Group Project

Workout Assistant - Final Release

by

Group 1
Odd-Øystein Mathisen, Yauhen Yavorski & Marija Konstantinovna Bravikova

in

IKT213-G 22H
Machine Vision

Guided by Nadia Saad Noori

Grimstad, November 25th, 2022

Abstract

This report covers the planning, research, design, and implementation of a back-end solution for a digital workout assistant. The project aims at detecting posture and repetitions during a workout session, as well as historical data to track progress over time. The workout assistant front end is envisioned to be run on edge mobile devices or as monitor feeds at static gym installations. Several research paths are covered. The systems posture detection is designed around body part detections, trigonometry, and state machines. The project was optimized several times. The final proof of concept manages to fulfill the project scope.

Contents

List of Figures

1 Introduction

1.1 Problem Definition

At times it can be challenging for people to work out by themselves, and at the same time, keep track of repetitions and try to assess their posture and form while working out.

The main goal of this project is to make the proof-of-concept of a back-end for a Workout Assistant app, where the solution will process a video feed of a person working out, count repetitions, and eventually give feedback on the user's posture either during or after a session.

Due to the back-end nature of the system, it can be used as the main driving force of both mobile and permanent installations. Mobile devices could be a user's mobile device at home, or a laptop at a work environment. Permanent installations could be a gym's static workout-bench facility where each workout routine could be customized to the relevant workout bench.

1.2 Objectives and Aims

The project's solution is divided into four main parts.

1. **DETECT**
   Detect features from a video feed.
2. **MAP**
   Map the detected features to defined workout routines and accepted postures.
3. **COUNT**
   Implement logic for repetitions based on valid detected poses.
4. **FEEDBACK**
   Give feedback on bad postures (if any) or repetitions, either during- or post-workout.

The feedback and count system should be as similar to a human training assistant as possible. The post-workout feedback should allow the user to review their historical workout session results.

1.2.1 Use Cases

The project has two prominent use cases. The first is to *give the user live feedback on posture and repetitions during a workout session*, and the second is to *allow a user to review their workout progress over time.*

1.2.2 Project Scope

As this is a proof of concept, the scope is limited to one type of exercise. The focus of this project is a *push-up routine*. The system's primary goal is to recognize a person's posture and track how many push-ups the person takes. Further, the system will give feedback on the user's posture while doing the push-ups and allow the user to track their workout progress over time.

2 Theoretical Background

2.1 Feature Detection Using Machine Learning

To be able to detect a human's posture in an image, the system has to detect specific features in the given image. There are many methods of detecting features in a frame, and several methods were

considered in this project. A promising project leveraging machine learning using YOLO5[1] was initially considered a good starting point, as it would allow for direct pose detection, which could be used to count repetitions and detect bad poses. Making a custom data set was considered, but being computer engineer students, the shortcomings of our general physique quickly revealed that this was not an optimal solution. Infinity.ai[2] is a company that designs custom, premium, computer-generated imaging (CGI) data sets for any use case. Among other things, the company released free samples of CGI renders of people doing push-ups in various poses[3], which could be an optimal data set for this case. The free demo was under the `CC BY 4.0` license, which meant it could be freely used for this project.

2.2 Machine Learning on a Mobile Device

The preliminary project idea was to run the workout assistant on edge devices like Android or iPhone. Some YOLO5 projects are specifically designed for Android[4] or variations of YOLO5 by Ultralytics[5], with models trained for iPhone. The benchmark reports do not look promising for any of these cases [6][7]. The analysis presented in these reports shows that the performance of the mobile implementations of YOLO is not as high as this project requires. Based on these benchmarks, a conclusion was made that current phones cannot process data at a sufficient speed without draining the battery at an unproductive rate.

2.3 Google ML Kit Pose Detection

Since the machine learning using YOLO turned out to be too heavy to process for a mobile edge device, the research continued to find other ways to detect features that would be light enough for an end device. One of the frameworks considered was ML Kit Pose Detection, released by Google [8]. The framework allows edge mobile devices to stream data to a back-end server and retrieves processed responses. The framework works on Android and iPhone devices but requires writing the code in Kotlin or Java for Android and Swift or Objective-C for iPhone. The time required to learn these languages would put the project at a severe time deficit as opposed to the actual implementation of the project scope.

2.4 Redefining the Research Approach

Researching the entire project as one problem became too large of a task. From now on, the problems were split into four issues, `feature detection`, `feature mapping`, `counting repetitions`, and `providing feedback`, where each issue was researched independently.

2.5 Machine Learning as a Back-End Solution

After the research approach was redefined, a full week was set aside to understand how machine learning worked and how to use the push-up data set from Infinity.ai in a basic implementation. This proved to be a definite roadblock due to a lack of knowledge in the field of artificial intelligence.

To understand and learn essential machine learning concepts and implementation tools, courses from Educative[9] and Front-End Masters[10] learning services were used. During the testing phase, a simple model for pose detection was implemented using Keras[11] and Tensorflow[12] libraries. Training a model for a specific angle made the model too sensitive, which meant that the model could not detect people when the angle between the person and the camera was slightly changed. Continuously adjusting the model turned out to be too time-consuming.

The course professor was contacted to get some guidance on edge devices and machine learning. After the discussion, it was decided to instead pivot towards a back-end solution. The back-end solution allows removing the mobile device from the equation. It was also decided to look for alternative ways to solve

the problem without the time-consuming AI implementation. The project implementation should instead proceed with the assumption that edge devices would send a video feed to the system, where the system acts as a back-end service solution.

2.6 OpenCV and MediaPipe

After redefining the approach used in the project, a tool that would help process the video stream was needed. OpenCV[13] (Open Source Computer Vision Library) is designed for computer vision in real-time[14]. OpenCV provides the functionality to perform a video capture and a magnitude of image processing options.

Further, the system should be able to detect features from the video data stream without relying on machine learning. The research should pivot towards pre-trained models that can be integrated into the project. The Mediapipe[15] library was considered a promising option for this task. This library offers pre-computed machine-learning solutions out of the box. It allows the detection of poses in an image without training a custom model on your own for each project[16]. MediaPipe's pose solution contains key point detection. The key point detection was envisioned as a base layer of abstraction that could be used to further develop the pose detection algorithm of this project.

2.7 CVZone

A test implementation was written during this preliminary research to ensure the researched material was feasible. For instance, a rudimentary count logic was implemented to check for repetitions based on hard-coded key point calculations. While testing OpenCV and MediaPipe, it was discovered that neither MediaPipe nor OpenCV have typing support yet (as of writing, November 2022). This made static type-checking of the data arguments in functions quite cumbersome. After some research on ways to automate this process, CVZone[17] could be a good solution for this problem. CVZone is a framework wrapper for some OpenCV and Mediapipe functions that have defined typing in the implementation.

On the other hand, the CVZone framework has limited documentation and a fairly relaxed development schedule[18] compared to the extensive documentation and systematic development updates of the MediaPipe and OpenCV libraries. Based on these factors it was decided that CVZone should not be relied on as a separate library. Instead, some of CVZones features should be used as inspiration for similar features within this project.

2.8 Push-up Variations

Various sources from online publications were used to gather data that can be used for defining push-ups[19][20][21]. A multitude of variations exist, each having different good or bad practices regarding the posture of the user. The data was processed, and the common factors of good or bad push-up postures were extracted. A set of variables covering many *good* variations and some of the *bad poses* was defined through an elimination process to minimize the number of variables.

2.9 State Machines

A test implementation of the variables of a good exercise routine was mocked up. When using all the bulk data, the calculations were cumbersome to work with. A brainstorming session led to research surrounding state machines. Askpython.com has a introduction to state machines in python [22]. This allows splitting large chunks of data into smaller and more manageable segments.

2.10 Data Storage

In order to store, retrieve and use data over time, it is required to have a storage mechanism in the system. The initial choice for the storage, was between using a light weight file, like JSON or CSV, or using a database. It is generally faster to read data from a file than from a database, but databases provide ACID (Atomicity, Consistency, Isolation, Durability) compliance [23]. Using a file for storage would be better for the overall performance of the application, but the ACID compliance of a database simplifies long term development, maintenance and extendibility of the project. Therefore storing in a database was concluded to be a viable solution. A database was chosen to store historical data because it allows for queries on the data, easier access for different types of data, and a combination of records [24].

