AutoRep: A Technical Report on the Development of a Real-Time Computer Vision Exercise Counter

Tyler Venner

September 6, 2025

Abstract

Manual counting of exercise repetitions during workouts is often inconsistent and distracting, leading to suboptimal training. This report details the development of Kinetic Count, a real-time computer vision application designed to automate this process using a standard webcam. Two distinct technical approaches were implemented and evaluated. The first, a proof-of-concept model, utilized simple frame differencing to quantify motion and a state machine to count high-intensity spikes. While computationally efficient, this method proved sensitive to environmental noise and lacked the contextual understanding to reliably count complex, multi-stage exercises. The second, more robust approach employed a pre-trained deep learning model via Google's MediaPipe Pose library to perform human pose estimation. By tracking the coordinates of key body landmarks, a posture-based state machine was developed, enabling the system to accurately count full-range-of-motion repetitions. The final implementation demonstrates that pose estimation provides a significantly more accurate and reliable solution for exercise counting. The system is built entirely in Python, leveraging the OpenCV and MediaPipe libraries.

Contents

1 Introduction

In fitness and athletic training, the accurate tracking of exercise repetitions is fundamental to measuring progress and ensuring adherence to a structured program. However, manual counting during high-intensity workouts presents a distracting cognitive load. The need to maintain focus on proper form, breathing, and exertion often leads to miscounting or losing track entirely. This distraction can detract from the quality of the workout and result in inaccurate performance data, hindering effective training analysis.

This report documents the design and development of AutoRep, a software application that addresses this problem by providing automated, real-time repetition counting through computer vision. The primary objective was to create a hands-free system that requires no specialized hardware beyond a standard webcam, making it accessible to a wide audience.

The project was approached as an iterative engineering challenge, beginning with a simple proof-of-concept and evolving into a sophisticated final product. This report details two primary methodologies:

1. **Motion Detection:** An initial, lightweight approach based on analyzing the magnitude of pixel changes between video frames. This method served to establish a baseline, validate the core software architecture, and explore the inherent limitations of a context-free counting system.

2. **Human Pose Estimation:** A more advanced implementation leveraging a pre-trained deep learning model to identify and track the geometry of the human body. This approach was developed to overcome the shortcomings of the motion-based system by using posture based counting logic.

This document will outline the system architecture, detail the implementation of both algorithms, analyze their respective performance and weaknesses, and conclude that the pose estimation approach provides a vastly more robust and reliable solution for real-world use.

2 System Architecture and Design

To facilitate the iterative development and comparison of two fundamentally different counting methodologies, a modular, object-oriented architecture was designed. The core principle guiding this design was Separation of Concerns, ensuring that each component of the application has a distinct and well-defined responsibility. This approach enhances code clarity, simplifies debugging, and allows for future extensions with minimal refactoring. The system is organized into a main application orchestrator and several specialized, pluggable modules.

2.1 Component Overview

The application's source code is structured into a main Python package, `src`, which contains the core logic, and a top-level script, `main.py`, which serves as the user-facing entry point.

- `main.py`**: The Orchestrator.** This script is responsible for parsing command-line arguments, initializing the necessary objects, and managing the main application loop. It acts as

the central coordinator, delegating tasks to the specialized modules but containing no algorithmic logic itself. Based on user input, it decides which counter engine ('motion' or 'pose') to instantiate and run.

- `src/utils/video.py`: **Video Stream Module.** The `VideoStream` class encapsulates all interactions with the webcam. Its sole responsibility is to connect to the hardware, capture raw video frames, and release the camera resource upon application exit. This isolates hardware-specific code from the rest of the application.
- `src/counters/`: **The Counter Engines.** This directory contains the "brains" of the application. Each file defines a class that implements a specific counting algorithm.
  - `motion_counter.py`: Implements the proof-of-concept algorithm based on frame differencing. It contains all logic for pre-processing, motion quantification, and the motion-based state machine.
  - `pose_counter.py`: Implements the algorithm using MediaPipe Pose. It handles pose landmark extraction and the posture-based state machine.
- `src/utils/plotting.py`: **Visualization Module.** The `LivePlotter` class is a specialized utility responsible for generating the real-time, scrolling graph used in the motion detection dashboard. It manages a history of data points and renders them onto a canvas for display.

3 Approach 1: Motion Detection (Proof of Concept)

The initial approach to creating the repetition counter was designed as a proof-of-concept to establish a baseline and validate the core application architecture. This method is based on the principle of frame differencing, a computationally inexpensive technique for detecting motion in a video stream without requiring complex object recognition. The goal was to quantify the overall magnitude of movement and correlate spikes in this metric with individual exercise repetitions.

