RealSense Physiotherapy Guidance System (RPGS)
A home-based physiotherapy aid

Chua Wei Pin, He Cheng Hui, Nabilah Binti Abdul Raub
Engineering Science Programme, National University of Singapore

1. Abstract

The RealSense Physiotherapy Guidance System (RPGS) is a home-based physiotherapy aid that provides real time sensing and guidance for physiotherapy patients to complete their exercises correctly and most importantly, safely. It consists of three synergistic components: an Intel RealSense D435 3D camera, an wearable module consisting of an Arduino UNO and an MPU-6050 Six-Axis (Gyro + Accelerometer) MEMS MotionTracking™ Device, and an interactive GUI that provides feedback to the patient in real time. This report will discuss how the RPGS was engineered to provide the best experience and accurate posture detection. It will then discuss the usage of RPGS in supporting a proof-of-concept exercise, the Single Arm Raise and Rotation. The RPGS was able to detect errors in posture and identify specific errors for this exercise.

2. Keywords

Physiotherapist, exercise, posture, RealSense, OpenPose, Arduino, Accelerometer, deprojection

3. Current Problem

Physiotherapy[1] helps restore movement and function when someone is affected by injury, illness or disability. It is an important aspect of our lives because common injuries like a strained shoulder or back can easily be treated with physiotherapy. The Global Burden of Disease Study conducted in 2015 states that 74% of years living with disability in the world is closely related to health conditions where rehabilitation is beneficial. Physiotherapy is one of these possible rehabilitation methods. In physiotherapy treatment, patients see a physiotherapist for a checkup, and to learn and practice rehabilitation exercises from the physiotherapist. The patient is then supposed to head home to continue practicing those exercises for continued and consistent rehabilitation, until the next physiotherapy session. This is important[2] because maintaining the continued doing of exercises facilitates the rate of recovery and helps to build physical strength. However, without the supervision of a physiotherapist when performing exercises by themselves at home, patients have a higher chance of performing the exercises incorrectly[3]. This will very likely slow down progress or even be detrimental to the recovery process. In more extreme cases, it may even lead to injuries[4,5]; all of which are undesirable effects going against the very reason for physiotherapy itself.

Evidently, these undesirable effects may only manifest when a physiotherapist is not present to provide guidance to the patient and to correct their posture/actions.

Hence, we have decided to design the **RealSense Physiotherapy Guidance System (RPGS)**, a real-time sensing and interactive GUI designed to provide guidance to patients performing physiotherapy exercises at home. The RPGS provides guidance by warning the patient when they are performing the exercises incorrectly, and subsequently teaches them how to correct their incorrect posture/pose.

4. Product Overview

The RPGS is made up of three synergistic components: an Intel RealSense D435 3D camera[6], a wearable module consisting of an Arduino UNO[7] and an MPU-6050 Six-Axis (Gyro + Accelerometer) MEMS MotionTracking™ Device[8], and an interactive GUI that provides feedback to the patient in real time.

4.1 Intel RealSense D435

The Intel RealSense D435 (Figure 1) is a 3D camera that provides quality depth imaging of up to 10m through a pair of depth sensors that leverage on stereo imaging. It has a wide field of view in both the vertical and horizontal dimensions, and a global shutter that allows for accurate motion capture. The D435 will be connected to a laptop to allow the software component of the RPGS to process the live video feed. Further elaboration on the D435's role in RPGS will be in Section 7 along with discussing the core architecture.

Figure 1: Intel RealSense D435 camera with tripod

4.2 Wearable Module

The wearable module consists of an MPU6050 Accelerometer connected to an Arduino UNO. The entire set-up is contained within a 3D printed box (Figure 2) with velcro straps attached at the bottom. The velcro straps can be tightened around the user's limbs to detect for rotation. Figure 3 shows the patient wearing the module around their wrist, with the correct orientation being the USB cable comparatively nearer to the hand. A hole in the casing allows for a wired connection via a 4m-long USB-B cable to the laptop for data processing.

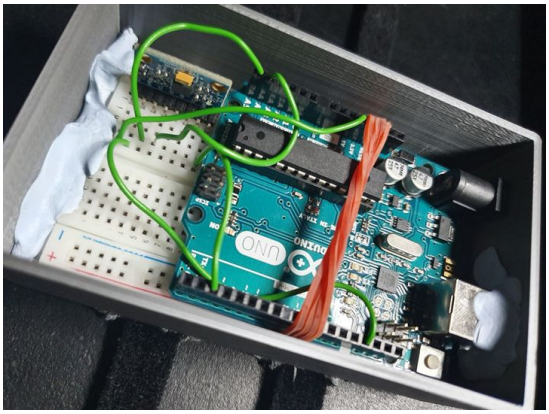

Figure 2: Contents within the casing

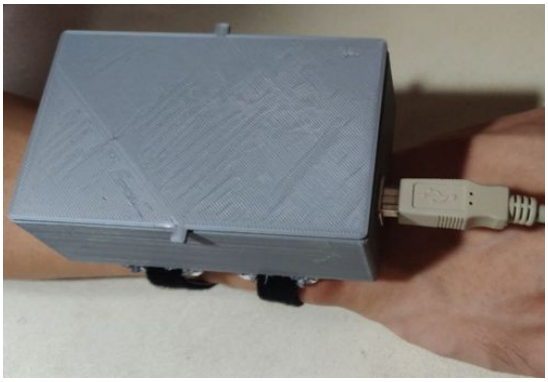

Figure 3: Wearable module strapped onto patient's wrist

According to how the accelerometer was aligned within the casing, the coordinate system of the entire wearable module is shown in Figure 4. Acceleration data will be recorded and normalized values for all 3 axes can allow for the determination of the direction of acceleration.