An SQL database was chosen in the implementation to allow for future extendibility by storing data in different tables, making queries, and combining data easier. Disadvantages might be the lower performance of an SQL database compared to a NoSQL database [24], but for the project, performance loss will not significantly impact the project's usage. An `SQLite` database was chosen because of its lightweight and simple usage properties [25], compared to heavier Postgres or MySQL databases, that are better suited for production environment [26]. Considering that the goal of the project was to make a proof-of-concept, `SQLite` was considered as an optimal temporary solution, and migration to another SQL database could be done in preparation for a release or production phase.

3 System Design and Implementation

3.1 Design

Based on the system requirements described in section 1.2, the system is divided into five main modules, as shown in figure 1. This is the final design after several passes of research. The video feed is a simulated external video source, acting as a user's entry point into the system. The first module detects the features in a given video stream using MediaPipe's landmark model. The second module uses several state machines to map the detected features to states each body part can be in. The combined states of all these state machines make up poses a user can be in during a push-up routine. The third module uses the states of the body parts to map them into postures and uses the postures to track the number of repetitions of the current workout. The main module (not depicted) extracts both the repetitions and the user's current posture and displays this data back to the user in a live feed. The storage module collects the data about the repetitions after a completed workout session and stores it in a database. The fifth module allows the user to access the data stored in the database and displays it back to the user in a user-friendly way. This will enable users to review their previous workout sessions and track their progress over time.

3.2 Implementation

The implementation is divided into five modules: detector, region of interest, state machine, exercise logic, and utilities. The utilities module is a collection of helper functions or classes that makes the rest of the implementation easier to work with.

The modules work together in a coherent system, allowing for the completion of the use cases mentioned in section 1.2.1. The application client side provides a video feed that acts as a simulated entry point into the system's back end. The back-end system will detect key points, add ROI, map points into poses, and perform repetition logic, as shown in figure 13. The figure also illustrates more detailed use cases of visualizing data, where data is requested, retrieved from a database, and displayed to the user.

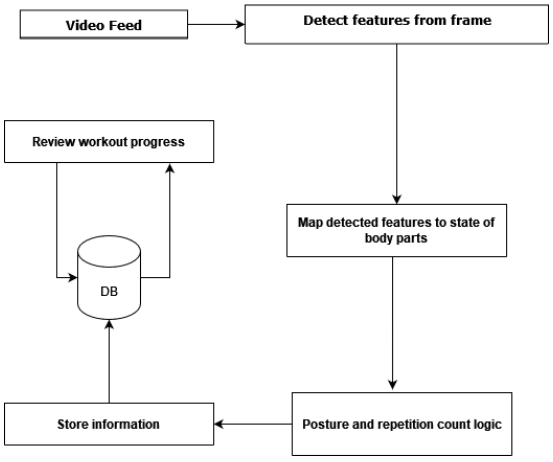

Figure 1: High-level UML description of the application

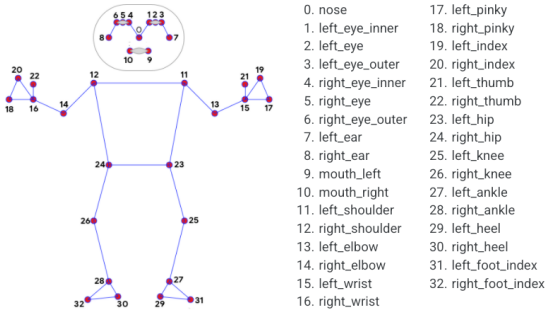

Figure 2: Landmarks that the MediaPipe landmark model can detect [16]

3.2.1 Detector

The detector class was derived from information in a video by Nicholas Renotte[27]. The detector class process a given image and detects landmark key points using MediaPipe's landmark model. This model analyzes an image and detects as many human body landmarks as possible. The model can detect all the landmarks in figure 2. This abstraction level allows the system to detect a list of coordinates of landmark key points, which can be processed further. The landmark object consists of the x-, y-, and z-coordinates and a visibility value. The x-, y- and z-coordinate indicate the width, height, and depth of the detected point on the image. The visibility value represents the system's confidence in the detected coordinates mapping.

The detector class is initialized at the start of the program. Every time a frame is read from the camera, the detector extracts a list of all the landmarks detected in the current frame. Then, the landmarks are drawn on top of the processed frame. Figure 3 illustrates how the landmark key points are drawn onto the final image.

3.2.2 Body Part Identification

Using the detector object, key points are recognized, and their coordinates are stored in a list. The landmark list is used to extract body parts. To consider a body part as detected, the *visibility* of all the landmarks that make up the body part must be above a predefined threshold limit. In other words, a body part is considered as undetected if at least one of its landmarks is not visible enough.

In the current system logic, two body parts are used: the arm and the back.

An arm consists of three parts in the implementation: the shoulder, elbow, and wrist landmarks. For each arm, all three landmarks must be considered visible for the whole arm to be detected. All three points must be visible because they are used to calculate the angle between them. The calculated angle indicates the bending of the arm (in this example).

There are usually two arms involved in the push-up process. One of the arms might not be visible to the application's camera from a particular camera angle. In such a case, only one visible arm will be considered in the push-up process, and the state of the visible arm will be assigned to the non-visible one. The implementation was designed in such a manner because of the assumption that the none visible

arm will mirror the visible arm, as shown in figure 3 where the left arm is not detected, and the push-up process is still allowed to continue.

Another body part that has to be detected and considered in a push-up is the user's spine. The user's back should be straight during the push-up, which means that the system has to check the user's back straightness and give any feedback on it. In the implementation, landmark key points of both hips, shoulders, and knees are used when considering whether the back is straight. The angle between these points is used to evaluate the user's back straightness. When the points are not sufficiently visible, the system cannot detect spine straightness and triggers a `Posture Abort!` warning. This warning is informational feedback to the user, letting them know they can still perform push-ups, but there will be no helpful feedback on the form of those push-ups. This is shown in figure 4, where the user is informed about the lack of knee visibility. Since a camera can't view both sides of a human body during a push-up from a certain angle, only the visible side will be considered in the back angle calculation.

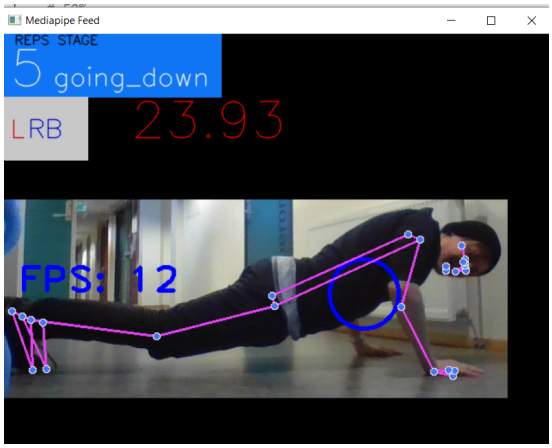

Figure 3: Visibility example where the left arm is not detected

Figure 4: Angle highlight example, showing missing legs

To better illustrate to the user which parts are most important for the push-up logic to work, all angle-related landmarks on both hips and elbows are marked with circles. Figure 4 shows a decent illustration of these circles.

3.2.3 State Machines

To make the body part logic more abstract, the project implements state machines for each major body part related to a workout and one state machine related to the overall exercise. The combined logic of the state machines and their interaction will be described in section 3.2.4.

One state of an exercise consists of two parts: a state of the arm(s) and a state of the back posture. Separate instances of the related state machine classes handle each state's logic.

The application's design was modeled as an abstract finite state machine (FSM). The FSM describes a conceptual model of computation consisting of a set of states interconnected by transitions[22]. Observable body parts of a person are abstracted to a group of states, and depending on the position of these observable body parts, their related state is updated.

