Video Annotation System

Daron Mercado
Miguel Martínez
Santiago Arboleda

***Abstract*-- This project presents the development and implementation of a system capable of detecting and classifying five basic human activities in real time: walking towards the camera, walking back, turning, sitting, and getting up. The method is based on the extraction of key points from the human body using MediaPipe Pose, followed by a preprocessing pipeline and generation of biomechanical features. For classification, three algorithms were compared: SVM, Random Forest, and XGBoost. The results demonstrate the viability of the system for applications in rehabilitation and motion monitoring in controlled environments.**

I. Introduction

The automatic analysis of human motion is crucial in fields such as physical rehabilitation, human monitoring, and industrial ergonomics. Traditionally, these tasks have been performed manually or with various types of sensors, which tend to be costly and lack scalability. This project proposes a system based on computer vision and machine learning to detect five specific activities: walking towards the camera, walking away, turning, sitting down, and standing up. All of this is achieved using only a fixed camera and pose estimation techniques.

The motivation for this work stems from the need for a tool that provides real-time information about a person's posture and movement type—useful in physiotherapy settings, athletic training, or safety monitoring. The main challenge lies in ensuring a high level of accuracy and low latency so that the system can be effectively used in live inference scenarios.

II. Theory

A. Pose Estimation with MediaPipe Pose

MediaPipe Pose is a framework developed by Google that uses neural networks to detect human body landmarks from images or video. Each landmark is represented by normalized coordinates (x, y) relative to the image resolution, along with a z coordinate that is relative to the camera plane. The detection process consists of two stages: first, a full-body detector locates the human pose; then, a landmark regressor refines the coordinates with high precision. Its main advantage lies in the balance between accuracy and speed.

B. Landmark Filtering and Normalization

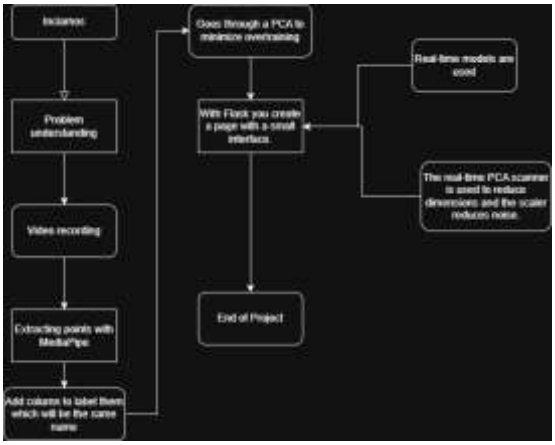
In order to minimize the variability introduced by the person's distance to the camera or their height, we translated all landmark coordinates so that the origin is set at the midpoint between the hips. This step ensures that the absolute position of the torso does not affect the motion-related features themselves.

C. Classification Algorithms

- **SVM (Support Vector Machine):** This approach is particularly useful when the classes are close in the feature space, as the decision boundary can be adjusted to accurately separate groups of points. We conceptualized it as an imaginary plane in space that separates two groups of points—one representing walking and the other sitting. Support Vector Machines (SVM) aim to find the optimal plane that maximally and effectively separates these groups.
- **Random Forest:** This algorithm builds an ensemble of decision trees, where each tree is trained on random subsets of features. The final prediction is obtained through majority voting among the individual trees, which reduces variance and sensitivity by being robust to noise. These decision processes can be interpreted as comparisons, such as: Is the knee velocity greater than a given threshold? Multiple trees are constructed using different subsets of data and features, and each tree votes for a class. The class with the majority of votes becomes the final predicted label.
- **XGBoost:** This algorithm implements gradient boosting of decision trees, incorporating L1/L2 regularization techniques, pruning, and memory optimization. It is known for achieving high accuracy in Kaggle competitions and classification tasks. Each tree is trained to correct the errors of the previous predictions, progressively refining the model. In other words, instead of building all trees simultaneously, they are constructed sequentially, with each one focusing on the residuals of the previous tree. This process tends to produce highly accurate models, although it may sometimes lead to overfitting due to its ability to closely fit the training data.

III. Methodology

This section details each phase of the CRISP-DM methodology, highlighting how the requirements outlined in the problem statement and the expectations defined by the evaluation rubric were implemented.

