# Automatic Repetition Counting and Form Assessment

Domenico D'Orsi, Matteo Manni, Lorenzo Mazzei, Denny Meini

d.dorsi@studenti.unipi.it , m.manni3@studenti.unipi.it , l.mazzei7@studenti.unipi.it , d.meini5@studenti.unipi.it

## ABSTRACT

Gym exercises are frequent activities for each athlete that wants to keep training their body. Despite all their best efforts, even the most expert ones are prone to errors: bad body posture, incorrect placing of the arms or feet etc.… Keeping track of the number of repetitions for an exercise may also be more difficult than expected sometimes, especially if the athlete is tired or is performing the exercise rather quickly. This issue could be tackled by asking a personal trainer to check for the athlete's execution of the exercise, but this solution has some problems, e.g. the personal trainer may not always be available, and it can also become expensive if needed for a prolonged period of time. In this paper, we introduce Automatic Repetition Counting, an application that we propose as an alternative solution for the aforementioned problem. Our application makes use of the camera of the athlete's smartphone in order to keep track of the number of repetitions for the exercise that is being carried out and to perform Form Assessment. There is also the possibility to connect a smartwatch for BPM monitoring. In the end, the application will report to the athlete information about exercise correctness and the BPM mean value.

## 1 Introduction

So far, different methods have been proposed in the literature regarding this topic, by making use of technical equipment and multiple devices, such as Cameras, body sensors, smartphones kept in the pockets during the exercise and so on. The athlete in this sense is constrained while doing the exercises and may not be encouraged to utilize these methods since he/she may prefer the possibility of making some mistakes during the executions with respect to having to setup everything and wearing those devices.

Our idea is to make use only of the athlete's smartphone: our application will record the athlete while doing the exercise and will provide reps counting and form assessment thanks to the MediaPipe PoseLandmarker model. Moreover, there's also the possibility to attach a smartwatch through Bluetooth to monitor BPM values.

MediaPipe is an open-source framework developed by Google that offers customizable machine learning solutions for various media processing tasks, primarily focused on video and image analysis. It provides a set of pre-built components and tools that developers can use to build applications for tasks such as object detection, pose estimation, facial recognition, hand tracking, and more. In particular, we use the PoseLandmarker model from the MediaPipe framework: the Pose Landmarker task lets you detect landmarks of human bodies in an image or video. Therefore, this task can be used to identify key body locations, analyze posture, and categorize movements. It uses machine learning (ML) models that work with single images or video streams.

## 2 PoseLandmarker

The PoseLandmarker model takes images as input and returns 33 landmarks.

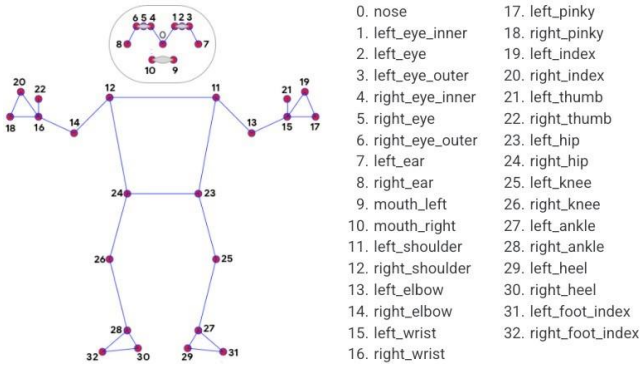


**Figure 1:** Landmark types and positions

Once the input is passed to it, the model goes into the Pose Estimation phase, where machine learning algorithms are run to detect the landmarks.

There are 33 landmarks detected. These landmarks are given in output from the model as a list of 33 elements, the index number corresponds to the number associated with the landmark in the fig.1 (e.g. the 26$^{th}$ element of the list corresponds to landmark 26 from fig.1). In particular, the coordinates for each landmark are computed: x-axis, y-axis and z-axis.

So, overall, the model returns a 3-dimensional information about all the main human parts that characterize one pose with respect to another, in this sense is perfect for our purposes.