An arm is modeled using a class called Hand, which extends a state machine comprised of the five states depicted in figure 14. The *Init state* is the starting state. The arm's current state depends on its angle

and previous state. The arm angle is determined by calculating the angle between the three landmarks, shoulder, elbow, and wrist. The state is changed to *straight* when the arm is detected in a straight position for the first time. When the arm is bent below a certain angle while the state is *straight*, the state is updated to *bent down*. Once the arm angle is sufficiently small and the state is *bent down*, the arm's state is changed to *into the body*, which indicates that the hand is close enough to the body for a push-up, and the arm extension process can begin. Once the arm angle is sufficiently large, and the current state is *into the body*, the state is changed to *bent up*. Lastly, when the arm is straight and the current state is *bent up*, the state machine will also return to the *straight* state.

To give the user feedback on the back straightness, a separate state machine is used for the back. The back state machine consists of the three states depicted in figure 15. It begins in the initial state, *init state*. As for the arm state machine, the current back state depends on the back angle and its previous state. The back angle is calculated between the shoulder's, hip's, and knee's landmarks. When a straight back is detected while in the *init state*, the state is changed to *straight*. If the back is detected as bent during the straight state, the state is updated to be *bent*. The back is considered bent if the back angle is either above a predefined maximum threshold or below a predefined minimum threshold. When the back is again considered straight while in the *bent* state, the state is changed to be *straight*. The calculations are performed on either the left or the right side, depending on which landmarks are visible.

To track a push-up, both state machines for arm and back are combined into a parent state machine, `PushUp` state machine, which is illustrated in figure 16. The push-up state machine considers two instances of the `Hand` class state machine, one for the right arm and one for the left arm. The push-up begins in the *none* state. On detecting a person in a correct upwards pose, the state of the push-up state machine is changed to *up*. All the relevant states are considered to determine a correct upward pose, where both the arm(s) and back must be straight. From now on, the state machine can switch the state only if the arm(s) switches its state and the back is still straight. When the state machine is in the *up* state, and the arm(s) switches to the *bent down* state while the back is still straight, the push-up state machine switches to the *going down* state.

Similarly, when the arm(s) switches to the *into the body* state, the push-up can switch to the *down* state, and so on. The states can only be switched in the same order as is shown in figure 16. This means that if the user bents their back during, for instance, state *going down* and continues the push-up, the repetition will not be counted by the system. In this case, the user must repeat the full push-up to start counting repetitions again.

3.2.4 Exercise Logic

The Exercise class is an abstraction of a workout routine. In this project, this instance represents push-up logic. The push-up Exercise instance will instantiate the state machines mentioned in section 3.2.3.

All detected body parts are used to compute the current state of the push-up. The push-up instance of Exercise is used to organize the high-level structure of the exercise states and keep track of variables related to the exercise. The instance first checks how many arms are visible based on the input parameters, then updates the states of the relevant arm(s) and checks the state of the back to identify whether the posture is good. All these states are combined into one state in an instance of the PushUp class, which keeps track of the states of the whole push-up exercise. The counter for push-up repetitions in the instance of the Exercise class will increment the repetition counter every time the instance of the PushUp class reaches a state which represents the user having straight arms, a straight back, and having had arms bent on the way up in a previous state.

3.2.5 Region of Interest

A region of interest is used to optimize the system and prevent the model from detecting people other than the person that uses the system. When the first push-up is counted, the ROI class calculates the coordinates of four points in the image, becoming the corners of the region of interest. The coordinates are calculated based on the coordinates of the maximum and minimum landmark locations detected in the image. This means that the region of interest will change based on how close the user is to the camera after the first repetition. When the region of interest has been detected, it will be applied to each of the following frames before the MediaPipe landmark detection model gets access. Figure 5 shows the initial image before the first repetition is detected, while figure 6 demonstrates an example of an image where the region of interest is applied. The ROI class was based on an example code snippet from *learningaboutelectronics*[28].

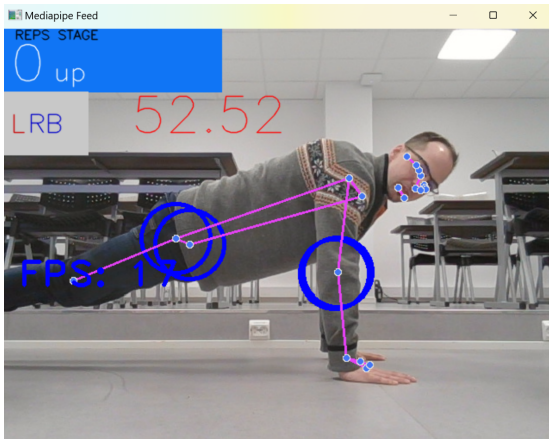

Figure 5: Before ROI locking

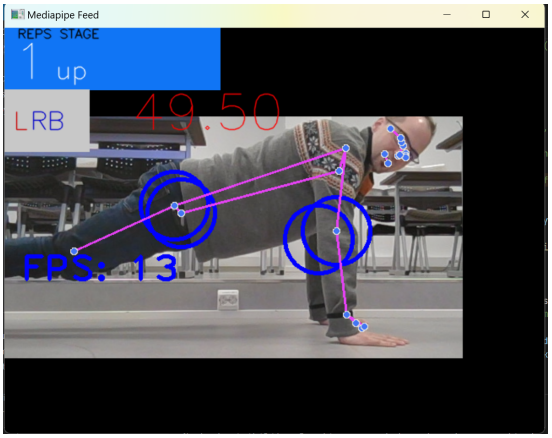

Figure 6: After ROI locking

3.2.6 Frame Per Second Injector

A *frames per second* (FPS) counter on the processed image was included as a measurement device for development. The FPS injector implementation was based on the implementation from CVZone[17] and modified to fit our project better. The FPS injector let the developers more easily assess how effective the various optimizations of the project had been at a glance. An optimization was deemed unnecessary if it had no noticeable effects and had other drawbacks.

3.2.7 User Interface

The user interface is intended as a visual aid for the user. It is rudimentary but effective. Four potential boxes are displayed on the screen, along with a timer and an FPS counter. The top box, in blue, represents the number of push-up repetitions performed in the current workout session, followed by the current state of the push-up being executed. Examples of this can be seen in the blue box in figure 5, figure 6, figure 9, among others.

The second box from the top, in gray, is a representation of the visibility of the left arm, the visibility of the right arm, and back straightness. Left arm visibility is represented by an “L”, right arm visibility is represented by an “R”, and back straightness is represented by a “B”. When the visibility factors are visible, the text is displayed in blue. If they are not visible, the text turns red. For the back straightness, the “B” will turn red if the back is bent out of good form and blue if the back is within acceptable angles. Examples of this can be seen in the same figures as the figures for the first box.

The third box from the top, in red, represents when the user's legs are not detected. A `Posture Abort!` will trigger if neither of the user's knees is detected. The `Posture Abort!` is intended as a warning to let the user know their legs are not visible and that the system cannot track their posture. Hitting this trigger will force the system's back straightness mechanism into the "straight" state, allowing the user to continue tracking push-ups despite the system being unable to detect their posture. Examples of this warning box can be seen in figure 4 and figure 11.

The fourth box from the top, in dust pink, represents the users back straightness being outside the recommended ranges. This box will only trigger if the user's posture is available for detection (all required landmark key points and body parts are detected) and the user's back state is bent. Examples of the bad posture warning can be seen in figure 7 and figure 8, with correction of the posture seen in figure 9.