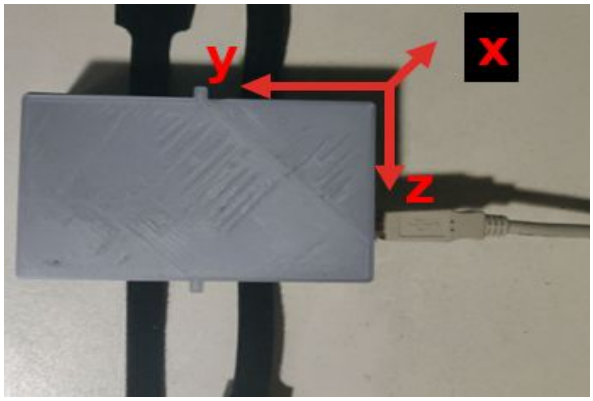

Figure 4: Cartesian coordinate system of the wearable module

4.3 Interactive GUI

An interactive GUI was created using the PyQt[9] platform and designed to be user-friendly, customisable to the patient's needs and abilities, and responsive in terms of feedback. The interactive GUI consists of two modes - the patient mode and the physiotherapist mode. The patient mode informs the patient on how to make corrections to the exercises whereas the physiotherapist mode allows for customisation of exercises for the patient by adjusting its level of difficulty.

Figure 5 shows the login page for the RPGS. Patients will have their own personal RPGS account with data concerning their exercises stored. Privacy is ensured through this personalised login function. The physiotherapist will have their own account as well, enabling them to take on an administrator role. This allows the physiotherapist to be able to monitor the progress of the patient and customise exercises to the patient's needs and abilities.

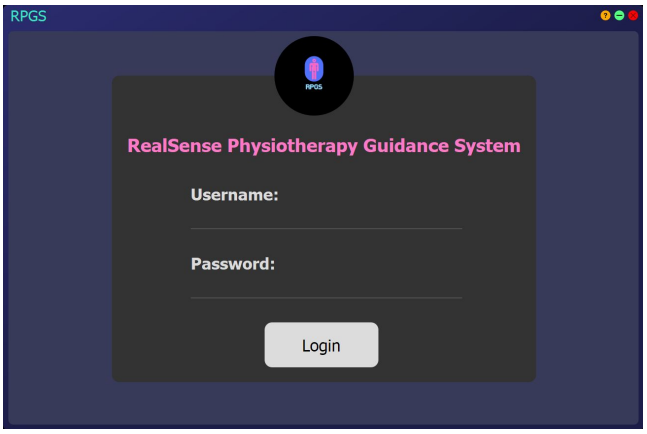

Figure 5: login page of the RPGS

4.3.1 Patient Mode

Upon logging into RPGS in patient mode, the patient is brought to the following screen shown in Figure 6. They are able to select an exercise from a list on the left and view the video showing the exercise, on the right. The video consists of a step-by step procedure on how to perform the exercise accurately to prevent errors in posture.

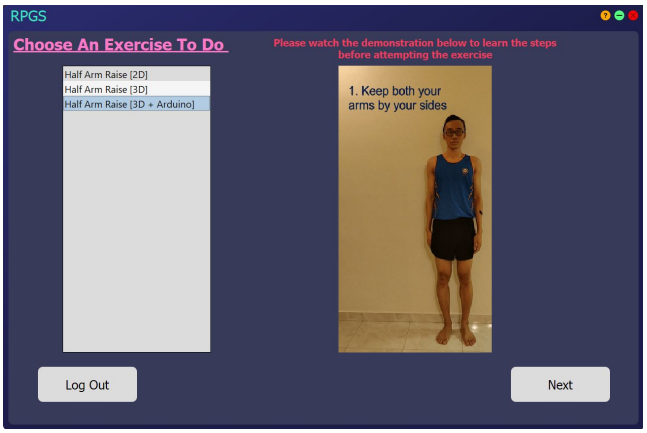

Figure 6: User selection in the patient mode of the RPGS

Upon clicking the "Next" button on the bottom right corner of the screen, the patient will be brought to the screen shown in Figure 7. This screen will show a real-time feed from the D435. A guidance skeleton is shown in red initially, while the patient's joints (called keypoints in Section 7) is highlighted in blue circles. A counter is also present on the top right hand corner showing the number of repetitions completed and the targeted number of repetitions of the current set. Here, the patient will try to attempt the chosen exercise as shown on the user viewer by matching their detected keypoints to that of the guidance skeleton. During the exercise, if the patient has not matched the posture of the guidance skeleton, or is doing something incorrectly, the guidance skeleton will remain red. Warnings will appear on the screen detailing how the patient should correct their posture. Once the patient completes one repetition of the exercise (consisting of a series of different postures), the completed number of repetitions will increase count by 1.

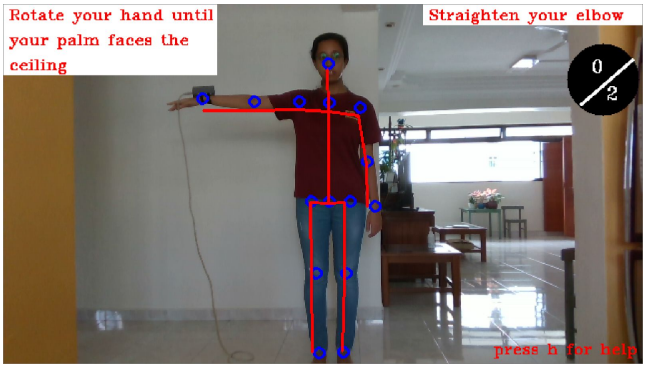

Figure 7: User viewer in the patient mode of the RPGS

When the user matches the guidance skeleton correctly, the guidance skeleton will flash green and change posture immediately. This is shown in Figure 8.

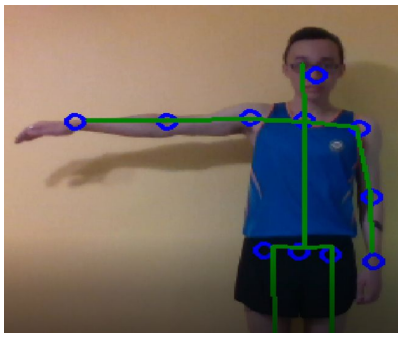

Figure 8: Guidance skeleton flashing green

4.3.2 Physiotherapist mode

Upon logging into RPGS in physiotherapist mode, the physiotherapist will be brought to the screen shown in Figure 9. The physiotherapist mode consists of two functions, where the physiotherapist is able to view the patient's medical and application history and also make adjustments to the level of difficulty of the exercise for the patient.

