The Development of Push Up Counter Android Application with Computer Vision

Renaldi Arlin
School of Electrical Engineering and Informatics
Bandung Institute of Technology
Bandung, Indonesia
renaldi.linar@gmail.com

Rinaldi Munir
School of Electrical Engineering and Informatics
Bandung Institute of Technology
Bandung, Indonesia
rinaldi@staff.stei.itb.ac.id

***Abstract*—Sports assistant devices in daily life have often emerged, one of which is the push-up tracking assistant tool. There are already several types of such devices, such as: 1) sensor-based push-up counters with computers; 2) motion differentiation devices for push-up and non-push-up based on multisensor smartphones without cameras; and 3) camera and computer-based push-up counters. From the previous devices, there is still room for improvement in terms of accuracy and portability. One alternative solution that can be used is the development of a computer vision-based smartphone application. The application development method follows the Waterfall method, and the implementation is carried out gradually without repetition. In this application, the Mediapipe Pose library is used for human pose detection, which provides 33 body points from the smartphone camera. For the classification of push-up body poses, the K-NN algorithm is used, which compares body poses directly and classifies them as upper or lower push-up movements. A total of 301 datasets were used, consisting of 246 CSV files and 55 images, with a total of 173 upper push-up movements and 128 lower push-up movements. The movement model was successfully created from the previous datasets by labeling and normalization, including translation, resizing, and rotation. The application was successfully created using Android Studio, and it includes: the previous model; camera usage program and connectivity with Mediapipe Pose; and push-up counter program, including the classification of push-up movement poses using the K-NN algorithm. The performance of the previous model was tested with 176 test cases, and an accuracy of 84.7% was achieved.**

Keywords—push up counter; computer vision; mediapipe pose; K-NN algorithm; Android application

I. Introduction

Hypertrophy is the growth of muscle cells in humans [1]. This can be achieved specifically in human skeletal muscles by engaging in sports activities with hypertrophic stimuli. According to Schoenfeld, one way to activate such stimuli is by providing mechanical tension, which is resistance force applied to a muscle to make it work [2]. One way to achieve this is by performing exercises such as sit-ups, which provide mechanical tension to the abdominal muscles, pull-ups that primarily target the back muscles, and push-ups for the chest muscles. Engaging in compound movements like squats and deadlifts can also promote hypertrophy in multiple muscle groups simultaneously.

However, it can be quite challenging to achieve hypertrophy, and even more difficult to adopt regular physical activity as a healthy lifestyle. Yet, with adequate muscle growth, individuals can avoid sarcopenia, a condition that leads to muscle mass and function loss, resulting in reduced body endurance. This disease is quite common in the Indonesian population. A study titled "Sarcopenia and Frailty Profile in the Elderly Community of Surabaya: A Descriptive Study," conducted with 308 subjects aged 60-100 years, showed a prevalence rate of 41.8% for sarcopenia [3].

Therefore, there is a need for assistance in helping individuals increase their muscle mass. Some measures have already been taken in modern times, such as workout tracking devices that can help maximize the effectiveness of one's exercise routine.

Wearable technology, including fitness trackers, has become the highest trend in a global survey with 3,037 respondents [4]. Sports trackers have gained popularity among the general population, especially for simple exercises like push-ups. There have been numerous studies resulting in various push-up counting devices, ranging from simple microcontroller-based systems with ultrasonic sensors connected to computers [5]; accelerometer and gyroscope sensors on mobile phones to detect hand movements [6]; to using computer vision to translate the human body into a model of points and lines and count push-up movements by joint angles through a computer [7].

However, the aforementioned devices have some drawbacks, including less accurate counting of proper push-up movements and complex accessibility due to the requirement of using a computer for push-up tracking activities. The previous approaches are challenging to implement in daily human life.

Currently, there is a rapid increase in the trend of using smartphones for daily activities. This offers new possibilities for creating better push-up counting solutions. Smartphones can capture push-up movements with the camera and model the human body as points and lines, similar to the previous computer vision approach. To improve accuracy, Artificial Intelligence can also be used for push-up movement counting, considering that AI projects, references, and libraries are widely available, including in Android applications.