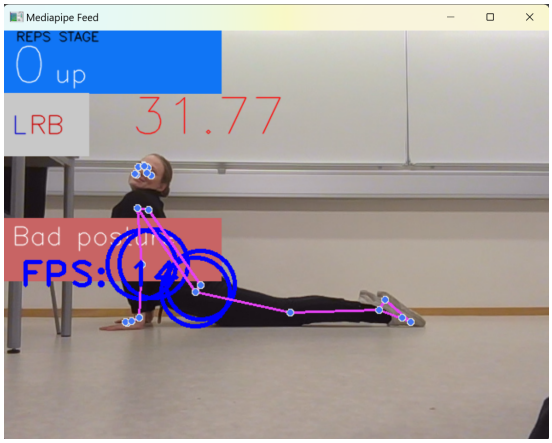

Figure 7: Example of a push-up attempt with bad posture in an up-dog pose

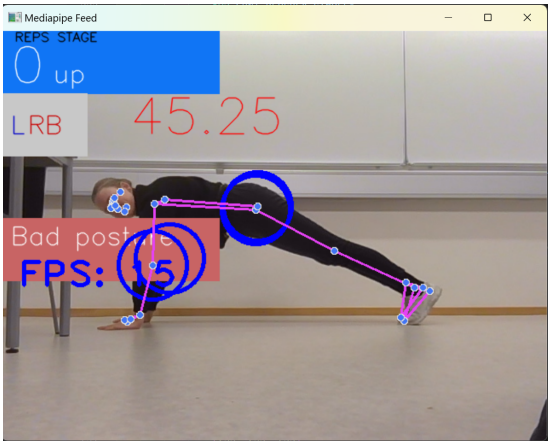

Figure 8: Example of a push-up attempt with bad posture in a partial down dog pose

The timer, in red, just underneath the top box, represents the time left in the current workout session. This was implemented as a proof of concept for a timed workout session, where the user has a certain amount of time to do push-ups before the timer runs out, and the current progress is logged. Examples of the timer in action can be seen in the same figures as for the figures for the first box.

The FPS counter, in blue, slightly occupying the space of the bad posture box, displays the current frames per second of video stream after all image processing is done. Examples of this FPS counter can be seen in the same figures as for the first box.

3.2.8 Data Storage

The data is stored in a database to enable users to track their progress using our system. When the workout session is over, the value of the repetition counter and the date and time when the workout session ended is saved and stored in an SQLite database. To add new data to the database, a python script is used. Later, another program module reads the data stored in the database and represents it in a plot. In figure 17, an example of this kind of historical data plot is presented.

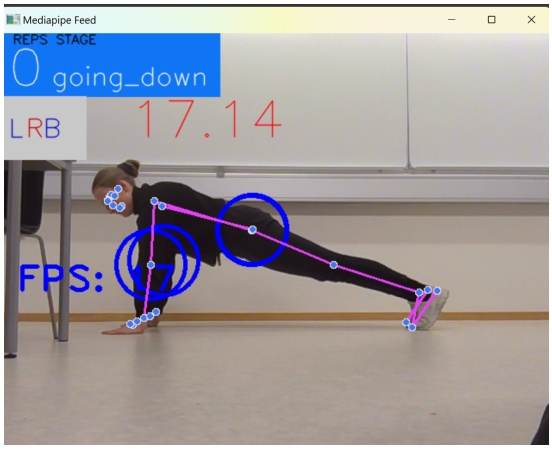

Figure 9: Example of a push-up attempt with good posture

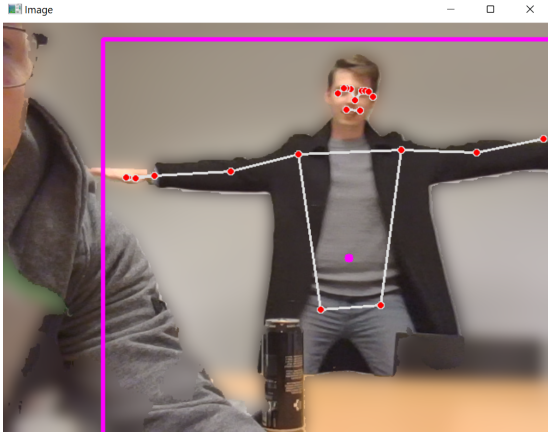

Figure 10: Example of a bounding box with variable width, dynamically adjusting to the user's arm.

4 Results and Discussion

4.1 Challenges, Cut Content, and Partial Implementations

This project has been marred in design issues due to an initial lack of machine vision and machine learning knowledge. Several research paths were explored to establish feasible solutions for the project. Several roadblocks during preliminary research (section 2) made the initial implementation of the project disjointed. As the research progressed, new roadblocks were encountered. New roadblocks lead to more research, leading to more roadblocks. Each new roadblock made the project's general path more visible as feasibility analyses were completed.

4.1.1 Data Collection and Data Set

Machine learning approaches were considered during the development of the project. It was theorized that this approach would require a data set containing workout routines combined with an unknown element yet to be detected. The initial data set plan was to film group members doing push-ups. Research into data set sizes revealed that the number of push-up videos required was unreasonably high, and that plan was abandoned. A data set that could be used to train a model was found online[3]. YOLO5 was considered, but as mentioned in section 2.2, this was abandoned.

Another attempt using Keras was attempted. The landmark points detected using MediaPipe were used to train a model. This test implementation is discussed further in section 4.1.2.

Discussions with the professor concluded that a machine-learning approach was not the best solution for this project, and all data set plans were abandoned.

4.1.2 Further Down the Hype Word Rabbit Hole

The machine learning research process was described in section section 2.5. Data sets were generated by using MadiaPipe, storing key points, and using those key points in testing. Also, an issue arose with the 3D positioning and mapping. In the training phase, an issue was recognized where the slightest changes in the angle between the human pose and the camera used to detect the pose radically reduced the model's

accuracy. It is significantly harder to create a model that will detect human pose from various angles, especially if the camera angle may differ in vertical and horizontal directions.

4.1.3 CVZone

During the early stages of the project, the entire project was rewritten to work with the CVZone[17] wrapper for MediaPipe and OpenCV. This rewrite was intended to give better type annotation for the project, make it less tedious to identify which data structures were used in the project, and allow for static typing in the program. While this made the project easier to work with on a code-implementation level, the CVZone project is not as popular as the two libraries it wraps. This made finding example code very difficult.

The wrapper was analyzed to determine if it was worth future use, section 2.7. The documentation for CVZone was very limited compared to both MediaPIpe and OpenCV. Further, CVZone was not updated regularly and lagged several months behind the current versions of both MediaPipe and OpenCV. A choice was made to drop CVZone as a wrapper and instead use CVZone as inspiration for some of the functionality needed in the project.

4.1.4 Background Segmentation

It was discovered early in the implementation that a problem using the MediaPipe landmark model was that the model could get distracted if several people were in the image simultaneously. This was reproducible if the model detected a person in the image followed by a new person walking into the frame or becoming visible. The model could get distracted and attempt to combine the two people into one detected model. This could be a problem for the project's scope since the model could get distracted by other people that could appear in the background and incorrectly start detecting repetitions of the active user.

The first solution to this problem was background segmentation. The first implementation of background segmentation extracted the background completely and turned it white, as demonstrated in figure 11. Parts of the implementation were borrowed from CVZone[17] and rewritten for the project's use case. While the background segmentation did alleviate some of the issues of the model being distracted by other people, it also caused two new issues. Firstly, it came at a slight performance cost, which was not a large issue but was still noticeable. Secondly, it wasn't easy to adjust the threshold correctly.

To solve this issue, several threshold logic implementations were tested. The one that worked best was where the amount of white in an image was compared to the other noise in an image as threshold metrics. The problem with this was that it would work very well for people working out in front of a white wall but would incorrectly adjust the threshold if there were dark objects in the scene (other than the person) that offset the threshold in the wrong direction. This led to the segmentation cutting off limbs when the person was far away.

A workaround was implemented where the background was blurred using Gaussian blur instead of being replaced with white. The blurred background implementation alleviated the issues, and the tracking worked well. Still, after rigorous testing of this implementation, it was determined that it was, at best, a net-neutral feature.

In most cases, this was a net negative because a limb could be considered a part of the background, which caused the model to lose track of a limb. The final version of the code before it was cut can be seen in listing 1. Figure 12 shows the segmentation in action using a blurred background with people behind the main actor actively attempting to "steal" the key point detection. Note the blurred arm; in this case, the arm is still detected, though this was not always the case.

```
Background segmentation
threshold = whiteness_offset(my_frame)
bg_image = cv2.GaussianBlur(my_frame, (55, 55), 0)
clean_img = segmenter.removeBG(
    my_frame, imgBg=bg_image, threshold=threshold
)
```

Listing 1: Background segmentation. The segmenter is an instance of a class inspired by the CVZone implementation of the same feature but modified to fit our use case

.