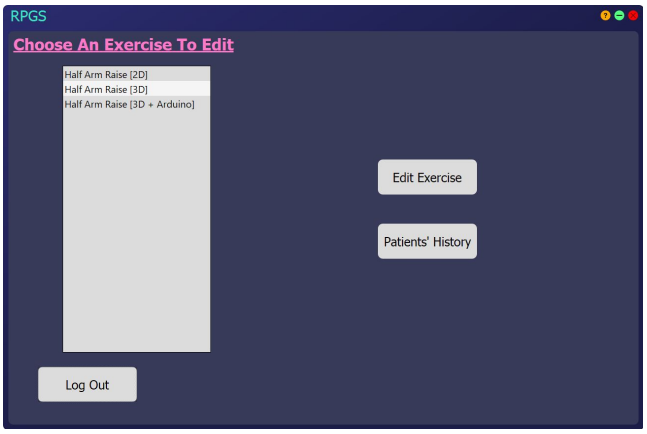

Figure 9: Main page in the physiotherapist mode of the RPGS

After clicking on "Edit exercise", the physiotherapist will be brought to the screen shown in Figure 10. There, the physiotherapist will be able to customise the exercise difficulty through adjusting the parameters that govern the correctness of each exercise.

One example of how exercises can be customised is to change the angles between relevant keypoints and the respective tolerances of the angles. The number of repetitions per set can also be changed at the bottom left corner.

Figure 10: User editor in the physiotherapist mode of the RPGS

5. Characterisation

Characterisation was carried out after the keypoints were successfully mapped onto a 3D Cartesian coordinate space by deprojection. Characterisation was done to ensure that the experiments were repeatable and the RPGS reliable.

5.1 Deprojection

The 2D coordinates (x,y) of the keypoints extracted from OpenPose will be in pixel coordinates, whereas the depth coordinate (z) will be in meters. This is further elaborated on in Section 7.2. Hence, there is a need to normalize all 3 coordinates to meters for an accurate representation of these points in 3D space. The (x, y) coordinates in pixels have their origin situated at the top left hand corner of the screen, with the positive x-direction towards the right of the screen and the positive y-direction towards the bottom of the screen. This is further complicated by the fact that pixels are wider in real space when they are further from the camera, because the field of view encompasses a larger area. This is shown in Figure 11.

With these problems taken into account, a deprojection algorithm was adopted to normalise the (x, y) pixel coordinates to meters, with the origin of the coordinate system being at the centre of the camera. This will provide us with a representation of the keypoint coordinates (x, y, z) in meters, allowing further characterisation to be done.

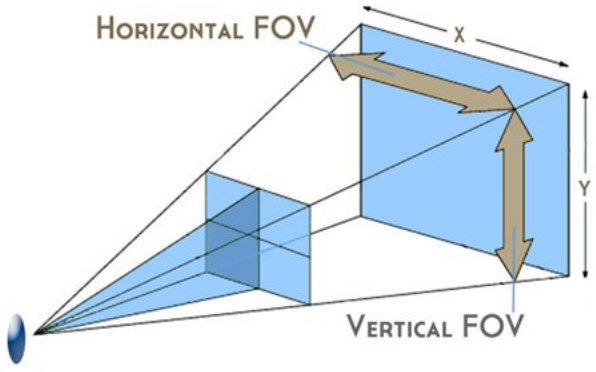

Figure 11: Difference between the area covered by the field of view for different distances from the camera[10]

5.2 Length of rod

The length of rod characterisation attempts to test the accuracy of measurements done using RPGS in 3D after being able to obtain the (x, y, z) coordinates of each keypoints in meters. The experiment involved the use of a rod that is be held by a person at fixed points. The distance between the centre of both wrists were measured using a measuring tape and found to be 0.82m (with an error of +-0.05cm). Holding the stick at the same 2 points throughout the entire experiment ensures that the relative distances between the wrist keypoints (4 and 7 according to Figure 20) is invariant (in real space) under the rotation and translation of the rod. Using OpenPose, the keypoints of

both wrists were detected and their depths found using the RealSense SDK.

The keypoints along with their depth values then goes through the deprojection function to obtain the 3D coordinates (x, y, z) in meters. Using equation (1), the distance between two keypoints can be calculated.

$$length\ AB = \sqrt{(A_x - B_x)^2 + (A_y - B_y)^2 + (A_z - B_z)^2} \quad \text{--- (1)}$$

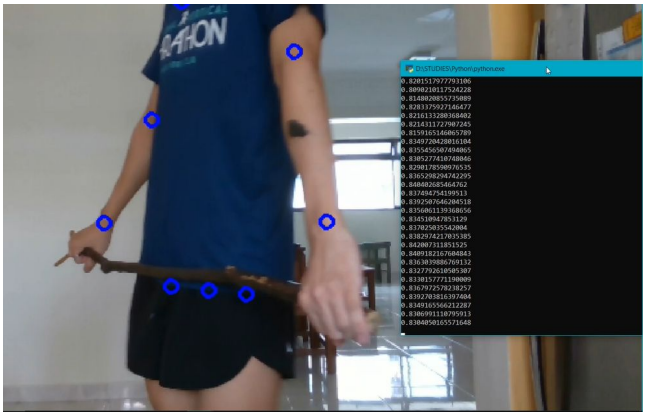

Figure 12: Length calculation using a modified version of PRGS. The distance between both wrists are shown on the terminal on the right

As seen in Figure 12, a user can be seen rotating his body along with the stick. Before using the camera system, the actual distance between both his wrists was at 0.82m apart. The user then stood facing the D435 camera and rotated his body to the right, then proceeded to move the rod upwards and downwards. In the terminal on the right, there is a column of values that shows what the camera system and distance algorithm calculated the distance between the wrists keypoints to be.

300 data points were sampled, with the average distance between the 2 keypoints being 0.832m and having a standard deviation of 0.015m, rounded to the nearest 3 decimal places. For purposes of large movements of limbs that physiotherapy is concerned with, this level of accuracy is acceptable and demonstrates that the deprojection algorithm adopted is functional.

5.3 Fluctuations of a fixed point

This experiment involved the characterisation of the stability of the keypoints identified by OpenPose and essentially tests the accuracy of OpenPose in detecting keypoints.

A person was made to sit on a chair with his back against the chair back. This is to ensure that the person remains stationary throughout the course of the experiment. The experimental setup is shown with the person and the camera in Figure 13.

Figure 13: Experimental setup for keypoint fluctuations experiment

The coordinates of the chest keypoint (x, y, z) in meters is recorded every frame and the experiment was terminated after at least 300 frames have elapsed. The standard deviations of the x, y, z points (in meters) are collated in Table 1, rounded off to the nearest 3 decimal places.