## 3 Methodology

The Squat Monitoring is done by computing metrics regarding distance between landmarks and angles of some key body parts. We make use of the Euclidean Distance and Dot Product to evaluate angles and distances.

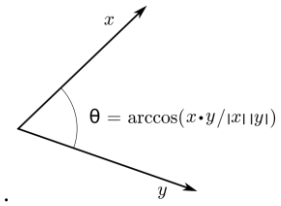


**Figure 2:** computation of the angle

We implemented the monitoring only for the squat exercise; however, it is easy to extend the application and permit other types of exercises with a low effort since the main logic is common to most exercises.

We exploit landmarks positioned on the left side of the human figure. For this reason, a user should position him/herself on the right side in front of the smartphone camera, in this way from the point of view of the smartphone the user side in foreground is the left one. Furthermore, to obtain the best performance in exercise monitoring, the user should stand neither in front nor by profile with respect to the camera, but slightly rotated. In fig. 3 an example is shown:

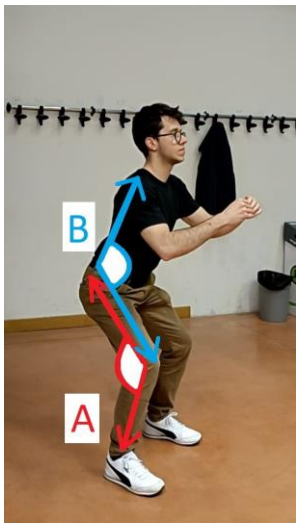


**Figure 3:** angleHip (B) and angleKnee (A)

The landmarks are in the 3-dimensional space and so all of them should be rotation-invariant, unfortunately the model is not perfect: the values regarding x and y axis are stable but the values for the z axis are often wrong or inconsistent. For this reason, we prefer to exploit 2-dimensional landmarks (x, y) for the part of repetition counting and partially for form assessment part. In the 2-dimensional space, the rotation-invariant property is lost, but the translation-invariant property still holds, this was enough to obtain satisfying results.

The landmarks considered are five: **12, 24, 25, 26, 28** (see Fig1). The following values are computed:

- **angleKnee:** 2D Angle of the knee (using landmarks **24, 26, 28**)
- **angleHip:** 2D Angle of the hip (using landmarks **12, 24, 26**)
- **ankleDistance:** 2D Distance between the ankle and the feet (using landmarks **26**, **28**)
- **kneeDistance:** 3D Distance between the ankles (using landmarks **25, 26**)

### 3.1 Repetition Counting

The repetition counting segment involves the angleKnee measurement and it is implemented by using a Finite State Machine, which works as follows:

***Exercise_State***: keeps track of which point of the exercise execution the user has reached (e.g. start of the squat, end of the squat, descending, ascending etc.…)

***Phase***: it represents how much low you are in the squat, both for descending and ascending. There are 3 phases in total (0,1,2) and the user is in one of these 3 phases depending on the angle of his/her knee during the exercise execution:

- **Phase 0**: in this phase the user is either at the start of the exercise (he/she is starting to bend down while descending) or at the end of the exercise (he/she is almost back at the starting position while ascending). The user is in phase 0 if **angleKnee ∈ (145,160)**
- **Phase 1**: in this phase the user is either almost getting to the bottom of the squat while descending or he/she is just the bottom of the squat while ascending. So this phase represents the middle part of the squat both while descending and ascending. The user is in phase 1 if **angleKnee ∈ (90,145]**
- **Phase 2**: in this phase the user is at the bottom of the squat. The user is in phase 2 if **angleKnee <= 90**

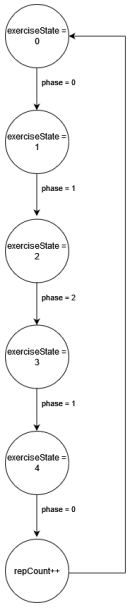


**Figure 4:** Finite State Machine

Overall, the Exercise State and Phase variables allow us to keep track, combined, of the execution of the whole exercise. When the user does a full execution of the exercise (standing → descending → ascending), the number of repetitions is increased.