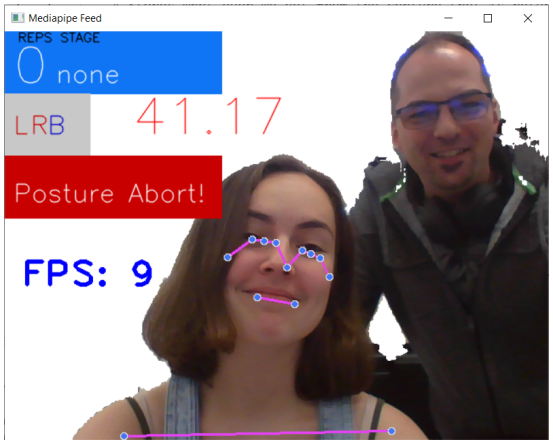

Figure 11: Example of background segmentation where the background is completely removed (turned white)

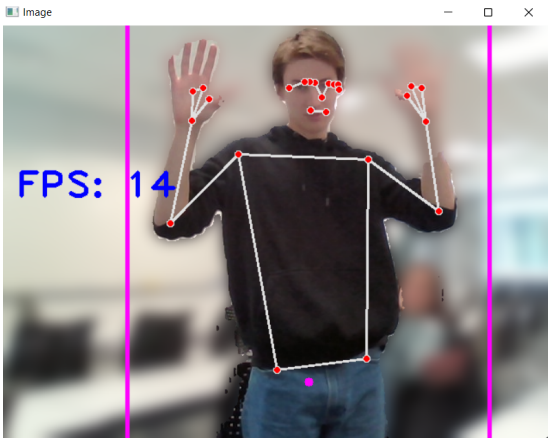

Figure 12: Example of background segmentation and bounding box working.

4.1.5 Bounding Box

Another solution to this problem that was tested was a bounding box implemented based on the same feature in CVZone[17]. The bounds of this bounding box were calculated based on a weighted center point calculated using all visible landmark key points. The box was wrapped around a fixed width combined with a weight of the combined delta x-values of the landmark key points. An example of the bounding box with fixed width can be seen in figure 12. An extra feature was added where the bounding box could extend if the user's hand extended beyond the bounding box. An example of this hand extension bounding box can be seen in figure 10. It was determined that this box did not add any value to the system's core functionality while adding extra computations to each frame. This was deemed a net loss for the project, and the feature was cut.

4.1.6 Total Body Angle Logic

The exercise logic was rewritten several times throughout the project's research phase. After detecting landmark key points using MediaPipe, several attempts were made to calculate matches for *good* or *bad* postures. Having every data point relate to every other as a large data set quickly became too cumbersome and static in its implementation. Every new feature required a rewrite of other calculations. The project aimed towards an extendable design, where it would be fairly simple to include new workout routines or

adjust existing ones without too much effort. To achieve this, the logic of detecting poses, posture, and repetitions was moved into a combination of different state machines. The manual landmark logic was deprecated and removed as state machines took over.

4.2 Optimizations

Throughout the project, several attempts were made to optimize the implementation while maintaining a sufficient detection of key points. The main optimization that was found to "kill two birds with one stone" was ROI locking. This mostly alleviated the issue of the model being distracted when several people are in the same image, but also slightly improved performance in general, as there were fewer intricate details to process.

An attempt was made to convert the entire implementation to gray-scale, but this proved to make MediaPipes landmark key point detection very inaccurate, so this optimization was cut.

As mentioned in section 4.1.5, an optimization was made by removing the bounding box. Despite looking fancy and acting as a reference box around a detected person, the bounding box did not add much value to the actual core functionality of the system.

4.3 Push-up Variations and Disability Support

Due to how the state machines work together, in addition to normal push-ups, the push-up routine also supports knee push-ups, wall push-ups, and, in theory, handstand push-ups - though the latter has never been tested due to the physical limitations of the system actors.

The left-hand and right-hand visibility check and the back straightness logic allow one limb in each set of legs and arms to be missing. The project was not intentionally designed to be inclusive of physically disabled people. Still, how the state machine logic was implemented around body part visibility allows for disabilities limited by the person having only one arm and one leg.

4.4 Historical Data and Progress Review

To allow the user to measure their progress over time, the back-end simulates a workout session by starting an arbitrary length timer when the `main.py` file is started. This timer can, for instance, be seen in figure 9. Once this timer lapses, the workout session is considered completed, which triggers relevant data after a session to be stored in a data storage. In the current implementation, the amount of repetitions done by the user during the workout and the workout date-time is stored in a `SQLite` database. The user can then access the historical data by running the `data_report.py` script, which acts as a separate interface. The script gets the data from the data storage and displays it to the user in a graph plot, as shown in figure 17. This allows the user to track their progress over time.

Storing data into `SQLite` database was a simple and reliable solution, based on the research described in section 2.10. Currently, the data is stored in one table, but in the future, that can be extended to store different exercises. An SQL database seems like overdoing it since only one table is used. At the same time, no performance issues were experienced using the database. Therefore using the `SQLite` was a viable short-term development solution.

5 Conclusion and Future Work

> How did it get so late so soon? It's night before it's afternoon. December is here before it's June. My goodness how the time has flewn. How did it get so late so soon?
>
> Dr Seuss

5.1 Time Spent on Planning and Test Implementation

The project was off to a rocky start due to the developers having too little insight into computer vision. It was easy to latch on to *hype words* without knowing the impact of the baggage they came with. The initial design plan did not last long after the research phase started. Several new design plans sprung up and then crumbled during this research phase. A lot of time was dedicated to hammering the final design plan into shape early on in the project. This might have seemed like a waste of time, but once the final plan was in place, the rest of the project implementation went much smoother.

5.2 Project Scope Assessment

Reviewing the project scope, the full scope of the system was to recognize the form and posture of a person and track how many push-ups the person took. A secondary goal was to give feedback on the user's posture while doing the push-ups.

As a proof of concept, the project itself is working and fulfilling its scope. The user's form and posture are correctly detected, and the push-up repetitions are tracked. In addition to tracking, the repetitions are also logged over time and are available for the user to review either post-workout or further down the line for statistical purposes. The system also gives users feedback on their posture live during the workout. The groundwork is laid out, and there is posture detection, and bad posture warning is in active use, shown visually in the camera feed. If a bad posture is detected, the repetitions will not be counted until the user corrects the posture.

5.3 Further Work Partially Within the Project Scope

Further work could improve the logic of what defines a good or bad pose. As some of the research mentioned in section 2.8, many different variables make up a good or a bad pose for push-up exercises. The implementation in this project's logic attempted to cover as many good poses as possible where the good poses were not in direct conflict with each other. For the bad poses, back curvature was used as a factor. There are many other variables, such as neck position, the angle of arms and hands compared to the orientation of the rest of the body, and how the elbows are aligned when going down or up during a push-up. Many of these were difficult to detect when the pose logic was built upon MediaPipe's key point landmarks. Given enough time and research, an alternative to MediaPipe might be able to solve this in a better way. Alternately a more elaborate manual logic that also considers rotation within the state machines might work, though this is only speculation.

5.4 Limitations or Further Work Outside of the Project Scope

Outside of the project scope, several features could be implemented.

Other students tested the project in several demo sessions. One issue prevalent throughout the project's demo sessions was that users would start the first push-up repetition from a bad camera angle. This would trigger the ROI lock-in when the user was in this awkward position. A solution to this was suggested

by the course professor and guide, Nadia, where some guidelines could be overlaid on the camera feed to indicate where the user should be positioned before starting their workout session.