Distance	X std dev	Y std dev	Z std dev
1m	0.006	0.004	0.093
2m	0.006	0.004	0.093
3m	0.006	0.004	0.092

Table 1: standard deviations (in metres) of the x, y and z points at 1m, 2, and 3m respectively

Evidently, the standard deviations for all 3 coordinates do not vary much regardless of distance from the camera. However, the standard deviation of the depth measurement varies slightly. This will not be a problem for the exercise we want to demonstrate in Section 6, since adopting the correct posture will effectively line up all keypoints along a plane and they will not have significant differences in depth. However, further calibration and characterisation would be needed for more complicated postures that involve keypoints being at different depths.

6. Programmed exercise - single arm raise and rotation

For the purpose of this design project, we will be programming a simple exercise that makes use of the 3D imaging function from the D435 and data from the accelerometer. The exercise is shown in Figure 14. Note that for easier visualisation purposes (especially for the wrist area), the wearable module is not worn by the model patient. However, the wearable module is to be worn by the patient throughout the entire course of the exercise to detect for wrist rotation.

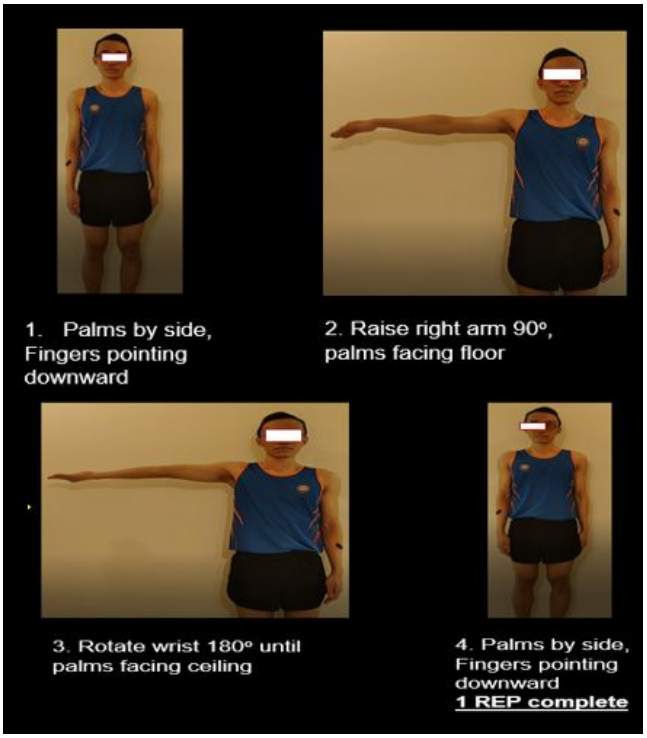

Figure 14: Pre-programmed exercise: Single Arm Raise and Rotation

The exercise involves 4 steps:

1. The patient first adopts the starting posture by having their hands by their sides, with their palms facing their thighs.
2. The patient raises their right arm such that it is 90° to their body, and 90° to the direction of view of the camera. The patient's palm should be facing the floor
3. The patient rotates their right arm along its own axis by 180° while maintaining the same posture as step 2. The patient's palm should be facing the ceiling.
4. The patient returns to the posture in step 1. This counts as 1 repetition of the exercise.

Throughout this exercise, various conditions have been implemented to detect for error in posture. Some examples include (but are not limited to): the palms being oriented the wrong way in steps 2 and 3, and the arm not being raised to 90° in step 2. A full list of these conditions, possible errors, and how they are implemented within the software will be detailed in Section 7.3.

7. Core Architecture

In this section, the underlying core architecture of the software component of the RPGS will be explained. The core architecture is made up of two toolkits: the RealSense SDK[11] (Software Development Kit) and OpenPose[12]. The RealSense SDK is a Github real-time multi-person keypoint detection library for body, face, hands, and foot estimation. It is only able to detect points on a 2D flat plane with no depth elements involved. The mapping of 2D points on a plane to 3D space using deprojection was already explained in Section 5.1

A pictorial representation of the software architecture is shown in Figure 15 below.

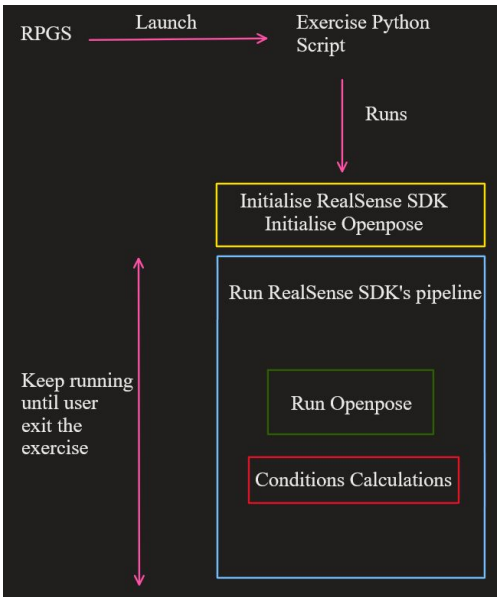

Figure 15: An overview of the technical workflow of RPGS

As seen in Figure 15, after the patient selects the appropriate exercise to do, the selected python script will first initialise both RealSense and OpenPose, then execute the RealSense SDK's pipeline, before running OpenPose within the SDK's pipeline. Conditions to decide whether the patient's posture is correct or not are also verified within the pipeline after extracting the necessary

information from the D435 camera, RealSense SDK and OpenPose. The pipeline including OpenPose then runs continuously until the user chooses to exit the exercise. The following sections will discuss the technical aspects of the RealSense SDK and OpenPose and how the correctness of poses are calculated.

7.1 RealSense SDK

The RealSense SDK works synergistically with the D435 camera to allow for image and depth manipulation. The native coding language of the SDK is in C++, but it also includes wrappers for other languages like Python, which was chosen to be the language to write RPGS. A pictorial representation of the SDK's pipeline can be seen in Figure 16 below.