II. Literature Review

A. Musculoskeletal Hypertrophy

According to Guyton and Hall, hypertrophy is the growth of muscle cells in humans [1]. Musculoskeletal hypertrophy specifically refers to the growth of skeletal muscle cells, which are the muscles responsible for moving the human bones, excluding the cardiac muscle in the heart and smooth muscles in other organs of the body.

1) Stimulus Musculoskeletal Hypertrophy

According to Schoenfeld, hypertrophy stimulus refers to the increase in muscle protein synthesis through specific activities performed by the skeletal muscles [2]. This stimulus encompasses several factors, including mechanical tension. Beardsley states that mechanical tension is a force that provides resistance to the muscles, requiring them to perform work [8].

B. Artificial Intelligence

Artificial Intelligence, or AI, is a mechanical simulation system designed to gather knowledge and information, process it using pre-existing intelligence (such as organizing and interpreting), and disseminate actionable intelligent information [9]. When developing AI programs, there are numerous algorithms that can be applied, including artificial neural networks, Naive Bayes, and K-nearest neighbors. These algorithms serve as tools to enable AI systems to learn from data, make predictions, and perform intelligent tasks.

1) Algoritma K-Nearest Neighbor

The K-Nearest Neighbors (K-NN) algorithm is a parameter-free supervised learning method used for classification and regression tasks [10]. In the K-NN classification algorithm, the output is the class or label of a given data point, while in the K-NN regression algorithm, the output is a property, attribute, or feature of an object.

In K-NN classification, when a new data point is to be classified, the algorithm looks for the K nearest data points (neighbors) in the training set based on a similarity measure (such as Euclidean distance) and assigns the majority class label among those neighbors to the new data point.

In K-NN regression, the algorithm finds the K nearest neighbors and predicts the output value for the new data point by taking the average (or weighted average) of the output values of those neighbors.

Both K-NN classification and regression algorithms are based on the idea that similar data points tend to have similar output values, and they rely on the proximity of data points in the feature space to make predictions.

C. Computer Vision

Computer Vision is the discipline concerned with how computers can gain a high-level understanding of images or videos [11]. It encompasses various tasks, including:

1. Image Formation: Understanding the process of image creation, including camera models, optics, and image formation principles.

2. Image Processing: Applying techniques to enhance, analyze, and manipulate images, such as filtering, noise reduction, and image restoration.

3. Model Fitting and Optimization: Fitting models to image data and optimizing parameters to align models with observed image features.

4. Deep Learning: Utilizing deep neural networks to learn and extract meaningful features from images, enabling tasks such as object recognition and segmentation.

5. Image Recognition: Identifying and categorizing objects or patterns within images using machine learning and pattern recognition techniques.

6. Feature Detection and Matching: Locating distinctive image features and matching them across different images for tasks like object tracking and image registration.

7. Image Alignment and Stitching: Aligning multiple images to create a seamless panoramic or composite image.

8. Motion Estimation: Analyzing temporal image sequences to estimate motion and track object trajectories.

9. Computational Photography: Combining computer vision and imaging techniques to enhance image quality, achieve special effects, or enable novel image capturing methods.

10. Structure from Motion and SLAM: Reconstructing 3D structure and camera poses from 2D image sequences or performing simultaneous localization and mapping for navigation tasks.

11. Depth Estimation: Inferring depth information from 2D images, often using stereo vision or depth estimation algorithms.

12. 3D Reconstruction: Building a three-dimensional representation of objects or scenes from multiple images or point cloud data.

13. Image-based Rendering: Generating novel views or synthesizing new images using existing image data.

These tasks collectively form the field of Computer Vision, which aims to enable machines to understand and interpret visual information like humans do [12]. When developing AI programs, there are numerous algorithms that can be applied, including artificial neural networks, Naive Bayes, and K-nearest neighbors [9]. These algorithms serve as tools to enable AI systems to learn from data, make predictions, and perform intelligent tasks.

1) Image Recognition

Image Recognition is the process of identifying an image or video and detecting objects or features effectively using artificial intelligence [13]. It encompasses various tasks, including facial recognition and body pose recognition.

