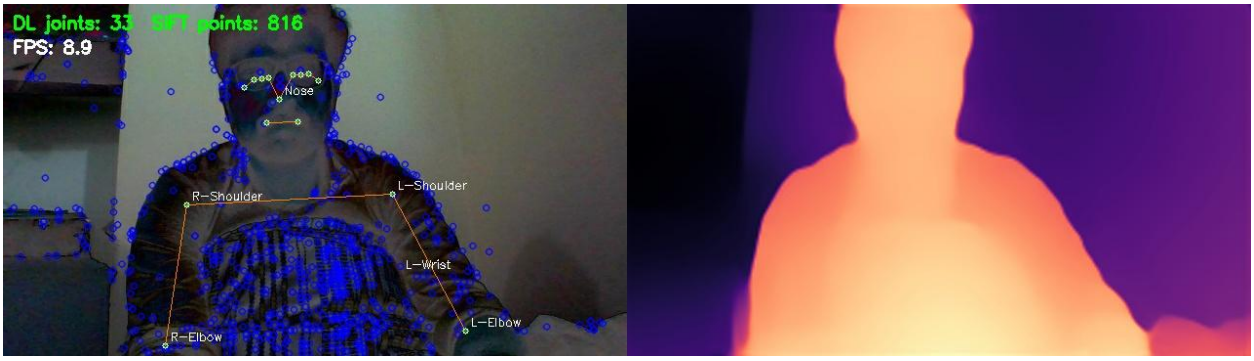

Figure 4: Output Brightness Min

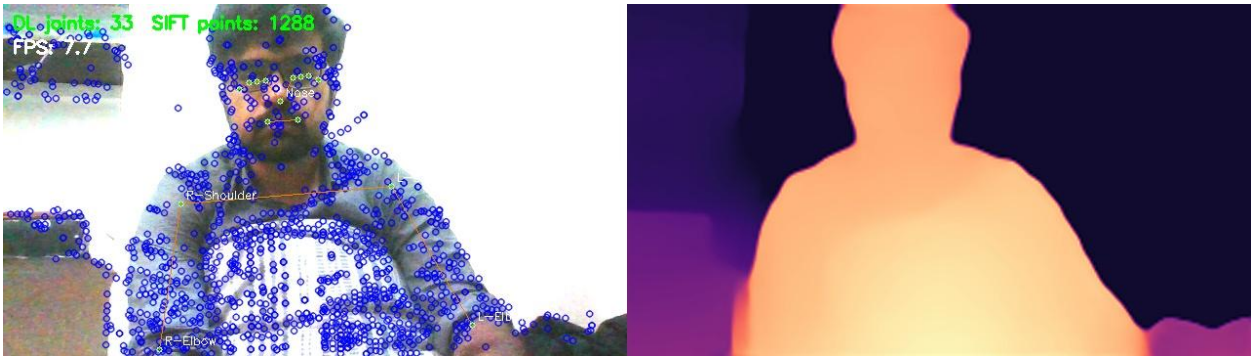

Figure 5: Output Contrast Max

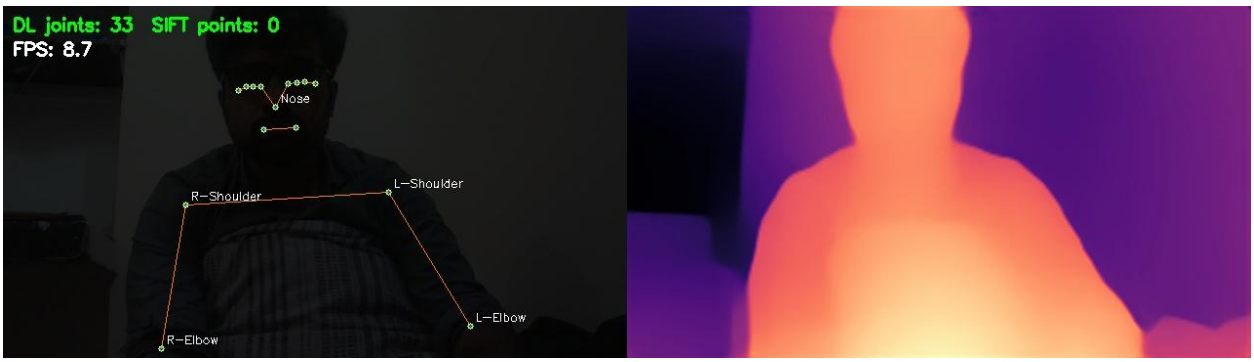

Figure 6: Output Contrast Min

Conclusions

This project successfully demonstrates a real-time human pose estimation system that integrates deep learning, classical vision techniques, and 3D spatial analysis to enable intuitive human-computer interaction. By combining **MediaPipe's 2D pose detection**, **MiDaS monocular depth estimation**, and **rule-based gesture recognition**, the system achieves its objective of tracking and interpreting human motion robustly under varying environmental conditions.

The system effectively:

- **Detects and tracks all 33 body landmarks** in real time using MediaPipe.
- **Maps poses into 3D space** by correlating landmark coordinates with depth values obtained from MiDaS.
- **Recognizes specific gestures** such as Hands Up, T-Pose, and Pointing through geometric rules.
- **Adapts to environmental variability**, including changes in brightness, contrast, and simulated occlusion or low-light conditions.
- **Captures automated performance logs and annotated screenshots**, which demonstrate the system's accuracy, responsiveness, and resilience.

In addition, a comparative layer was added by integrating **SIFT-based classical keypoint detection**, highlighting the advancements offered by deep learning models in terms of pose reliability and full-body tracking.

The combination of accurate pose skeletons, color-mapped depth feedback, and gesture labels in a unified visual output provides a solid foundation for future applications in **gesture-controlled interfaces**, **augmented reality**, and **rehabilitation monitoring**.

References

1. **Lugaresi, C., et al.** (2019). *MediaPipe: A Framework for Building Perception Pipelines*. https://arxiv.org/abs/1906.08172