Several refactors could be done throughout the project to make data passing more streamlined. This was never a goal of the project itself, as refactoring is not something that should take place during the proof-of-concept phase and is a process that can become a time sink. The body part detection class, is one of the major refactors that could be done. Currently, it detects individual body parts. Instead, it could be rewritten into a human skeleton class and take in the key points as updated arguments. This entire skeleton class could be sent to the state machines instead of passing in individual body-part tuples.

The region of interest locking is static. This means the user cannot move after the ROI is locked in. A solution to this was discussed, where the ROI border points had a delayed motion effect triggered by a user moving too close to the edge of the ROI borders. The intent would be that the border would grow slightly over time if the user's landmark key points repeatedly approached the edged of the current ROI border. This would allow the ROI to float around the user's active area of the video feed, minimizing the user's risk of moving outside the ROI area.

As mentioned in section 4.2, there were many optimization passes in this project, and elements that were not beneficial to the project's main goals were removed. There are probably several elements that are not yet considered within the system that can be further optimized.

The historical data measure aspect of the project could be extended to include several other factors, and the arbitrary timer used to limit a workout session could be customized. This extendibility allows for much deeper analytical value for the user.

Defining and abstracting the variables that make a good or bad posture during a push-up routine took some effort. The abstractions used for poses in the system currently support the most common push-up variations, such as wall push-ups, knee push-ups, normal push-ups, and hand-stand push-ups. However, there is a major limitation to how the MediaPipe landmark model reads landmark key points. When considering elbow and wrist rotation compared to the orientation of the torso and shoulders, the MediaPipe landmarks become too jittery to measure good arm rotation posture accurately. Further work on this could be to replace the MediaPipe model or attempt to interpolate some rotational points for the landmark skeleton.

References

[1] *(7) Post | LinkedIn*. [Online; accessed 3. Oct. 2022]. Oct. 2022. URL: https://www.linkedin.com/posts/riteshkanjee_yolov7-computervision-opencv-activity-6975069428346466304-y3sm/?utm_source=share&utm_medium=member_android.

[2] *Infinity AI*. [Online; accessed 8. Nov. 2022]. Oct. 2022. URL: https://infinity.ai.

[3] *Fitness Basic Dataset*. [Online; accessed 3. Oct. 2022]. Oct. 2022. URL: https://marketplace.infinity.ai/products/fitness-basic-dataset.

[4] lp6m. *yolov5s_android*. [Online; accessed 8. Nov. 2022]. June 2022. URL: https://github.com/lp6m/yolov5s_android.

[5] ultralytics. *yolov5*. [Online; accessed 8. Nov. 2022]. Nov. 2022. URL: https://github.com/ultralytics/yolov5.

[6] lp6m. *yolov5s_android*. [Online; accessed 8. Nov. 2022]. June 2022. URL: https://github.com/lp6m/yolov5s_android/tree/master/benchmark.

[7] ultralytics. *yolov5*. [Online; accessed 8. Nov. 2022]. Nov. 2020. URL: https://github.com/ultralytics/yolov5/issues/1276.

[8] *Pose detection | ML Kit | Google Developers*. [Online; accessed 3. Oct. 2022]. Sept. 2022. URL: https://developers.google.com/ml-kit/vision/pose-detection.

[9] *Machine Learning with NumPy, pandas, scikit-learn, and More - Learn Interactively*. [Online; accessed 21. Nov. 2022]. Nov. 2022. URL: https://www.educative.io/courses/machine-learning-numpy-pandas-scikit-learn.

[10] *Machine Learning with Keras and TensorFlow | Frontend Masters*. [Online; accessed 21. Nov. 2022]. Mar. 2020. URL: https://frontendmasters.com/courses/practical-machine-learning.

[11] Keras Team. *Keras: the Python deep learning API*. [Online; accessed 21. Nov. 2022]. Nov. 2022. URL: https://keras.io.

[12] *TensorFlow*. [Online; accessed 21. Nov. 2022]. Nov. 2022. URL: https://www.tensorflow.org.

[13] opencv. *opencv*. [Online; accessed 3. Oct. 2022]. Oct. 2022. URL: https://github.com/opencv/opencv.

[14] Contributors to Wikimedia projects. *OpenCV - Wikipedia*. [Online; accessed 15. Nov. 2022]. Oct. 2022. URL: https://en.wikipedia.org/w/index.php?title=OpenCV&oldid=1113479408.

[15] google. *mediapipe*. [Online; accessed 3. Oct. 2022]. Oct. 2022. URL: https://github.com/google/mediapipe.

[16] *Pose*. [Online; accessed 13. Nov. 2022]. Nov. 2022. URL: https://google.github.io/mediapipe/solutions/pose.html.

[17] *cvzone - Overview*. [Online; accessed 3. Oct. 2022]. Oct. 2022. URL: https://github.com/cvzone.

[18] cvzone. *cvzone*. [Online; accessed 15. Nov. 2022]. Apr. 2022. URL: https://github.com/cvzone/cvzone/commits/master.

[19] Rachel Grice. "The Best Push-Up Variations for Beginner, Intermediate and Advanced". In: *LIVESTRONG* (Oct. 2022). URL: https://www.livestrong.com/article/13731958-best-push-up-variations.

[20] Annie Malaythong. *Part 1: Exploring a Proper Pushup Form and Technique*. [Online; accessed 15. Nov. 2022]. Feb. 2021. URL: https://blog.nasm.org/nasm-guide-to-push-ups/form-and-technique.

[21] Cassie Dionne. "Avoid These Push Up Mistakes to Protect Your Wrists - Breaking Muscle". In: *Breaking Muscle* (Jan. 2016). [Online; accessed 15. Nov. 2022]. URL: https://breakingmuscle.com/avoid-these-push-up-mistakes-to-protect-your-wrists.

[22] *Designing State Machines using Python [A Quick Guide] - AskPython*. [Online; accessed 15. Nov. 2022]. Apr. 2022. URL: https://www.askpython.com/python-modules/state-machines-python.

[23] Vaibhav Singh. "Should I use DB to store file ? - Vaibhav Singh - Medium". In: *Medium* (Dec. 2021). URL: https://medium.com/@vaibhav0109/should-i-use-db-to-store-file-410ee22268c7.

[24] *Advantages and Disadvantages of SQL - GeeksforGeeks*. [Online; accessed 21. Nov. 2022]. July 2021. URL: https://www.geeksforgeeks.org/advantages-and-disadvantages-of-sql.

[25] *Use SQLite for a simple, lightweight database option*. [Online; accessed 21. Nov. 2022]. June 2007. URL: https://www.techrepublic.com/article/use-sqlite-for-a-simple-lightweight-database-option.

[26] Ankush. "Top 12 Open Source Database Software for Your Next Project". In: *Geekflare* (Oct. 2022). URL: https://geekflare.com/open-source-database.

[27] Nicholas Renotte. *AI Pose Estimation with Python and MediaPipe | Plus AI Gym Tracker Project*. [Online; accessed 16. Nov. 2022]. Apr. 2021. URL: https://www.youtube.com/watch?v=06TE_U21FK4.

[28] *How to Create a Region of Interest in an Image in Python using OpenCV*. [Online; accessed 15. Nov. 2022]. Nov. 2022. URL: http://www.learningaboutelectronics.com/Articles/Region-of-interest-in-an-image-Python-OpenCV.php.