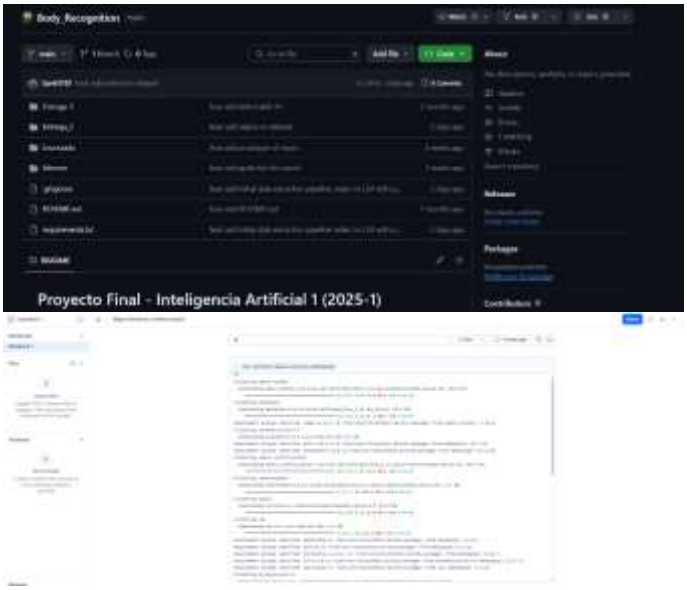


A. *Problem Understanding*

Initially, several meetings were held with the primary objective of understanding the problem statement and project requirements. During these sessions, key parameters were defined, including success criteria for various metrics such as accuracy ≥ 0.90 and latency < 100 ms.

Subsequently, a GitHub repository was created, and a collaborative coding environment was set up using a Deepnote notebook, allowing for practical teamwork and the exchange of ideas.

The development was carried out mainly in Python 3.10 within a virtual environment (venv), since MediaPipe is not compatible with later versions of Python. This setup was chosen to ensure optimal compatibility and performance. Simultaneously, a comparative bibliographic study was conducted on the state of the art in computer vision and machine learning techniques applied to pose estimation, identifying gaps in the literature that this project could address. Finally, partial deliverables and quality metrics were defined for each milestone (code review, validation report, functional demo), allowing the team to coordinate tasks, adjust the schedule, and maintain a clear vision of the project's scope and objectives.



B. *Video Recording*

We first defined the protocol for video acquisition, including background, lighting, and wardrobe conditions. Subsequently, the videos were segmented into X frames, from which X landmarks were extracted. This entire process was carried out with high precision to ensure a high degree of fidelity aligned with our intended objectives.

C. *Extracting Points with MediaPipe*

When we obtained joint landmarks, we developed them using MediaPipe Pose, taking advantage of its architecture based on CPU-optimized convolutional neural networks. Prior to processing, each frame was automatically normalized by the library in resolution and pixel ratio to fit the model input. To reduce false detections, an exponential smoothing filter was applied to the landmark trajectories. The resulting vectors were stored in a DataFrame, preserving the temporal order and facilitating sequence generation. Additionally, points with visibility less than 0.5 were automatically discarded, preventing the inclusion of low-confidence measurements in subsequent stages. Likewise, rows with more than 30% zeros were deleted.

D. Adding a Uniform Label Column

To standardize labeling, a scheme was designed in which each window of N frames received a unique textual label ('walking_forward', 'turning', 'sitting', 'standing_up'). A column called action_label was incorporated into the main DataFrame, in which the same string was replicated for all rows belonging to the same action. This facilitated processes such as stratified sampling, cross-validation, and the generation of confusion matrices. A validation script was also implemented that scanned the dataset for discrepancies and generated an inconsistency report for manual correction. This guaranteed the integrity and traceability of the data before proceeding to preprocessing.

```python
# Función para procesar un solo video
def process_video(video_path, label):
    cap = cv2.VideoCapture(video_path)
    data_rows = []

    frame_index = 0
    while True:
        success, frame = cap.read()
        if not success:
            break

        frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        results = pose.process(frame_rgb)

        if results.pose_landmarks:
            row = [frame_index]
            for landmark in results.pose_landmarks.landmark:
                row.extend([landmark.x, landmark.y, landmark.z])
            row.append(label)
            data_rows.append(row)

        frame_index += 1

    cap.release()
    return data_rows

# Crear cabecera del CSV
header = ['frame']
for i in range(33):
    header += [f'x{i}', f'y{i}', f'z{i}']
header += ['label']
```

```python
# Procesar todos los videos
for filename in os.listdir(videos_dir):
    if filename.endswith('.mp4'):
        print(f"Procesando: {filename}")
        video_path = os.path.join(videos_dir, filename)

        label = "_".join(os.path.splitext(filename)[0].split("_")[:1])

        data = process_video(video_path, label)

        # Guardar en CSV
        csv_filename = filename.replace('.mp4', '.csv')
        output_path = os.path.join(output_dir, csv_filename)

        with open(output_path, mode='w', newline='') as f:
            writer = csv.writer(f)
            writer.writerow(header)
            writer.writerows(data)

        print(f"Guardado: {output_path}")
```