#### 3.1.1 Model Behavior Challenges

Despite working efficiently, we though had to find a way around a specific behavior of the PoseLandmarker model: the model tries to predict the values of each landmark as soon as it is called in the code, this means that during the time the user is setting up for the exercise there is a possibility that the values of the landmarks we work with for detecting the repetitions fall in the range of values we have during an actual repetition. So, in this case we have a false positive. Since there was no way to stop the model from predicting the values once it is called, we decided to insert a timer before the start of each set, so that the user can setup for the set and rest between any two sets and so that the model is not called in the code while the timer is on.

### 3.2 Form Assessment

The Form Assessment segment involves the angleHip,kneeDistance and ankleDistance measurements and it is also performed during *Phase* 0,1 and 2:

- ***Phase 0***: the angle of the hip is checked to see if the user is bending the body correctly, if he/she is bending too much towards the knees, the system signals an incorrect execution; also, if the knees are too wide with respect to the distance of knee from the feet, the system signals an incorrect execution. Therefore, the user is executing the **exercise in a wrong way** in phase 0 if **at least** one of the following 2 conditions result true:
  A) angleHip < 140
  B) kneeDistance < 4*ankleDistance or kneeDistance > 8*ankleDistance

- ***Phase 1***: like in phase 0, we check the hip angle and the knees distance with respect to the distance of the knee from the feet. Therefore, the user is executing the **exercise in a wrong way** in phase 1 if **at least** one of the following 2 conditions result true:
  A) angleHip <= 90
  B) kneeDistance < 3*ankleDistance or kneeDistance > 8*ankleDistance

- ***Phase 2***: in this phase we only check for the knees distance with respect to the distance of the knee from the feet, since when the user arrives in this phase, the hip movement is limited. Therefore, the user is executing the **exercise in a wrong way** in phase 2 if the following condition result true:
  A) kneeDistance < 3*ankleDistance or kneeDistance > 8*ankleDistance

We used both kneeDistance and ankleDistance in this phase because we wanted to check for the correct distance of the legs, more specifically the knees, e.g if they were too wide. Measuring only the distance of the knees was not the proper way to do it since it changes depending on the user, so we opted for a relative measurement.

#### 3.2.1 Model Behavior Challenges

The form assessment segment was more challenging to tackle, due to some imprecisions of the model in the landmarks' computation. We noticed that the hip angle measurements were likely always lower than the actual angle created by the user body, e.g even when standing up and still, that angle was around 170 instead of 180. This issue was solved by simply setting a threshold that lower in value as well during the 3 phases. There were still some mistakes made by model that we had no way to avoid, e.g sometimes the coordinates of a landmark were off from their true value significantly, so to mitigate this noise we put a threshold (**exerciseThresholdOnErrors**) on the number of times the system should detect an incorrect posture during the execution of the exercise in order to be fairly sure that it is an actual bad exercise performance, if so then it will be signaled to the user. We performed various tests alternating intentionally between good executions and bad executions to determine a good value for the **exerciseThresholdOnErrors**; we found a really good solution in setting **exerciseThresholdOnErrors = 10**.

## 4 Smartwatch Connector

An important component, introduced in order to guarantee to the user insights about BPM during the workout, is the smartwatch connection with the application. This is obtained by means of a service called SmartwatchConnector.kt, which is in charge of receiving data from the smartwatch and forwarding them to the CameraActivity, so that the current BPM value can be printed on screen and to show, in the final exercise report, the mean value during the workout.

The main problem encountered with this approach is due to the fact that our smartwatch model is a FitBit, which is well known for not having an open and direct API to send data to a device. Currently, the Fitbit smart watches measure heart rate, but do not advertise as generic heart rate monitors and can only maintain one Bluetooth Low Energy connection with the Fibit phone app.

The only way to make this work around is the following:

- Install a specific application on the FitBit, downloading it from the specific store.
- Keep the FitBit phone application open along with our application.
- Retrieve the data posted by the FitBit application on the web repository.