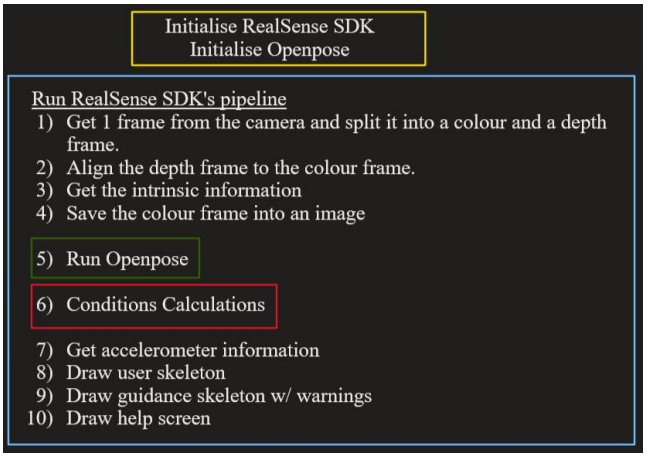

Figure 16: Breakdown of the operations happening within the RealSense SDK pipeline

As seen in Figure 16, the RealSense SDK will first undergo initialisation before opening up the pipeline. The initialisation will first enable both colour and depth sensors to collect a colour and depth stream respectively at a resolution of 640x360. The aforementioned resolution was chosen as it is the most common aspect ratio[13], and the experimental setup uses a laptop with the same aspect ratio. A resolution of 360p was chosen to improve the frames per second that is able to be shown on the screen. After setting up the sensors and resolution, the last step involves creating a function that will be used later during step 2 to align the depth frame to the colour frame. This is because the depth frame has a larger field of view than the colour frame.

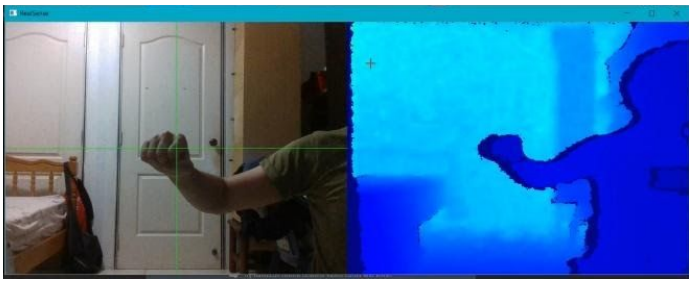

Figure 17: Field of view comparison between the colour (left) and depth (right) frame

As seen in Figure 17, the depth frame has a larger field of view when compared to the colour image. This is evident in the head of the person being imaged in the depth frame, but not in the colour frame. Hence, there is a need to make sure both frames contain the same field of view as accurate depth calculation would require the frames to be aligned

When the pipeline starts running, the SDK will first extract a frame from the D435 camera stream and split it into a colour and depth frame for further processing. After aligning both frames, the SDK then extracts the colour frame's intrinsic information, such as the dimensions of the camera image, principal point, focal points, distortion model and distortion coefficients[14]. This is because the intrinsic information of the colour frame would be necessary to create the final 3D coordinates of any pixel point. In step 4, the 2D colour image will be saved for OpenPose to process. After extracting the 2D coordinates of the patient's joint coordinates, also known as keypoints, calculations would be made (based on the customisation) from those keypoints to determine whether the patient's posture is correct or not. In Step 7, information from the wearable module would be sampled before drawing the user skeleton, guidance skeleton, any applicable warnings and a help function. Finally, as long as the user does not close or logout from RPGS, the pipeline will continuously loop from step one to ten.

7.2 OpenPose

OpenPose is an open source, real-time keypoint detection library, which is able to detect the human body, face, hand and feet. It is able to detect keypoints either in 2D, using a single camera or in 3D, using multiple cameras. For the user's convenience of not having to set up more than one camera and budget constraints, RPGS will first use OpenPose to obtain the 2D keypoints of the user's body, before subsequently lifting the keypoints to 3D by using the RealSense SDK. This is detailed in Section 7.3. Hence,

OpenPose was written in Python as well for language compatibility. An illustration of OpenPose's workflow can be seen in Figure 18 below.

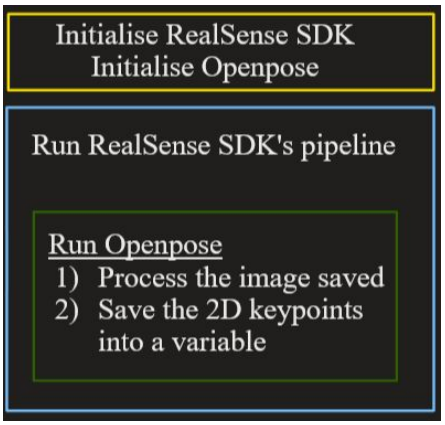

Figure 18: Breakdown of the operations happening when running OpenPose

In Figure 18, OpenPose will also first undergo an initialisation stage. It will first load the chosen pose model called "body_25", as it is faster and more accurate than the other two options called "COCO" and "MPI"[15]. An illustration of the body_25 model is shown in Figure 19 below.

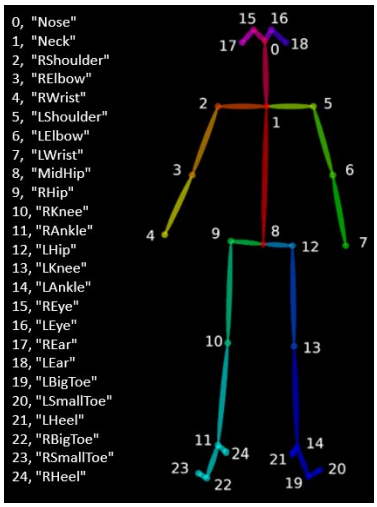

Figure 19: Labelled keypoint of body_25 model[16]. Image is mirrored

In Figure 19, the body_25 model involves identifying 25 different keypoints around the human body, and theoretically, we can program different exercises involving any of these 25 different keypoints. However, since the chosen exercise in Section 6 only involves the arms, RPGS uses a modified version of body_25 as a representation for the user and guidance skeleton as seen in Figure 20 below (the model used to identify the keypoints is still the full version of body_25). However, it is still possible to return the keypoints for the ankles and face if they are needed to be programmed for more exercises focusing on those keypoints.

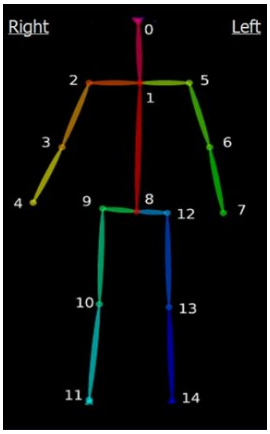

Figure 20: Modified version of body_25 used in RPGS

As seen in Figure 20, the modified version removes ten redundant keypoints from the head and from the feet. This makes it look simpler and will be easier to understand.