3.1 Methodology

The algorithm processes each frame from the video stream through a multi-stage pipeline to convert raw pixel data into a quantifiable motion score.

1. **Frame Pre-processing:** To reduce computational complexity and isolate relevant motion data, each captured frame is first converted from the BGR color space to grayscale. A Gaussian blur with a 21x21 kernel is then applied.
2. **Frame Differencing:** The core of the motion detection lies in calculating the absolute difference between the current pre-processed frame and the previous one. This operation, performed by `cv2.absdiff`, produces a new image where pixel intensity corresponds directly to the change between frames. Areas with no movement appear black, while areas with significant motion appear in shades of gray.
3. **Motion Mask Generation:** The resulting difference image is then thresholded to create a binary motion mask. Any pixel with an intensity value above a predefined cutoff is set to pure white (255), and all other pixels are set to black (0). This step effectively isolates significant movements from minor background fluctuations.

4. **Motion Quantification:** The final step is to convert the binary motion mask into a single scalar value, or "Motion Score." This is achieved by summing the intensity values of all pixels in the mask. Because non-motion pixels are black (0), this sum is directly proportional to the area and intensity of movement in the frame. This score provides a continuous time-series signal representing the user's activity level.

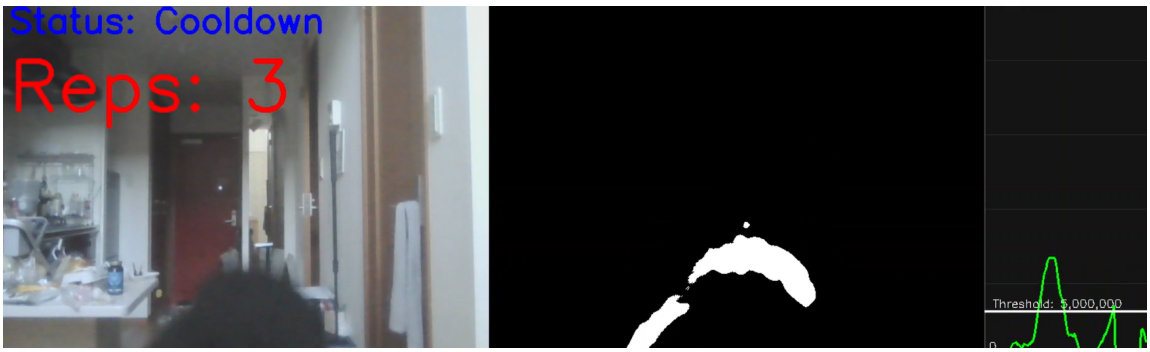

Figure 1: The visual dashboard for the Motion Detection approach, showing the user's live feed, the generated binary motion mask, and the real-time plot of the Motion Score.

3.2 State Machine for Repetition Counting

A simple state machine was implemented to translate the continuous, and often noisy, Motion Score signal into a discrete count of repetitions. The machine operates with three primary states: 'Ready', 'In-Rep', and 'Cooldown'.

A repetition is registered when three conditions are met simultaneously: the Motion Score must exceed a high-value `MOTION_THRESHOLD`, the system must not already be in the 'In-Rep' state, and a time-based 'Cooldown' period must have elapsed since the last rep was counted. Upon counting a rep, the system enters the 'In-Rep' state and resets a cooldown timer. The state only resets to 'Ready' after the Motion Score drops below a lower, secondary threshold, indicating that the high-intensity portion of the movement has concluded. This logic is designed to count a single burst of motion while preventing a continuous, prolonged movement from being counted multiple times.

3.3 Analysis and Limitations

While this approach successfully established a working prototype, testing revealed several fundamental limitations that rendered it unsuitable for reliable, real-world use.

- **Lack of Contextual Understanding:** The system is fundamentally a "change detector," not an "exercise detector." It has no semantic understanding of the scene and cannot differentiate between a burpee, a jumping jack, or a person simply waving their arms. Any motion that is sufficiently large will trigger a count.
- **Environmental Sensitivity:** The algorithm is highly susceptible to environmental factors. Changes in ambient lighting, moving shadows, or other objects in the background can generate a high motion score and lead to false positives.

- **Oversimplification of Exercise Kinematics:** The most significant weakness, discovered during testing, is that a single, complex exercise like a burpee does not produce a single, clean motion spike. Rather, it generates a sequence of distinct spikes (e.g., one for dropping to the floor, another for jumping back up). The simple state machine is incapable of logically grouping these separate events into a single repetition, leading to inconsistent and inaccurate counts.

These limitations made it clear that a more intelligent approach was necessary, one that could understand the posture and form of the human body, rather than just the raw magnitude of pixel changes. This analysis directly motivated the development of the pose estimation model.