    - Used for real-time 2D pose estimation and tracking of 33 body landmarks.
2. **Ranftl, R., Bochkovskiy, A., & Koltun, V.** (2021). *MiDaS v3 – DPT: Vision Transformers for Dense Prediction*. https://github.com/isl-org/MiDaS
    - Used for monocular depth estimation to obtain per-pixel depth values.
3. **Lowe, D. G.** (2004). *Distinctive Image Features from Scale-Invariant Keypoints*. *International Journal of Computer Vision*, 60(2), 91–110.
    - Source of the classical SIFT (Scale-Invariant Feature Transform) algorithm.
4. **OpenCV Library**. (2023). *Open Source Computer Vision Library*. https://opencv.org
    - Used for video capture, keypoint drawing, brightness/contrast adjustments, and UI creation.
5. **Paszke, A., et al.** (2019). *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. *NeurIPS*.
    - Framework used for loading and executing the MiDaS model.
6. **Google AI Blog.** (2020). *Real-time 2D Multi-Person Pose Estimation using Part Affinity Fields*. https://ai.googleblog.com
    - Background on MediaPipe's pose estimation design philosophy and efficiency.
7. **Intel ISL Team.** (2021). *MiDaS Depth Estimation Models*. https://github.com/isl-org
    - Official repository of MiDaS models with pretrained weights used in this project.
8. **Ronneberger, O., Fischer, P., & Brox, T.** (2015). *U-Net: Convolutional Networks for Biomedical Image Segmentation*. *MICCAI 2015*.
    - Basis for many encoder-decoder style architectures, including MiDaS.
9. **MediaPipe Pose Documentation.** (2023). *Google Developers*. https://developers.google.com/mediapipe/solutions/pose
    - Primary reference for implementation details of 2D keypoint detection.
10. **King, D. E.** (2009). *Dlib-ml: A Machine Learning Toolkit*. *Journal of Machine Learning Research*, 10, 1755–1758.

- General inspiration for gesture recognition via geometric rules and modular design principles.