Another important parameter to initialise is the number of people OpenPose will take into account. Since RPGS is meant for one user at a time, OpenPose will only process the most prominent person it can detect. This parameter could incur better performance since computation is only done on one person, and any false positive detection in the background is eliminated. Afterwhich, OpenPose is set to asynchronous mode, as it will be processing an image and not the default camera stream which is occupied by the RealSense SDK. In the pipeline, OpenPose will save the computed keypoints into a variable in the code, which is more efficient than the previous workflow as seen in Figure 21 below.

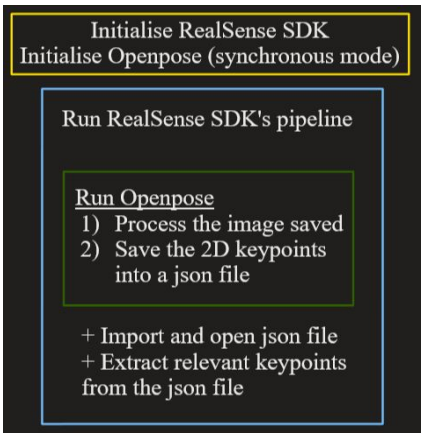

Figure 21: Previous version of the workflow of OpenPose

As seen in Figure 21, the previous version of RPGS ran OpenPose in synchronous mode and had to save the computed keypoints into a json file. This caused RPGS to run two additional functions to extract the relevant 2D keypoints from OpenPose. Hence, the current implementation of OpenPose is much more efficient than before.

7.3 Calculation of Conditions

Before deciding whether the user's pose is considered correct or not, calculations need to be based on the pre-programmed conditions and whether the relative orientations of the detected keypoints fit those conditions. For the purposes of our chosen exercise, the pre-programmed conditions will only involve the angles between various keypoints, and the spatial orientation of the wearable module.

7.3.1 Calculation of Angles

For the chosen exercise, the keypoints identified to be relevant are the:

1. Chest
2. Right shoulder
3. Right elbow
4. Right wrist
8. Centre of waist
9. Right pelvis
12. Left pelvis

Figure 22 shows a pictorial representation of the relevant keypoints and the angles. For example, angle 1-2-3 simply refers to the smaller angle (less than 180º) between the keypoints 1, 2, 3. Vector N is defined as the normal of the plane defined by the keypoints 1, 9, 12. It is chosen to be oriented out of the page, or towards the front of the patient, by taking the cross product of angles 1-9 and 1-12. Vector 1-3 refers to the vector in the originating from keypoint 1 and ending at keypoint 3. All relevant vectors have been normalized to unit vectors before calculating angles, and a standard tolerance of +- 10º is allocated to account for the differences in the skeletal structure of patients and possible systematic errors such as the points 1, 2, 3, 4 not lying exactly on a plane because keypoint 1 (chest) will be closer to the camera than keypoints 2 (shoulder), 3 (elbow), 4 (wrist) due to its protruding nature. However, the physiotherapist in physiotherapist mode can alter the ideal angle and respective tolerances as well (Figure 10).

Throughout the exercise, we pre-programmed five conditions, being:

1. Angle 1-2-3 = (180 +- 10)º
2. Angle 2-3-4 = (180 +- 10)º
3. Angle 8-1-4 = (90 +- 10)º
4. Angle between vector N and vector 1-3 = (90 +- 10)º
5. Angle between vector N and vector 1-4 = (90 +- 10)º

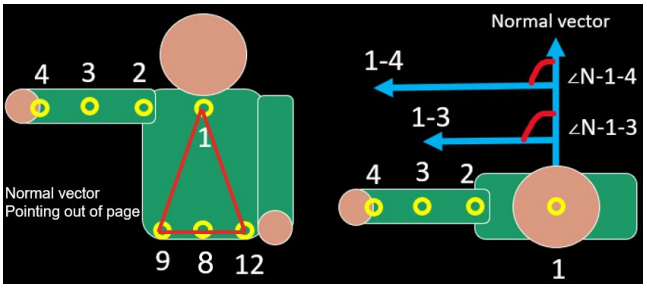

Figure 22: Pictorial representation of keypoints and angles

The calculation of these angles is achieved by first finding the depth of keypoints 1, 3, 4, 8, 9 and 12 in meters using the built-in function in the RealSense SDK. Next, all the keypoints pass through another built-in function called deprojection. According to Intel's documentation, deprojection takes a 2D coordinate on a stream's image, as well as its depth, and maps it to a 3D point location within the stream's associated 3D coordinate space[17]. This basically converts the 2D coordinates from OpenPose into 3D coordinates in the camera's coordinate space. Once the 3D coordinates are found, the vectors 1-3, 1-4, 1-9 and 1-12 can then be computed along with their lengths using

$$Vector\ AB = (B_x - A_x), (B_y - A_y), (B_z - A_z) \quad \text{--- (2)}$$

$$length\ AB = \sqrt{(A_x - B_x)^2 + (A_y - B_y)^2 + (A_z - B_z)^2} \quad \text{--- (3)}$$

Where $A_{x/y/z}$ is keypoint A in the x, y and depth axis, and $B_{x/y/z}$ is keypoint B in the x, y and depth axis. Finding both values allows for the computation of a normalised vector by dividing the vector with its length. Next, the normal vector is calculated using the cross product function provided by the numpy module between normalised vectors 1-9 and 1-12. The normal vector would also be normalised. Finally, the angles N-1-3 and N-1-4 can be found by using the dot product function provided in the numpy module, between the normal vector and the normalised vectors 1-3 and 1-4 respectively.

The angles 1-2-3, 2-3-4 and 8-1-4 are 2D angles also calculated using geometry methods. In this case, the vectors and lengths can be calculated using

$$Vector\ AB = (B_x - A_x), (B_y - A_y) \quad \text{--- (4)}$$

$$length\ AB = \sqrt{(A_x - B_x)^2 + (A_y - B_y)^2} \quad \text{--- (5)}$$

For example, the angle 1-2-3 can be found by dividing equation (3) and (4) to find the normalised vector 2-1 and 2-3. Then angle 1-2-3 can be found by using the dot product function between normalised vectors 2-1 and 2-3.