A UML

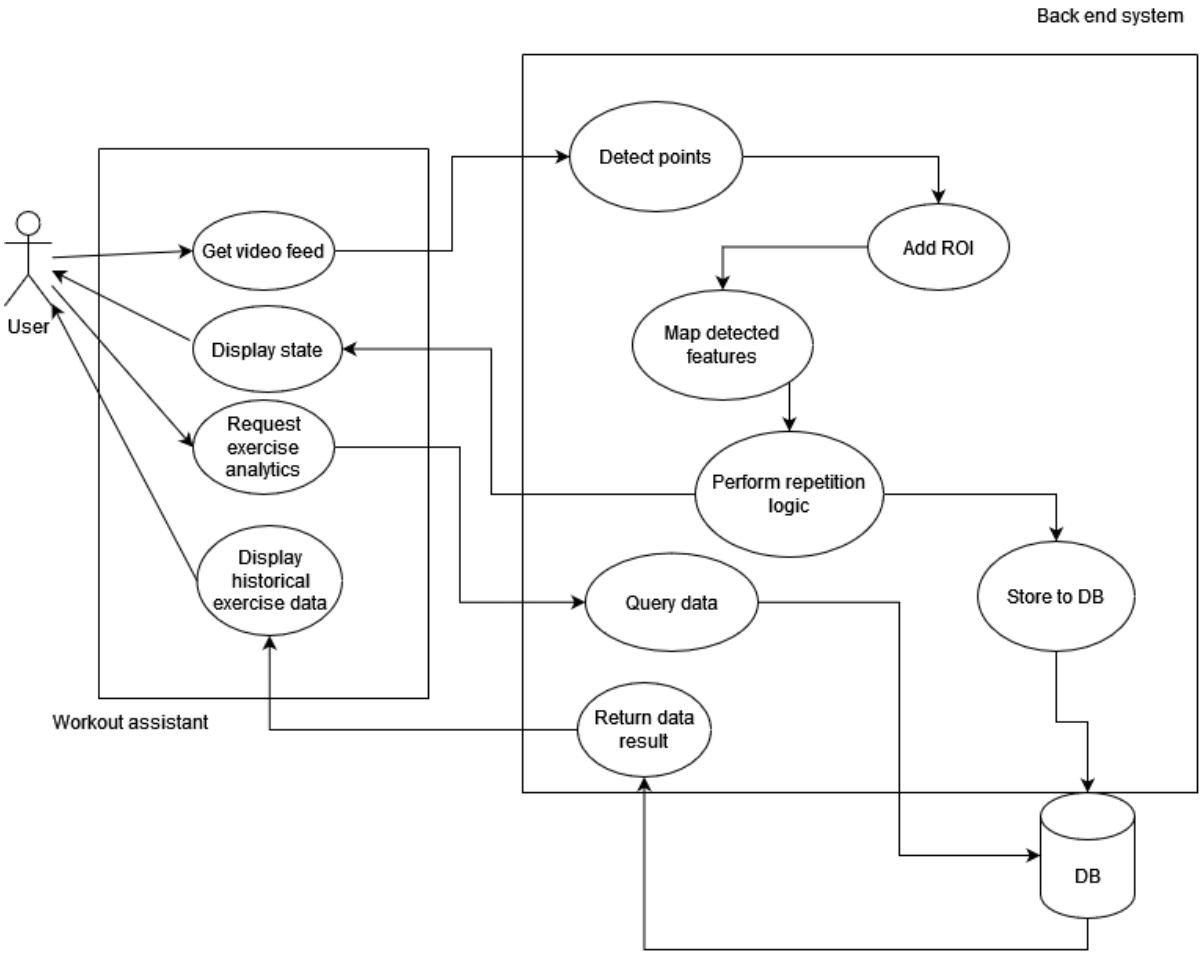

Figure 13: More detailed UML Use case description diagram

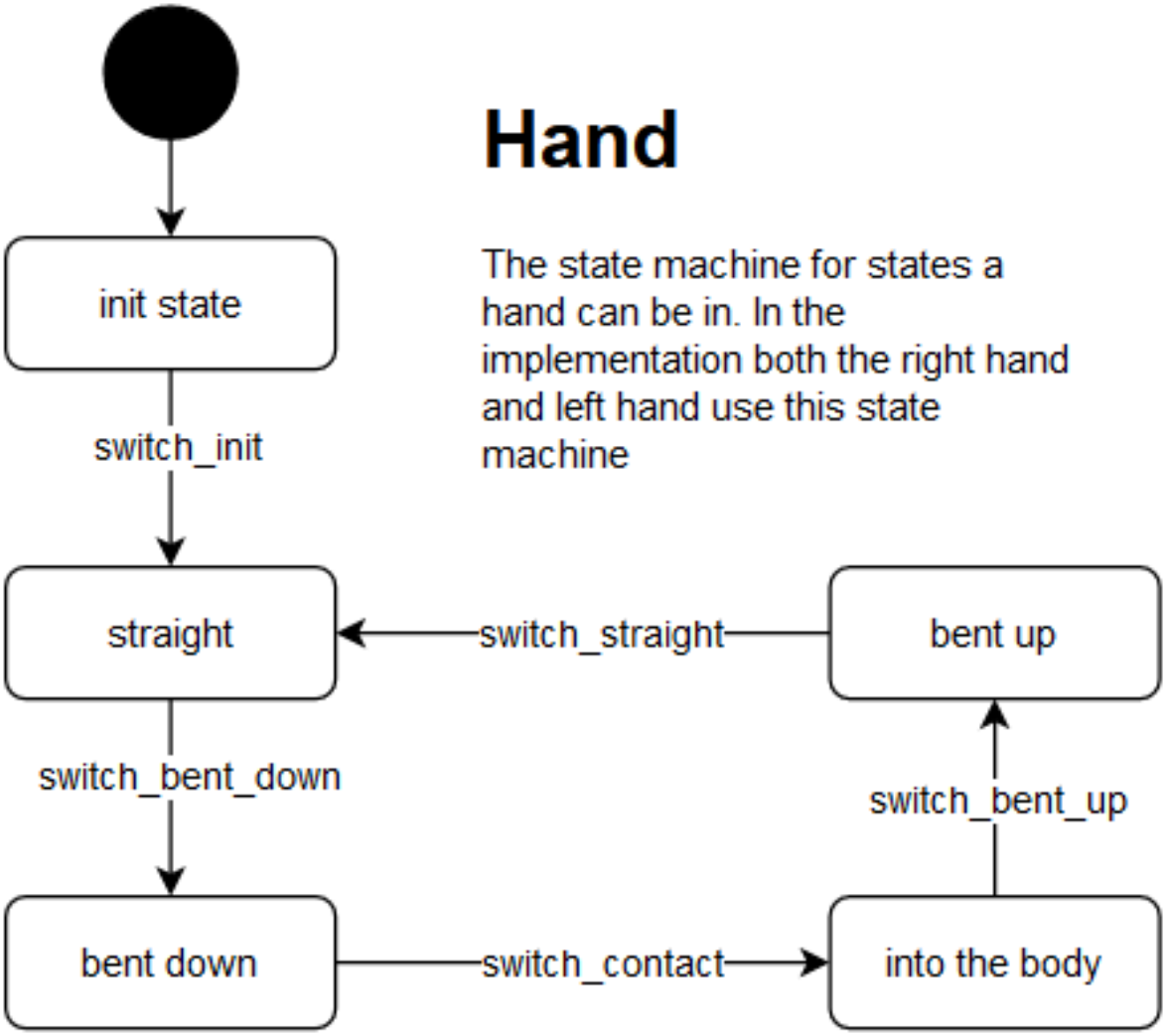

Figure 14: UML state machine for each of the hands

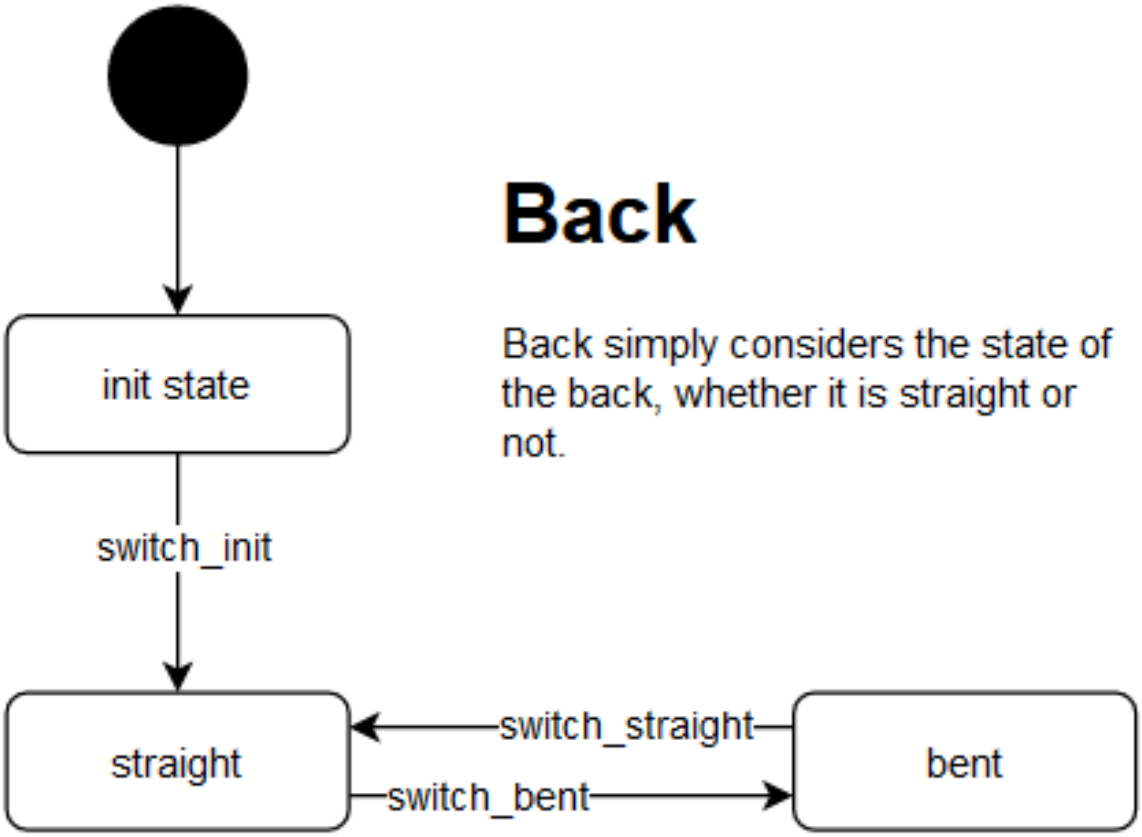

