Final Report

Team Name: PhysicalQt

Team members: Vijay Rajagopal, Zhenning Yang

Introduction

With recent advancements in technology across all fields, the usage of these new technologies in more and more diverse fields is what is going to drive innovation for the upcoming decade. In this project, we attempt to apply some of these aforementioned technologies (machine learning mainly) to a medical problem. This medical problem is the fact that many people, when they get a physical injury, need physical therapy in order to help them get back to working order. Because of this great importance on physical therapy, it is important that a given exercise needs to be completed correctly. Although there exists medical professionals who focus on advising patients during their physical therapy sessions, this avenue might be unavailable to a lot of patients due to the costs or lack of medical coverage. In order to combat this, we have developed an application that can tell a user if they are doing a given exercise correctly. This application — which we call PhysicalQt — uses a machine learning model in order to conduct the evaluation and requires no additional hardware other than a web camera. Throughout our time developing PhysicalQt, there were many changes to the project timeline (more discussed in the "Project Management" section), but we were able to create a functional minimum viable application that reflected our initial objectives.

Customer Value (no change)

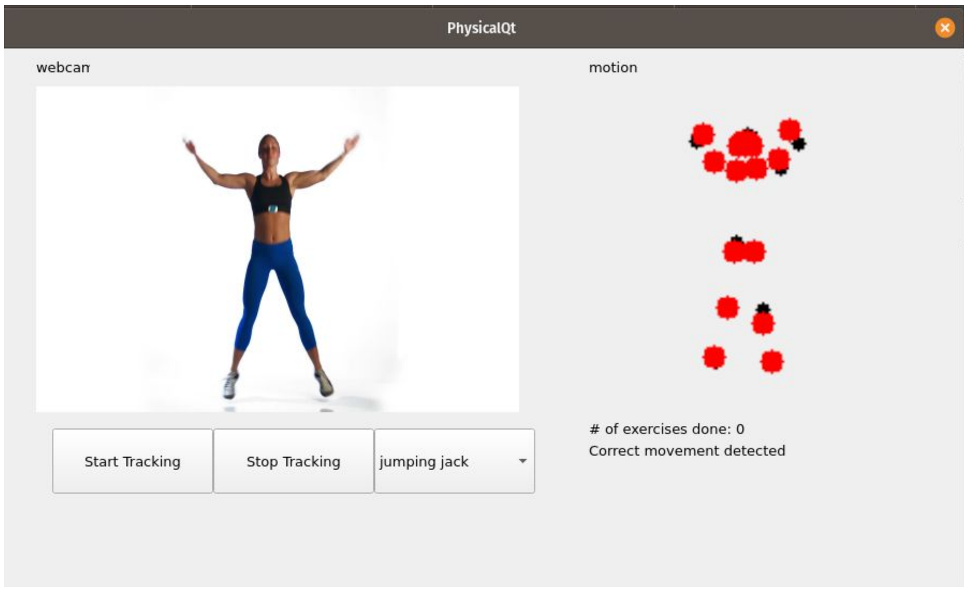
Our main customer would be an out-patient that has been assigned various physical therapy exercises by their physical therapist or doctor. These patients could likely have never done such an exercise, and when they are left to their own devices to do that exercise they could execute it incorrectly and could hinder their recovery phase.

Technology

PhysicalQt mainly relied on Qt5, a C++ powered library, for the GUI and user interaction. Moreover, PhysicalQt uses a Python API wrapper called PyQt5 to rapidly prototype and develop GUI interactions in the end product. The ability to detect if a person's exercise movement required a pose estimation model that is able to take in an image frame from a camera, compute the general pose of the figure, and output coordinates in 2D space. All of this has to happen in real time to provide the best user experience. For this reason, we decided to use a network called PoseNet, a 2015 neural network able to run with low inference speed and average accuracy. The implementation of Posenet we used was written in Python with a machine learning library called Tensorflow.

In our final product, we were able to create a simple interface which could process a given movement through Posenet and output an accurate pose. We were also able to create a simple GUI interface to showcase the aforementioned Posenet process. Finally, we created many utilities that enabled us to “evaluate” a pose and see if an exercise is being done correctly.

The screenshot below showcases the GUI and Posenet working to generate a user-generated pose:



# of exercises done: 0
Correct movement detected

The screenshot below is a showcase of the aforementioned “utilities” that were used to correct and evaluate a given pose:

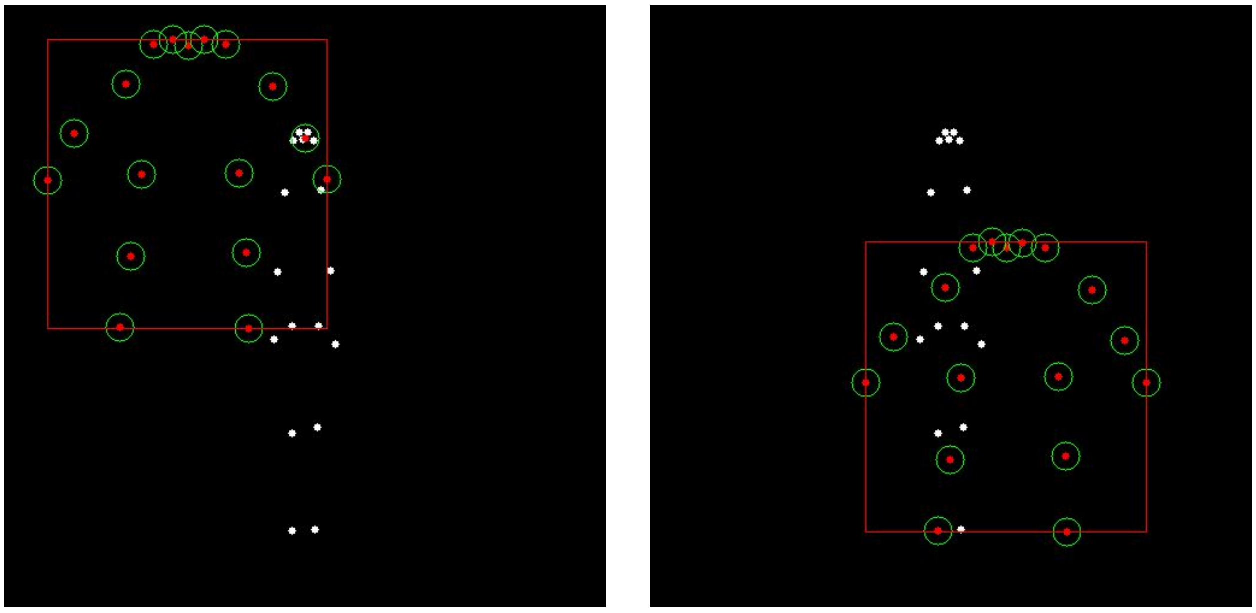


Diagram 1. Left image shows no corrections applied to an image; Right image shows both "scaling" and "binding" corrections applied to pose. Pose 1 is white and Pose 2 is red.

One important aspect of the "corrections" in the utility functions is the ability to correctly match up two different poses to each other in order to properly compare them. This becomes difficult when the two poses are from two differently sized input images, which causes a scaling issue. Our solution to this problem was the following:

- Normalization the coordinates of any pose
- Shift one pose to the another pose by "binding" one commong joint (for our case, we used "right ankle")
- Generate a bounding box from each pose and scale based on the "scaling factor"

As for the actual evaluation, it is a simple method of computing a "tolerance" around each joint of a given pose and then measuring if a corresponding joint in another pose is within the distance of the first pose's joint's tolerance. After going through each joint, a score is computed and if it is higher than a threshold, the pose will be counted as "correct". In the image above, the tolerances are displayed as green circles around each joint.

Team

Both Zhenning and Vijay have extensive experience in both Python and C++ as well as experience in other applications of Tensorflow that could help in implementing the models needed for pose estimation. Due to this, our time was spent delegating responsibilities regarding implementing both the GUI-logic and evaluation of the Posenet output.

Project Management

This was the original "Project Management" section in our proposal document:

Week:

- **1:** Setup the Qt project and have all the prerequisites installed
- **2:** Find and test out the fastest public version of PoseNet and OpenPose
- **3:** Start writing the algorithm to determine a good movement setup for **one** exercise; Start building GUI at the same time; Complete iteration 1 status report
- **4:** Continue building GUI and the pose estimation logic; Look to implement other more specific estimation models (hand position estimation, feet position, etc.)
- **5:** Extend pose estimation logic to other types of exercises; Complete iteration 2 status report
- **6:** Refine GUI elements to look sleek and modern; Run through the flow of the application
- **7:** Complete intended goals and work on resolving any bugs or UX issues

- **8:** Create a project report and continue debugging the application
- **9:** Present project to class

Although we had high hopes, the majority of the objectives in the later half of the semester had to be delayed or discarded due to the fact that we had some difficulty setting up Posenet, getting ourselves comfortable with PyQt5, and creating the logic for a robust evaluation script. Due to these problems, objectives such as “Extend[ing] pose estimation logic to other types of exercises” were not able to be completed. Rather, we decided to focus on getting a single exercise (Jumping Jacks) working well.

Reflection

What went well

Our team management is efficient. Since the flow of the project is very clear for us at the beginning. We were able to develop a plan and split the project into two parts -- User interface and Motion detection, so then we could work parallelly as we discussed above. Towards the end of development, we have to combine our work, then test and debug together. Although we couldn't have face-to-face meetings, we communicated frequently to report progress and issues, most importantly, to avoid conflict on GitHub.

What did not go well

We still need to improve our documentation. Both of us don’t comment code and we don’t have physical documentation such as README files or something, to keep track of individual progress. It took way too long to get the UI working, the package we chose is called Pyqt5 and there are not many resources for webcam usage, which makes the UI part surprisingly hard.

Success?

By developing this application, we get more familiar with some of the useful tools such as Docker, PyQt5 and Tensorflow Lite. Our team management is efficient. We worked parallelly by dividing the project into two parts: User Interface and motion detection; then we combined all pieces at later development of the project. As a result, we can propose a functional and complete application. One single exercise -- Jumping Jack is being used for texting and more exercises can be easily added to the program. Of course, there are some adjustments we could make to give the users the best experience. For instance, using multithreading, place webcam and detection into different threads to make the program run smoother. Overall, we consider our final project a success.