If any of the keypoints mentioned above is not detected, the angles that contain that keypoints would equate to zero.

7.3.2 Accelerometer conditions

Data readings from the accelerometer in the wearable module is also used as a condition in RPGS. A copy of the Arduino code can be found in Appendix III.

Float variables (accelX, accelY, accelZ) for the acceleration in x, y, z directions according to the coordinate system defined in Figure 4 are first initialised. A loop is then activated for the accelerometer registers to start recording data. The data is transferred to the Arduino UNO with the Wire.h library and normalized to the acceleration due to gravity ($9.81 ms^{-2}$). A serial.print command is activated for each of the float variables to be displayed on the serial monitor. For purposes of our chosen exercise, data from the wearable module is only used to determine the spatial orientation of itself. That is to say, most of the acceleration the wearable module experiences is due to gravity.

After uploading the Arduino code into the Arduino, the acceleration data can be imported directly into the RealSense pipeline using the pyserial module. The data is then filtered to identify the highest absolute value out of all 3 orthogonal directions, and the axis that corresponds to that value. This is because the wearable module will be most aligned to the axis that experiences the greatest magnitude of gravitational acceleration. With that, RPGS will be able to detect the spatial orientation of the wearable module and thus, be able to use it as another condition to check if the patient is rotating their arm to the correct orientation.

The accelerometer conditions for all 4 steps are:

1. Predominatly negative y-direction
2. Predominatly negative x-direction
3. Predominatly positive x-direction
4. Predominatly negative x-direction (same as 1)

7.3.3 Deciding the outcome

After checking against the conditions mentioned previously, the correctness of the posture will be shown through the colour of the guidance skeleton. When the guidance skeleton flashes green, it means the pose is correct, and the guidance skeleton will change posture to the next step of the exercise. However, if the guidance skeleton turns red, it means the pose is incorrect. An illustration of the sequence of checks can be seen in Figure 23 below.

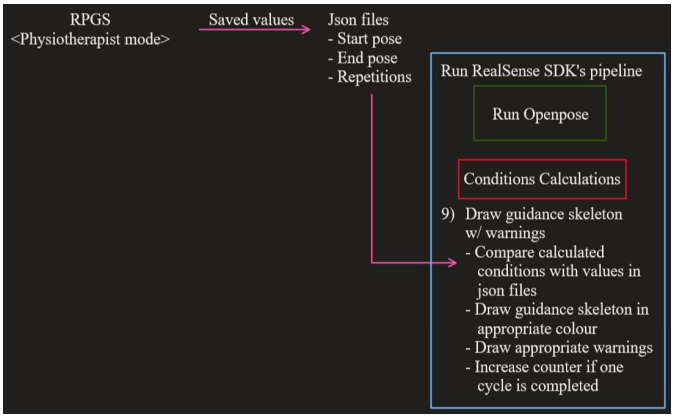

Figure 23: Workflow of how the outcome of the user's pose is decided using the guidance skeleton

As seen in Figure 23, after a physiotherapist sets the angles and the respective tolerances, the values will be saved into a json file. When the RealSense SDK's pipeline reaches step 9, it will compare the calculated values with their

corresponding saved values from those json files. If the calculated angles fall within the range of values saved and the rotational direction is correct for that pose, the guidance skeleton will be drawn in green to indicate that the pose is correct. On the other hand, if any of the conditions fails, the guidance skeleton will remain as red to indicate that the pose is incorrect. A warning system is also in place in the event that the outcome is incorrect to guide the user to the correct pose.

A list of warnings for the Single Arm Raise and Rotation exercise is shown below:

Distance from camera:

1. "Move forward/backward" when any of the keypoints are <2m or >3m

Arm movement (steps 2 and 3):

1. "Move your arm up" when angle 8-1-4 is less than the lower bound of the tolerance
2. "Move your arm down" when angle 8-1-4 is more than the upper bound of the tolerance
3. "Move your arm backwards" when either the angle between vector N and vector 1-3 or vector 1-4 is less than the lower bound of the tolerance
4. "Move your arm forwards" when either the angle between vector N and vector 1-3 or vector 1-4 is less than the upper bound of the tolerance
5. "Straighten your elbow" when angle 2-3-4 is out of the range of the tolerance

Rotation (all steps):

1. "Rotate your hand until your palm faces your leg" (step 1 & 4) when the wearable module is not oriented sufficiently in the negative y-direction
2. "Rotate your hand until your palm faces the floor" when the wearable module is not oriented sufficiently in the negative x-direction
3. "Rotate your hand until your palm faces the ceiling" when the wearable module is not oriented sufficiently in the positive x-direction

Lastly, by leveraging on the condition checking procedures, RPGS is also able to count the number of repetitions the patient has completed, against the total number of repetitions they are to do, as set by the physiotherapist.

8. Results and Discussion

The RPGS was able to demonstrate static posture recognition with the Single Arm Raise and Rotation in front of a live ZOOM audience during the ESP3902 Final Presentation. 3 different programs were run: purely 2D posture recognition, purely 3D posture recognition, and 3D posture recognition with the wearable module. This allowed the illustration of how the purely 2D and 3D posture recognition programs were unable to recognise some errors in posture due to their limited degrees of freedom. This included the inability to define a normal vector N in 2D, and the inability to detect rotation of joints in 3D. These problems were solved and demonstrated by using the 3D posture recognition with the wearable module.

However, feedback given included the response time for detecting the correct posture being relatively slow. This has to do with the Arduino within the wearable module transferring through PySerial at a faster rate than the frames per second, causing a misalignment between the frame and the specific values of the acceleration corresponding to that frame. Improvements can be made in the form of not sampling every single frame and introducing a common clocking system to sync both data from the Arduino and the frame rate.

All in all, the RPGS was able to execute a proof-of-concept exercise by using the functions of the 3D camera and accelerometer. This shows that more complicated exercises can be programmed in future, using the same fundamental concepts that have been explained in earlier sections.

9. Future Works

Although the RPGS is currently able to function for a simple exercise of a single arm raise followed by rotation of the arm, there are still further improvements that can be made in future with more time.