Facial recognition involves analyzing and identifying specific facial features to match them with known identities, enabling applications such as biometric authentication, surveillance systems, and social media tagging. It utilizes algorithms that extract facial landmarks, analyze facial expressions, and compare them against a database of known faces.

Body pose recognition focuses on understanding and estimating the positions and orientations of human body joints and limbs. It enables applications such as gesture recognition, action recognition, and human-computer interaction. Body pose recognition algorithms analyze the spatial relationships between body joints, track movement patterns, and classify different poses or actions.

Image Recognition techniques leverage machine learning, deep learning, and computer vision algorithms to process and analyze visual data, allowing machines to recognize and interpret images and videos with a level of intelligence similar to human perception.

2) Face Detection

Face Detection is an advanced technology derived from image recognition that can locate and extract facial areas from the background of an image [14]. The core process of face detection is determining whether a face is present or not in any given image. If one or more faces are detected, the locations of each face are extracted.

In face detection, there are two broad approaches: feature-based approach and image-based approach.

1. Feature-based Approach: This approach involves defining specific facial features or patterns and designing algorithms to detect those features. Examples of facial features include eyes, nose, mouth, and facial contours. The algorithm searches for these features in the image and analyzes their spatial relationships to identify and locate faces. Techniques such as Haar cascades and Viola-Jones algorithm are commonly used in feature-based face detection.

2. Image-based Approach: In this approach, face detection algorithms analyze the overall characteristics and patterns of the image to determine the presence of faces. The algorithms typically leverage machine learning techniques, such as convolutional neural networks (CNN), to learn and recognize patterns associated with faces. These models are trained on a large dataset of labeled images, enabling them to generalize and detect faces accurately in various conditions.

Both approaches have their strengths and limitations. Feature-based approaches tend to be faster and more efficient but may rely on predefined features and struggle with variations in pose, lighting, and occlusions. Image-based approaches, on the other hand, can handle more complex scenarios but may require more computational resources and training data.

Face detection is a fundamental step in many face-related applications, including facial recognition, emotion detection, age estimation, and face tracking. It serves as a crucial building block for higher-level face analysis and understanding.

3) Pose Estimation

Pose in Computer Vision refers to the visual representation of the position and orientation of an object, typically in three dimensions [15]. Pose Estimation is the process of estimating the pose of an object, usually in three dimensions. One of the key applications of pose estimation is estimating the pose of a human.

Pose estimation for humans involves analyzing an image or video to determine the positions and orientations of various body joints, such as the head, shoulders, elbows, wrists, hips, knees, and ankles. The goal is to understand the spatial configuration of the human body and accurately estimate the pose.

There are different approaches to pose estimation, including model-based and data-driven methods. Model-based methods utilize predefined models or templates of the human body and match them to the image or video data. These methods often rely on prior knowledge of the human body's anatomical structure and joint relationships.

Data-driven methods, on the other hand, leverage machine learning and deep learning techniques to learn the correlations between image features and human poses from a large dataset. These methods train models to directly predict the joint positions or orientations given an input image or video. They can handle more complex poses and variations but require significant amounts of annotated training data.

Pose estimation has numerous applications, including action recognition, motion capture, human-computer interaction, augmented reality, and robotics. It plays a crucial role in understanding human movements and behaviors from visual data, enabling a wide range of applications in various fields.

D. Pose Estimation Library

Mediapipe Pose is a technology developed by Google that combines machine learning and computer vision to model human body pose estimation as a set of 3D skeletal points. It is based on the BlazePose library developed by Valentin Bazarevsky. BlazePose is a fast neural network architecture that can produce 33 key body landmarks for a human and run at 30 frames per second on devices like the Pixel 2 smartphone [16].