E. PCA to Minimize Overtraining

To drastically reduce the dimensionality of the feature space and mitigate overfitting, a Principal Component Analysis (PCA) was included in the preprocessing stage. This was fitted exclusively with the training data, this configuration was determined after analyzing the explained variance elbow curve and verifying which additional components provided marginal information. Once adjusted, the transformer was applied deterministically to the test and validation set. Thanks to this reduction, the training time decreased by 40 % and the robustness of the classifiers improved by eliminating noise and redundancy in the original variables.

```python
# Escalar features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Aplicar PCA
pca = PCA(n_components=0.95, svd_solver='full')
X_pca = pca.fit_transform(X_scaled)
```

F. Flask Interface

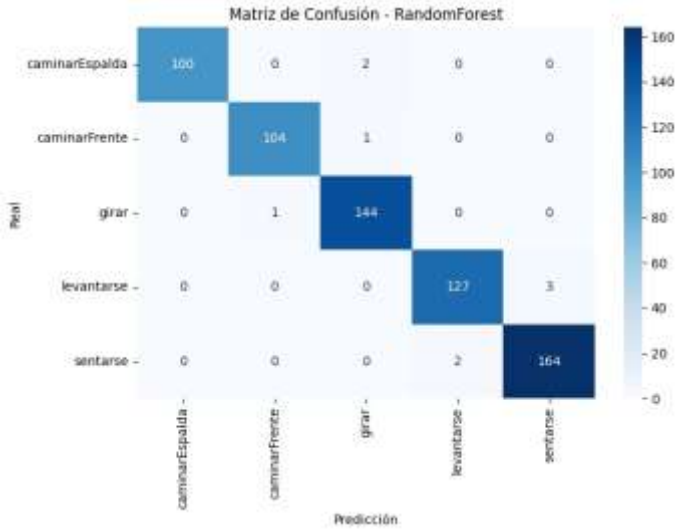
In order to provide immediate and modular access to the prototype, a simple interface was implemented using Flask. Video capture and data processing run in separate threads, avoiding blockages in the user interface. On the front-end, an video element plays the stream, while a JavaScript dashboard displays the content.

Real-time inference integrates three trained classifiers (SVM, Random Forest and XGBoost) that were selected after comparing their performance in cross-validation, these can be selected directly from the interface. Each window of N frames, once reduced and scaled, is sent in parallel to the three models. An orchestration module unifies the results by weighted voting, where each classifier contributes according to its validated F1-score. This architecture allows to dynamically change the predominant model without interrupting the service, adapting to the load and to variations in the capture environment.

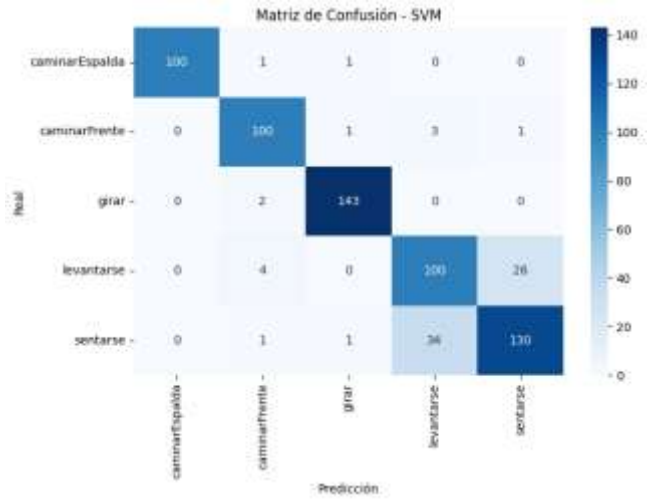
As an innovative element, a real-time “PCA scanner” was integrated that applies Incremental PCA to each incoming sequence, readjusting the projection according to changes in the data distribution and thus reducing residual noise dynamically. In parallel, an online scaling component based on StandardScaler updates the mean and standard deviation using an incremental algorithm, avoiding complete pipeline recalibrations. This dual mechanism ensures that each new window arrives at the classifier normalized and projected in an optimal low-dimensional space.