Figure 15: UML state machine of the back

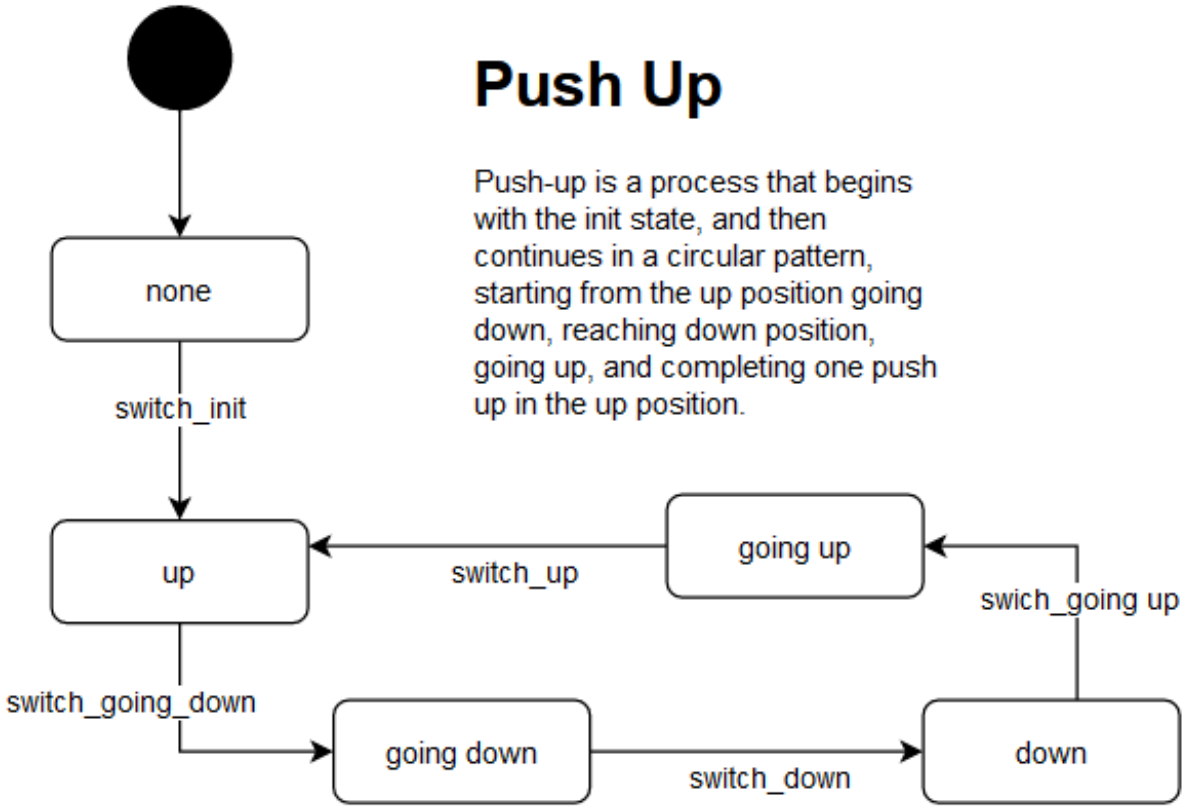

Figure 16: UML state machine of the push-up

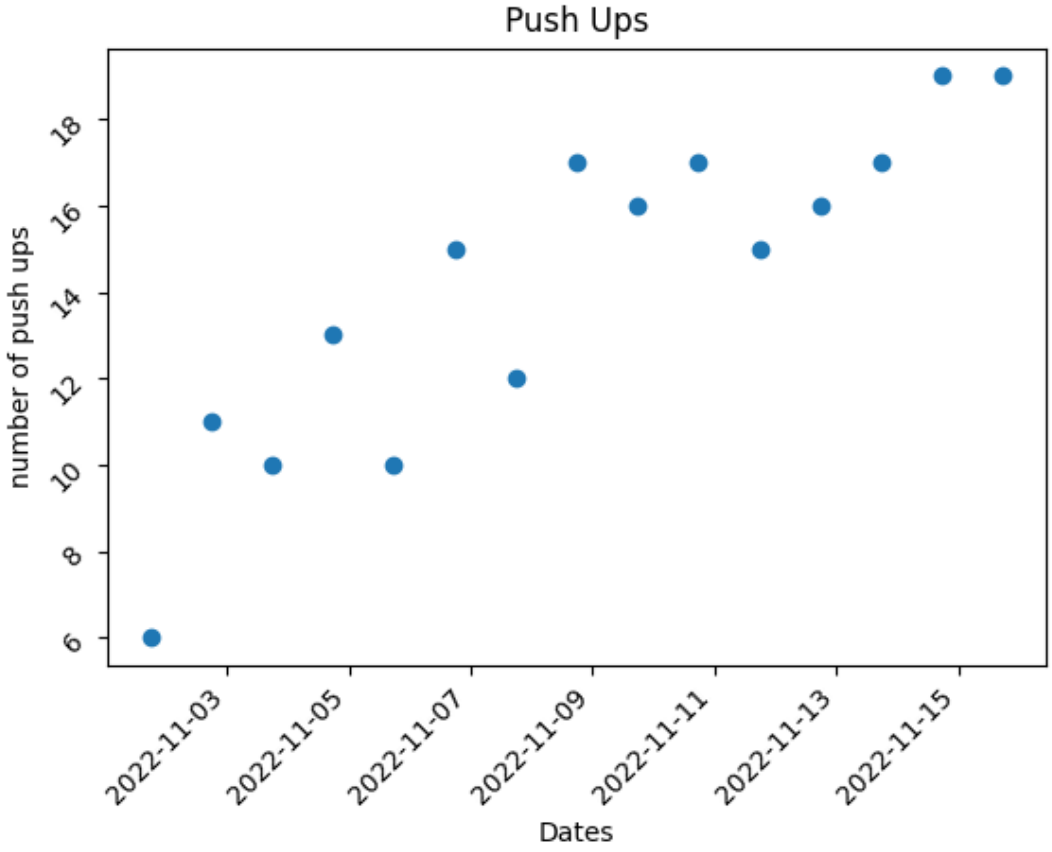

Figure 17: Plot of historical data