Here is a detailed explanation of how the Mediapipe Pose library works:

1. Input: The library takes an input image or video frame as input, typically obtained from a camera or stored media.

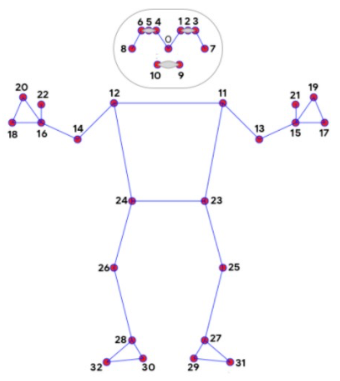
2. Preprocessing: The input image or frame is preprocessed to enhance its quality and normalize the data. This may involve resizing, cropping, or applying filters to improve the accuracy of pose estimation.

3. Pose Estimation: The preprocessed image is fed into the BlazePose neural network architecture. The network processes the image and produces a set of 2D or 3D keypoints representing

the detected body landmarks. These keypoints correspond to important body joints such as the head, shoulders, elbows, wrists, hips, knees, and ankles.

4. Keypoint Refinement: The detected keypoints are refined and adjusted to improve their accuracy and consistency. This step helps to reduce any noise or errors introduced during the pose estimation process.

5. Pose Representation: The refined keypoints are used to construct a pose representation, typically as a set of lines or skeletal connections between the body landmarks. This representation provides a visual depiction of the estimated human body pose.



0. nose
1. left_eye_inner
2. left_eye
3. left_eye_outer
4. right_eye_inner
5. right_eye
6. right_eye_outer
7. left_ear
8. right_ear
9. mouth_left
10. mouth_right
11. left_shoulder
12. right_shoulder
13. left_elbow
14. right_elbow
15. left_wrist
16. right_wrist
17. left_pinky
18. right_pinky
19. left_index
20. right_index
21. left_thumb
22. right_thumb
23. left_hip
24. right_hip
25. left_knee
26. right_knee
27. left_ankle
28. right_ankle
29. left_heel
30. right_heel
31. left_foot_index
32. right_foot_index

Figure I. Pose Landmark List

6. Output: The final output of the Mediapipe Pose library includes the pose representation, which can be used for various applications such as motion tracking, gesture recognition, augmented reality, and more.

The Mediapipe Pose library combines advanced machine learning techniques, efficient neural network architectures, and real-time processing capabilities to provide accurate and robust human pose estimation. It offers developers a powerful tool for integrating pose estimation capabilities into their applications and systems.

E. Android Application Development

Android is a mobile operating system based on a modified version of the Linux operating system. The architecture of the Android operating system consists of five main components: the Linux kernel, libraries, Android runtime, application framework, and applications. In Android app development, developers can use either the Java programming language or the Kotlin programming language [17].

Here's a breakdown of the different components in the Android architecture:

1. Linux Kernel: The Linux kernel forms the core of the Android operating system. It provides low-level functionalities such as device drivers, memory management, process management, and security.

2. Libraries: Android includes a set of libraries that provide various capabilities and functionalities to developers. These libraries cover a wide range of areas, including graphics rendering, database access, networking, multimedia, and more. Developers can leverage these libraries to build robust and feature-rich applications.

3. Android Runtime (ART): The Android runtime is the engine responsible for executing and managing Android applications. It includes the core libraries and the Dalvik Virtual Machine (DVM) or, more recently, the Android Runtime (ART), which performs just-in-time (JIT) compilation and optimization of the application bytecode.

4. Application Framework: The application framework provides a set of reusable components and services that simplify the development of Android applications. It includes high-level APIs for activities, content providers, broadcast receivers, and services. The framework also offers functionalities such as resource management, user interface controls, and inter-process communication.

5. Applications: This layer comprises the actual applications that users interact with on their Android devices. These can be pre-installed system apps or third-party apps downloaded from the Google Play Store or other sources. Applications can range from simple utility apps to complex games and productivity tools.

Developers have the flexibility to choose between Java and Kotlin as the programming languages for Android app development. Both languages are officially supported by Google and provide extensive libraries, tools, and frameworks for building Android applications.

Overall, the Android architecture provides a robust and flexible platform for developers to create a wide variety of applications for mobile devices.

III. Proposed Solution

Solution is built with training process, Android application development, and lastly the push up counter algorithm.