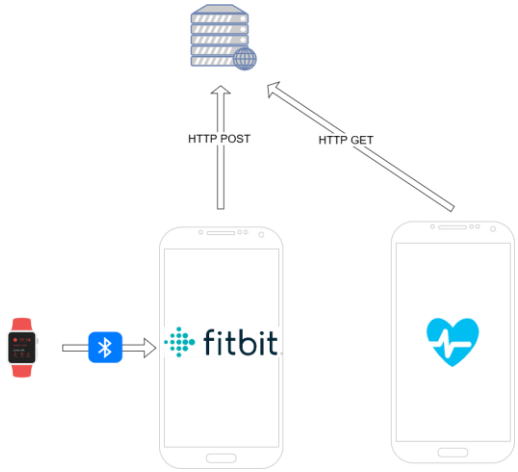


**Figure 5:** smartwatch workflow

In this way, BPM values gets correctly forwarded to the main application and this is also the main reason for which 4 our application requires the user to enable Bluetooth, otherwise the FitBit app couldn't connect to the smartwatch.

In the code, we simply leverage the io.ktor library for HTTP operation, in particular for the POST one, where each time new data gets posted, we forward its content to the main activity. On the main activity side, instead, the only operation required is to register to the service and to initialize a BroadcastReceiver, which listens for the "*android.intent.action.BPM_UPDATE*" intent.

## 5 User Interface

When the application is started, the window reported in fig.6 is presented to the user.

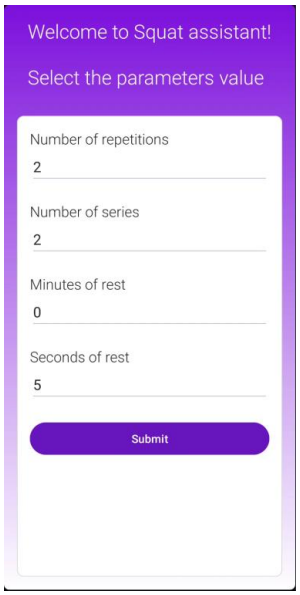


**Figure 6:** UI at the start of the application

The user is asked to insert 4 parameters values:

- **Number of repetitions**: how many repetitions of the exercise have to be done in a set
- **Number of sets**: how many sets have to be done
- **Minutes of rest**: how many minutes the user will rest between any two sets
- **Seconds of rest**: how many seconds the user will rest between any two sets

The total time of rest between each set is therefore given by: Minutes of rest + Seconds of rest. Once the user has inserted of these values, he/she can click the Submit button and after doing so, another window will be presented to the user, as in the fig.7 here below:

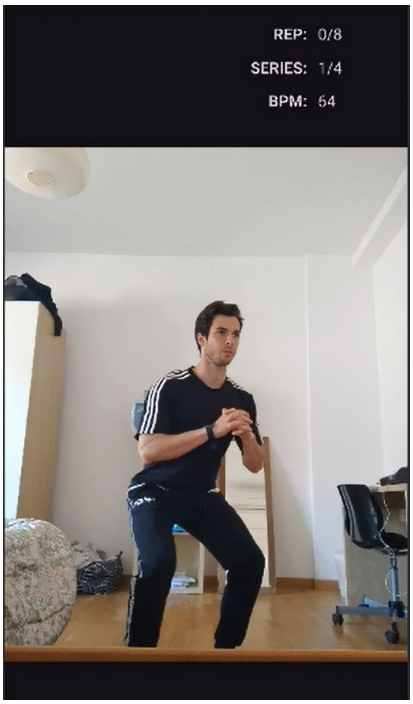


**Figure 7:** UI after submitting the parameters

As it can be seen, the phone's camera is activated in order to start recording the user performing the exercise. A timer is started at this point, so that the user can get into position for performing the exercise: the repetition counting algorithm won't be executed until the timer runs out. In the top-right of the window there are 3 elements:

- ***REP***: here the number of repetitions will be reported and updated along the execution of the exercise.
- ***SERIES***: here the current number of sets reached by the user will be reported.
- ***BPM***: here the current Heart Beats Per Minute of the user will be reported.