4 Approach 2: Pose Estimation (Final Implementation)

The limitations of the motion detection approach necessitated a more intelligent solution capable of understanding the context of human movement. The second and final approach leverages the power of human pose estimation, a subfield of computer vision that identifies and tracks the specific locations of major body joints. By shifting the analysis from raw pixel changes to the structured geometry of a "digital skeleton," the system gains a semantic understanding of the user's posture, enabling a vastly more robust and accurate counting methodology.

4.1 Methodology

This implementation is built upon Google's MediaPipe Pose library, a pre-trained deep learning model optimized for real-time performance on standard CPU hardware. The model analyzes an image and predicts the 2D coordinates of 33 key body landmarks.

1. **Pose Landmark Detection:** For each frame captured from the webcam, the image is first converted from BGR to the RGB color space expected by the model. The frame is then passed to the MediaPipe Pose instance for inference. The model returns a comprehensive data structure containing the coordinates of all 33 landmarks, if a person is detected in the frame.

2. **Landmark Extraction:** While the model provides a full skeleton, only a few key landmarks are required to analyze a burpee. For this implementation, the system isolates the `LEFT_HIP` and `RIGHT_HIP` landmarks. The coordinates provided by MediaPipe are normalized (ranging from 0.0 to 1.0), making the logic independent of camera resolution or distance from the subject. The average vertical position (y-coordinate) of the two hip landmarks is used as the primary metric for determining the user's posture.

3. **Visualization:** To provide the user with clear visual feedback, the detected skeleton is drawn directly onto the video frame using MediaPipe's built-in drawing utilities. This overlay serves as an effective diagnostic tool, confirming that the model is accurately tracking the user's body.

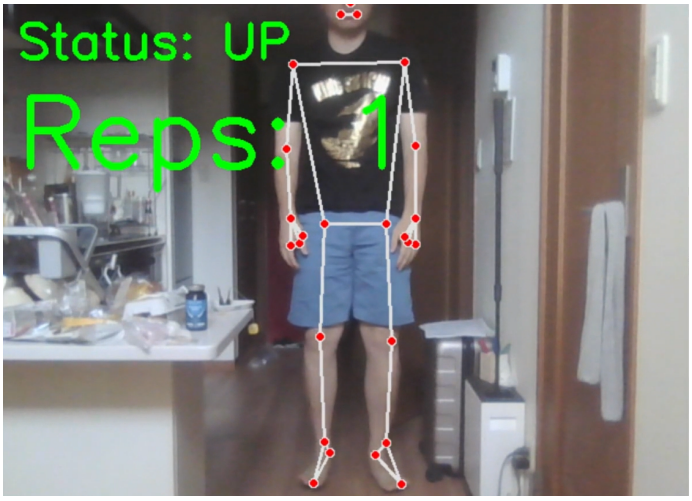

Figure 2: The visual dashboard for the Pose Estimation approach, showing the user's live feed with the detected pose skeleton overlay and the real-time repetition count.

4.2 Posture-Based State Machine for Repetition Counting

With access to structured data on the user's body position, a more sophisticated state machine was developed. This logic is based on the user's posture rather than the magnitude of their movement.

The machine operates with two primary states: 'UP' and 'DOWN'. These states are determined by comparing the average hip y-coordinate to two predefined, normalized thresholds.

- The system transitions to the **'DOWN'** state when the user's hips move below a `down_threshold` (e.g., 0.7), indicating they are in the lower phase of the burpee.
- A full repetition is registered only when the system transitions from the 'DOWN' state back to the **'UP'** state. This occurs when the user's hips rise above a higher `up_threshold` (e.g., 0.4).

This logic ensures that a count is only triggered after a full range of motion has been completed. By explicitly waiting for the user to return to a standing position after being in a prone position, it correctly groups the multiple motion spikes of a single burpee into one valid repetition.

4.3 Results and Advantages

The pose estimation approach proved to be a resounding success, effectively resolving all the limitations of the motion-based prototype.

- **Contextual Understanding:** The system is no longer a simple "change detector." It specifically tracks a human form, making it inherently robust to irrelevant background motion, shadows, or other environmental distractions.
- **Accurate Exercise Modeling:** By defining states based on posture, the counter correctly understands the kinematic sequence of a burpee. It reliably counts one repetition for the entire down-and-up cycle, solving the "multiple spikes" problem that plagued the first approach.

- **Robustness:** The use of normalized coordinates and the model's training on a diverse dataset make the counter highly effective across different users and camera angles without requiring significant recalibration.

The final implementation successfully achieves the project's goal of creating an accurate, reliable, and accessible real-time exercise counter.