A. Training Process

The training process of the model consists of four phases: 1) Data labeling, 2) Image-to-pose conversion, 3) Normalization, and 4) Attribute adjustment.

1) Data Labeling

In this phase, the acquired dataset is labeled. For CSV file type datasets obtained from the internet, the data already has labels and can directly proceed to the next phase. However, for image data, labeling is performed by separating and categorizing the images into specific folders within the program's code scope, namely "pushups_up" and "pushups_down."

2) Image-to-Pose Conversion

This phase is applied to image data only. The images are processed using the Mediapipe Pose library's method, "mp.solutions.pose.process(image)," which translates the image into 33 points or poses.

3) Normalization

Pose normalization involves several processes, including translation, scaling, and rotation. Translation shifts all 33 points of the pose so that the right hip point is located at coordinates (0, 0, 0). Scaling adjusts the pose's size by dividing all points by the distance between the left and right hips. Rotation performs a 3D rotation on the pose, ensuring that the left hip point has coordinates (X positive, 0, 0), and the left shoulder point has coordinates (X positive, 0, Z positive).

4) Attribute Adjustment

In this phase, attribute adjustment is performed to assign specific attributes to each data instance. The attributes are arranged in the following order: pose_id, label, followed by the 33 pose points.

B. Android Application Development

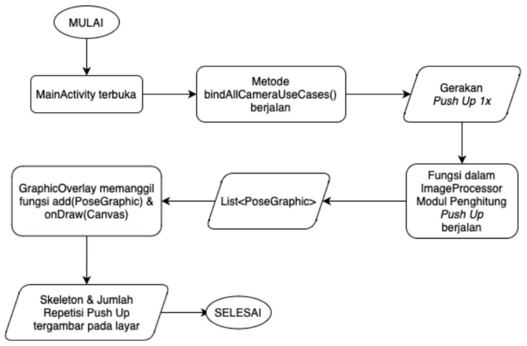

MULAI
MainActivity terbuka
Metode bindAllCameraUseCases() berjalan
Gerakan Push Up 1x
Fungsi dalam ImageProcessor Modul Penghitung Push Up berjalan
List<PoseGraphic>
GraphicOverlay memanggil fungsi add(PoseGraphic) & onDraw(Canvas)
Skeleton & Jumlah Repetisi Push Up tergambar pada layar
SELESAI

Figure III. Flowchart of Android Application Module

In the Android Application module, there is a flow that explains the process of executing the program in the Android Application module. Android Application Module Flow Diagram. The flow of the Android Application module starts with the opening of MainActivity, then the bindAllCameraUseCases method is executed. Next, the user will provide input in the form of a single push-up movement. The ImageProcessor will execute several functions within the Push-Up Counter Module, and the output provided is a List<PoseGraphic>. The GraphicOverlay will take the previous output and call the add(PoseGraphic) method to draw it on the phone screen using onDraw(Canvas). The Android App will have this particular screen inside it.