Once the user has concluded their training, the following fragment is shown to the user (fig.8):

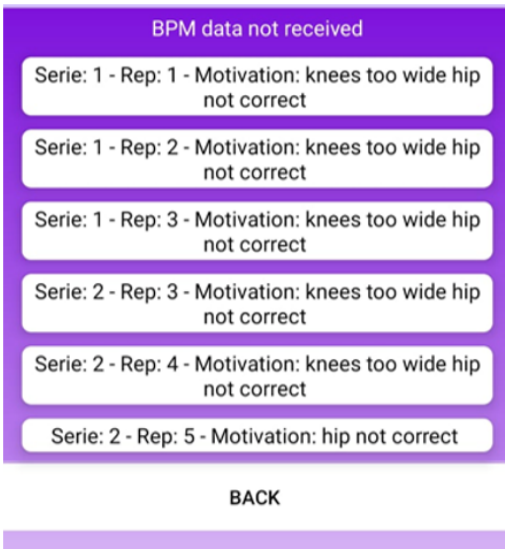


**Figure 8:** Exercise Report

During the exercise, the Form Assessment algorithm will check for its correct execution; if a mistake is detected, it will be reported in the window above (fig. 8) so that the user can see what he/she did wrong. In this example it can be seen that the user committed various errors in series 1 and 2.

## 6 Results

The application was tested by using 3 subjects. The subjects were asked to perform correct squats for 3 sets of 10 repetitions, and subsequently another 3 sets of 10 repetitions but this time trying wrong postures and poses intentionally to check the Form Assessment correct functioning.

### 6.1 Repetition Counting

The repetition counting segment works really well with almost all the repetitions being correctly detected, this is due to the fact that we used the angle of the knee for this part, making it basically independent of the person performing the exercise. The results for this part are shown in the fig.9 below:

| User | n.Rep | n.Rep correctly counted | Acc. |
|---|---|---|---|
| 1 | 60 | 56 | 0.93 |
| 2 | 60 | 58 | 0.96 |
| 3 | 60 | 57 | 0.95 |
| tot | 180 | 174 | 0.96 |

**Figure 9:** Counting Repetitions Results

As it can be seen, the results are pretty good and consistent independently from the user performing the exercise.

### 6.2 Form Assessment

The records of all 3 users are merged. We created a confusion matrix to represent the results: True Positives (TP) represents the right executed repetitions actually recognized as right by the app, False Positives (FP) represents the wrong executed repetitions erroneously recognized as right, False Negatives (FN) and Negatives (TN) are analogous.

| | Predicted True | Predicted False |
|---|---|---|
| Actual True | 87 (TP) | 3 (FN) |
| Actual False | 24 (FP) | 56 (TN) |

**Figure 10:** Form Assessment Results

**F1 score**: 0.86
**Accuracy**: 0.75

## 7 Conclusion

In this work, we developed an application for Automatic Repetition Counting and Form Assessment. By using the Mediapipe Pose model, we were able to obtain the landmarks of key body parts and the associated coordinates. We used these coordinates to compute 2 metrics (Euclidean Distance between landmarks and Angle of some body parts). The Repetition Counting segment was developed through a finite state machine which makes use of the angleKnee. The Form Assessment segment was implemented with checks on the angleHip, kneeDistance and ankleDistance measurements. We took some countermeasures such the initial timer, the use of 2-dimensional space instead of 3-dimensional for some of the metrics and the **exerciseThresholdOnErrors** parameter in order to mitigate the imprecisions of the model. Overall, we were able to obtain satisfying results both on the Repetition Counting segment and the Form Assessment segment. There is space for further improvements for our application, in fact our approach could be extended to other types of gym exercises and in the future, it could be able to help athletes in performing correctly a whole training session consisting on various types of exercises. The implementation of both the Form Assessment and Repetition Counting could also be improved by trying to make use of Machine Learning models to detect the three phases and the wrong postures in a more robust way.