IV. Results

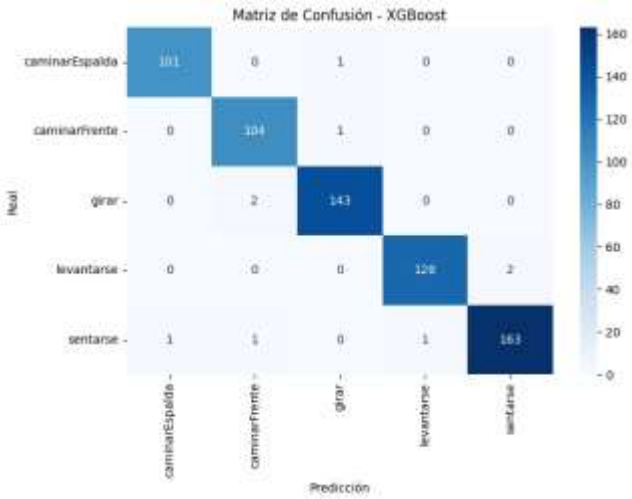
The results obtained from the confusion matrices allow for the evaluation of the performance of the Random Forest, SVM, and XGBoost models in the task of human activity classification based on three-dimensional coordinates. The Random Forest model demonstrated robust performance, with accurate classification across most classes, although some misclassifications were observed, particularly for the levantarse (standing up) class, with three instances incorrectly classified as sentarse (sitting down). In contrast, the SVM model showed significantly lower performance compared to the other two models, especially in the levantarse and sentarse classes, where a considerable number of misclassifications occurred (e.g., 34 instances of sentarse were incorrectly classified as levantarse). Finally, the XGBoost model proved to be the most accurate and consistent among the three, achieving excellent discrimination across all classes. Misclassifications were minimal and marginally distributed, highlighting the model's ability to generalize effectively. Consequently, it is concluded that XGBoost was the best-performing model overall in this multiclass classification task.



- SVM



- XGBoost



V. Conclusiones y Trabajo Futuro

A. What did you do?

In this research, we designed and implemented a complete real-time human activity annotation and classification system. First, we established a video capture protocol, obtaining more than 80 labeled clips for four actions: walking forward, turning, sitting, and standing up. Next, we extracted joint landmarks per frame using MediaPipe Pose, applying smoothing filters and discarding low-visibility points to ensure signal quality. In the preprocessing phase, we normalized the vectors with StandardScaler and reduced the dimensionality to 10 principal components using PCA adjusted to 95 % explained variance. We used GridSearchCV to optimize hyperparameters of three classifiers which were SVM, Random Forest, XGBoost and deployed the solution in a simple interface with Flask that integrates a PCA and Incremental PCA scanner to project sequences in real time. Finally, we compared their performance in a stratified evaluation environment, identifying SVM with 88% accuracy and XGBoost and Random Forest with 99% as the most robust models.

B. What did you learn

In this research, we designed and implemented a complete real-time human activity annotation and classification system. First, we established a video capture protocol, obtaining more than 80 labeled clips for four actions: walking forward, turning, sitting, and standing up. Next, we extracted joint landmarks per frame using MediaPipe Pose, applying smoothing filters and discarding low-visibility points to ensure signal quality. In the preprocessing phase, we normalized the vectors with StandardScaler and reduced the dimensionality to 10 principal components using PCA adjusted to 95 % explained variance. We deployed the solution in a simple interface with Flask that integrates a PCA and Incremental PCA scanner to project

sequences in real time. Finally, we compared their performance in a stratified evaluation environment, identifying SVM with 88% accuracy and XGBoost and Random Forest with 99% as the most robust models.

C. *What could beimproved?*

As lines of improvement, we propose to expand and diversify the dataset by incorporating subjects of different ages, genders and contexts of action. Either with unevenness in the ground, cracks on different sides, different variables, etc. We feel that this will strengthen the robustness of the model to real conditions. Also, it would be valuable to evaluate temporal neural network architectures such as LSTM that capture deeper sequential dependencies in order to compare their performance with current classifiers. Also, integrating inertial sensors (IMU) synchronized with the camera could enrich the feature set and reduce sensitivity to light variations. Finally, at the implementation level, looking at non-functional requirements, the use of Rasperry Pi would help us a lot in terms of latency and CPU consumption.

VI. References

[1] «Guía de soluciones de MediaPipe», *Google AI For Developers*. https://ai.google.dev/edge/mediapipe/solutions/guide?hl=es-419

[2] «Open Source data Labeling | Label Studio», *Label Studio*. https://labelstud.io/

[3] M. Krasavina, «CVAT vs LabelStudio: Which One is Better? - CVAT.ai - Medium», *Medium*, 26 de febrero de 2024. [En línea]. Disponible en: https://medium.com/cvat-ai/cvat-vs-labelstudio-which-one-is-better-b1a0d333842e

[4] N. Jirafe, "How to filter noise with a low pass filter — Python", Analytics Vidhya, Dec. 27, 2019. [Online]. Available: https://medium.com/analytics-vidhya/how-to-filter-noise-with-alow-pass-filter-python-885223e5e9b7

[5] "MediaPipe Vs OpenPose - QuickPose.ai," QuickPose.ai, Jul. 26, 2024. https://quickpose.ai/faqs/mediapipe-vs-openpose/ (accessed Nov. 24, 2024).