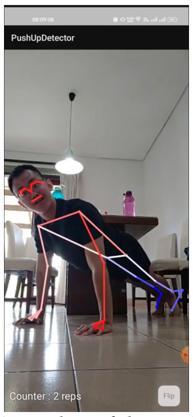


Figure IIIII. Screnshot of the Push Up Detector

C. Push Up Counter Algorithm

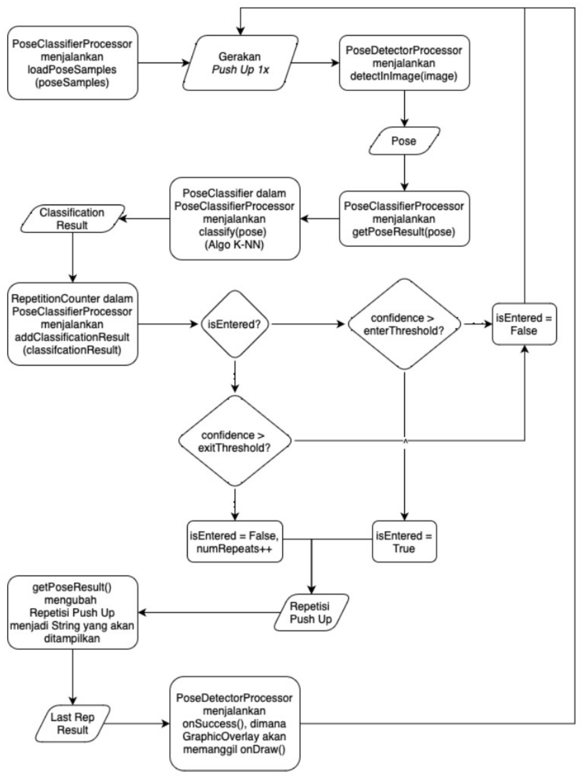

PoseClassifierProcessor menjalankan loadPoseSamples (poseSamples)
Gerakan Push Up 1x
PoseDetectorProcessor menjalankan detectInImage(image)
Pose
PoseClassifierProcessor menjalankan getPoseResult(pose)
PoseClassifier dalam PoseClassifierProcessor menjalankan classify(pose) (Algo K-NN)
Classification Result
RepetitionCounter dalam PoseClassifierProcessor menjalankan addClassificationResult (classifcationResult)
isEntered?
confidence > enterThreshold?
isEntered = False
confidence > exitThreshold?
isEntered = False, numRepeats++
isEntered = True
Repetisi Push Up
getPoseResult() mengubah Repetisi Push Up menjadi String yang akan ditampilkan
Last Rep Result
PoseDetectorProcessor menjalankan onSuccess(), dimana GraphicOverlay akan memanggil onDraw()

Figure IVII. Flowchar of Push Up Counter Module

The flow of the Push-Up Counter module is explained in Figure IV.4. The flow starts with the PoseClassifierProcessor class executing loadPoseSamples with the model created in the Data Set & Model Module. Then, input is provided in the form of a single push-up movement. The image is received by the PoseDetectorProcessor and the detectInImage method is executed, which outputs a Pose. The PoseClassifierProcessor executes getPoseResult with the previous Pose as input. The PoseClassifier inside the PoseClassifierProcessor will then execute the classify method, which classifies whether the given Pose represents an upward push-up, a downward push-up, or a non-push-up movement using the K-NN algorithm. The result of the previous process is a ClassificationResult, and the output is used in the RepetitionCounter to execute the addClassificationResult method.

In this process, it first checks if a push-up movement is being performed or if the downward push-up movement has already been completed. If it has, it checks the confidence of the previous result against the exit threshold. If the confidence is higher, the numRepeats variable increases, indicating that an upward push-up movement has been completed. If the upward push-up movement has not been performed, it checks the confidence of the ClassificationResult against the entry threshold. If the confidence is higher, it means that a downward push-up movement has been completed. If none of the previous conditions are met, it means that a non-push-up movement has been performed.

If the upward or downward push-up movement is classified based on the previous conditional process, an integer value representing the number of push-up repetitions is generated.

This number is converted to a string in the getPoseResult method. The string output of the previous process, lastRepResult, is passed from the PoseDetectorProcessor class to the GraphicOverlay, where it is drawn on the phone screen. contribute to the training process, enabling the model to learn and make predictions based on the labeled and normalized pose data.

IV. Testing and Evaluation

Testing is conducted with the aim of obtaining accuracy of the push-up counter. The accuracy is calculated using the metric of incremental push-up count.

The testing process involves evaluating the system using a set of push-up movements with known accurate repetition counts. During the testing, the user performs push-up movements that are detected and counted by the system. The system's count is then compared to the actual count to determine the accuracy. The metric of incremental push-up count measures how accurately the system tracks the increase in the number of push-up repetitions performed by the user.

A. Test Case

The testing cases will consist of two types: Type A, involving real human users performing push-up movements, and Type B, involving push-up movements performed in videos. For each type, 10 test cases will be conducted. Each test case will include five push-up movements and five non-push-up movements. Examples of non-push-up movements include performing squats, performing push-ups with one leg raised, standing and moving the arms forward and backward like a push-up motion, and others. By creating two types of testing cases, the number of test cases can be increased, resulting in a total of 200 push-up movements for testing.

B. Testing Scenario

The testing is conducted with the camera on the phone positioned diagonally towards the side of the human body, capturing the head to the feet, with the phone oriented vertically. This setup aims to achieve an optimal condition for push-up calculations. The optimal condition is when the entire face to feet of the human body is visible on the phone's screen. This ensures that there are no assumptions in the translation of the image into Pose by the Mediapipe Pose library.

In type A testing, it starts with the user installing the PushUpDetector Android application APK file on their smartphone. The user places the smartphone on a wall or any other stable surface facing them (using either the rear or front camera). Then, the user opens the PushUpDetector application. Next, the user performs five push-up movements and five non-push-up movements.

In type B testing, it is conducted independently. It begins with the installation of the PushUpDetector Android application APK file on a smartphone. The smartphone is then directed towards another device. On the other device, a video is played demonstrating five push-up movements. Then, another video is played demonstrating five non-push-up movements. This type B testing is performed 10 times independently.

C. Testing Result

In Table I, the results of type A test cases are described. There were 7 different users who performed the test with 5-6 push-up movements and 5-10 non-push-up movements. Therefore, there were a total of 76 type A test cases. The test results were recorded using the Xrecorder application and uploaded to Google Drive on the following page:

https://drive.google.com/drive/folders/1X0e0B5Wqb8C4G 3V74q2iAzXuIQcg2t4 5?usp=share_linkz.

TABLE I. Testing Result of Type A

No.	Push Up Counted Correctly	Non-Push Up Counted	Push Up Not Counted	Non Push Up not Counted
1	5	0	0	10
2	5	0	0	5
3	5	0	0	5
4	5	0	0	5
5	5	0	0	5
6	5	0	0	5
7	3	0	3	5

In Table II, the results of type B test cases are described. It provides the results of the type B test cases, which include a total of 50 push-up movements and 50 non-push-up movements. Therefore, there are a total of 100 test cases. The test results were recorded using the Xrecorder application and uploaded to Google Drive on the following page:

https://drive.google.com/drive/folders/15E5JPQ0duwdC5tjOe7 _m7G4juRQRRC Hx?usp=share_link."

TABLE II. Testing Result of Type B

Push Up Counted Correctly	Non-Push Up Counted	Push Up Not Counted	Non Push Up not Counted
48	22	2	28

V. Conclusion and Future Works

A. Conclusion

1. The accurate calculation of push-up movements from a video utilizes Computer Vision by detecting human body poses and classifying them using a human body pose model during the upward and downward phases of a push-up. When a pose is classified as a push-up downward phase and the subsequent pose is classified as a push-up upward phase, it counts as one repetition. The push-up movement classification is performed using the K-NN algorithm.

2. Creating a portable and computationally lightweight solution for Android devices involves developing an Android application that can quickly detect human body poses using the

Mediapipe Pose library. To ensure lightweight push-up calculation, the model is first trained on a computer system.

3. A total of 301 datasets were obtained, consisting of 55 images and 246 CSV files. Among them, there were 173 datasets of upward phase push-up movements and 128 datasets of downward phase push-up movements.

4. The model's performance was tested with 176 cases, resulting in an accuracy of 84.7%.

B. Future Works

For future research, the following suggestions can be implemented:

1. Provide feedback and temporarily pause the push-up calculation process when the body tilt angle is not 45 degrees and the body position is not in the optimal state (ensuring that the face, body, hands, and feet are fully visible on the screen).

2. When there are errors in push-up movements, provide detailed information about the specific mistakes made and the correct form of the movement.

3. Only utilize relevant keypoints in the push-up movement calculation instead of using all 33 keypoints. Some keypoints, such as the nose, may not be relevant to the push-up movement. By focusing on relevant keypoints, it is expected to improve the accuracy of the calculation.

These suggestions aim to enhance the accuracy and usability of the push-up tracking system.

Acknowledgment

The researcher expresses gratitude to God for the blessings received throughout the completion of this research. The researcher hopes that this study will contribute to the advancement of push up tracker and computer vision usage in health and fitness world. During the research process, the researcher received assistance from numerous individuals. The researcher extends gratitude to their family for providing motivation and support. The researcher would also like to acknowledge Mr. Dr. Ir. Rinaldi Munir, M.T. for providing guidance on this research. Additionally, the researcher appreciates the assistance and support received from friends throughout the research endeavor.

References

[1] A. C. Guyton and J. E. Hall, *Textbook of medical physiology*. Elsevier Saunders, 2006.

[2] B. Schoenfeld, *Science and development of muscle hypertrophy*. 2020.

[3] N. Widajanti *et al.*, "Sarcopenia and Frailty Profile in the Elderly Community of Surabaya: A Descriptive Study.," *Acta Med Indones*, vol. 52, no. 1, pp. 5–13, Jan. 2020.

[4] W. R. Thompson, "Worldwide Survey of Fitness Trends for 2020," *ACSMs Health Fit J*, vol. 23, no. 6, pp. 10–18, Nov. 2019, doi: 10.1249/FIT.0000000000000526.

[5] D. Rosadi, L. Hardiansyah, and A. Rusdiana, "PENGEMBANGAN TEKNOLOGI ALAT UKUR PUSH UP BERBASIS MICROCONTROLLER DENGAN SENSOR ULTRASONIC," *Jurnal Terapan Ilmu Keolahragaan*, vol. 3, no. 1, p. 34, Jun. 2018, doi: 10.17509/JTIKOR.V3I1.8064.

[6] M. Nilsson and H. Wilén, "Push-up Tracking through Smartphone Sensors," *DEGREE PROJECT IN TECHNOLOGY, FIRST CYCLE, 15 CREDITS*, 2016.

[7] H. J. Park, J. W. Baek, and J. H. Kim, "Imagery based Parametric Classification of Correct and Incorrect Motion for Push-up Counter Using OpenPose," *IEEE International Conference on Automation Science and Engineering*, vol. 2020-August, pp. 1389–1394, Aug. 2020, doi: 10.1109/CASE48305.2020.9216833.

[8] C. Beardsley, *Hypertrophy: Muscle fiber growth caused by mechanical tension*. 2019.

[9] P. Dalvinder and S. Grewal, "A Critical Conceptual Analysis of Definitions of Artificial Intelligence as Applicable to Computer Engineering," no. 2, 2014, Accessed: Dec. 03, 2022. [Online]. Available: www.iosrjournals.org

[10] E. Fix and J. L. Hodges, "Discriminatory Analysis. Nonparametric Discrimination: Consistency Properties," *Int Stat Rev*, vol. 57, no. 3, p. 238, Dec. 1989, doi: 10.2307/1403797.

[11] D. H. Ballard and C. M. Brown, *Computer Vision*. New York: Prentice Hall, 1982.

[12] R. Szeliski, "Computer Vision," 2011, doi: 10.1007/978-1-84882-935-0.

[13] H. Bhardwaj, P. Tomar, A. Sakalle, and U. Sharma, "Principles and Foundations of Artificial Intelligence and Internet of Things Technology," *Artificial Intelligence to Solve Pervasive Internet of Things Issues*, pp. 377–392, Jan. 2020, doi: 10.1016/B978-0-12-818576-6.00020-4.

[14] A. R. Khan, *Face Detection and Recognition Theory and Practice*. Chapman and Hall/CRC, 2015. Accessed: Feb. 23, 2023. [Online]. Available: https://www.academia.edu/26626901/Face_Detection_and_Recognition_Theory_and_Practice_eBookslib

[15] W. A. Hoff, K. Nguyen, and T. Lyon, "Computer-vision-based registration techniques for augmented reality," D. P. Casasent, Ed., Oct. 1996, pp. 538–548. doi: 10.1117/12.256311.

[16] V. Bazarevsky, I. Grishchenko, K. Raveendran, T. Zhu, F. Zhang, and M. Grundmann, "BlazePose: On-device Real-time Body Pose tracking," Jun. 2020.

[17] J. DiMarzio, *Beginning Android Programming with Android Studio (Wrox Beginning Guides)*. 2016.