Firstly, the frame rate can be increased from 8 to 15 which is the frame rate of Assistive Rehab[18] which is another physiotherapy guidance platform that involves the patient following the actions of a life-sized robot. Assistive Rehab uses both OpenPose and RealSense libraries together with the physical robot to guide the patient on how to attempt the exercise correctly. This will make RPGS even more accurate in posture detection and have a faster response time.

Secondly, dynamic monitoring of velocity of the limbs can also be implemented on top of having static poses. This will give both the patient an accurate analysis of the speed of their limb movements, effectively adding an additional degree of freedom to quantify and analyse. Possible applications of this can be in the form of detecting how fast the exercise is being done, to determine if the speed of the exercise is too fast or too slow

Lastly, machine learning can possibly be implemented for complicated exercises poses that can be difficult for conditions to be set. This will involve multiple repetitions of the exercise by different patients to accumulate a diverse and large data set consisting of time series data of the joints. Time series analysis can then be done in real time to identify if the patient is performing the exercise accurately.

10. Conclusion

The RPGS is a home-based physiotherapy guidance system that is easy to set up and operate. It has proven to be accurate in determining the correctness of exercises, and able to provide specific real-time feedback of how to correct one's pose when one is performing it incorrectly. Through the proof-of-concept demonstration of the Single Arm Raise and Rotation exercise, the RPGS has been shown to be able to leverage on the functionalities of both D435 3D camera and accelerometer to detect the correctness of postures based on the spatial relationships between keypoints in 3D and spatial orientation of the wearable module.

11. References

[1]What is physiotherapy? (n.d.). Retrieved November 15, 2020, from https://www.csp.org.uk/careers-jobs/what-physiotherapy

[2]The Importance of Doing Physio Exercises. (2020, April 06). Retrieved November 15, 2020, from https://www.cairnhillphysiotherapy.co.nz/blog/the-importance-of-doing-physio-exercises/

[3] Tang, R., Yang, X., Bateman, S., Jorge, J., & Tang, A. (2015). Physio@Home. *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems - CHI '15*. doi:10.1145/2702123.2702401

[4] Graves, J. M., Iyer, K. R., Willis, M. M., Ebel, B. E., Rivara, F. P., & Vavilala, M. S. (2013). Emergency department-reported injuries associated with mechanical home exercise equipment in the USA. *Injury Prevention, 20*(4), 281-285. doi:10.1136/injuryprev-2013-040833

[5] Geskey, J. M. (2011). Home Exercise Equipment-Related Injuries. *AAP Grand Rounds, 26*(3), 28-28. doi:10.1542/gr.26-3-28

[6] Depth Camera D435. (2020, September 25). Retrieved November 15, 2020, from https://www.intelrealsense.com/depth-camera-d435/

[7] Arduino Uno Rev3. (n.d.). Retrieved November 15, 2020, from https://store.arduino.cc/usa/arduino-uno-rev3

[8] MPU-6050: TDK. (n.d.). Retrieved November 15, 2020, from https://invensense.tdk.com/products/motion-tracking/6-axis/mpu-6050/

[9] What is PyQt? (n.d.). Retrieved November 15, 2020, from https://riverbankcomputing.com/software/pyqt/intro

[10] Pourreza Ghoushchi, Vahid. (2015). Opto-Mechanical Design And Development of an Optodigital Confocal Microscope. 10.13140/RG.2.2.33261.38884.

[11] Developers area – Intel RealSense SDK 2.0 - Depth and Tracking cameras. (2020, October 22). Retrieved November 15, 2020, from https://www.intelrealsense.com/developers/

[12] CMU-Perceptual-Computing-Lab. (n.d.). CMU-Perceptual-Computing-Lab/openpose. Retrieved November 15, 2020, from https://github.com/CMU-Perceptual-Computing-Lab/openpose

[13] Rusen, C. A. (2020, October 25). What do the 720p, 1080p, 1440p, 2K, 4K resolutions mean? What are the aspect ratio & orientation? Retrieved November 15, 2020, from https://www.digitalcitizen.life/what-screen-resolution-or-aspect-ratio-what-do-720p-1080i-1080p-mean

[14] Pterneas, V. (2018, November 02). Intel RealSense D415/435: Coordinate Mapping in C#. Retrieved November 15, 2020, from https://lightbuzz.com/intel-realsense-coordinate-mapping/

[15] CMU-Perceptual-Computing-Lab. (2020, November 09). CMU-Perceptual-Computing-Lab/openpose. Retrieved November 15, 2020, from https://github.com/CMU-Perceptual-Computing-Lab/openpose/blob/master/doc/faq.md

[16] CMU-Perceptual-Computing-Lab. (2019, November 10). CMU-Perceptual-Computing-Lab/openpose. Retrieved November 15, 2020, from https://github.com/CMU-Perceptual-Computing-Lab/openpose/blob/master/doc/output.md

[17] Projection in Intel RealSense SDK 2.0. (2019). Retrieved November 15, 2020, from https://dev.intelrealsense.com/docs/projection-in-intel-realsense-sdk-20

[18] Vasco, V. (n.d.). Assistive-rehab. Retrieved November 16, 2020, from https://robotology.github.io/assistive-rehab/doc/mkdocs/site/

Appendix I

(GUI code)

(main UI code)

```python
################################################################################
##
## BY: WANDERSON M.PIMENTA
## PROJECT MADE WITH: Qt Designer and PySide2
## V: 1.0.0
##
################################################################################

import sys
import platform
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtCore import (QCoreApplication, QPropertyAnimation, QDate, QDateTime, QMetaObject, QObject, QPoint, QRect, QSize, QTime, QUrl, Qt, QEvent)
from PyQt5.QtGui import (QBrush, QColor, QConicalGradient, QCursor, QFont, QFontDatabase, QIcon, QKeySequence, QLinearGradient, QPalette, QPainter, QPixmap, QRadialGradient)
from PyQt5.QtWidgets import *

## ==> SPLASH SCREEN
from ui_splash_screen import Ui_SplashScreen

## ==> GLOBALS
counter = 0




# SPLASH SCREEN
class SplashScreen(QMainWindow):
    def __init__(self):
        QMainWindow.__init__(self)
        self.ui = Ui_SplashScreen()
        self.ui.setupUi(self)

        ## UI ==> INTERFACE CODES
        ########################################################################

        ## REMOVE TITLE BAR
        self.setWindowFlag(QtCore.Qt.FramelessWindowHint)
        self.setAttribute(QtCore.Qt.WA_TranslucentBackground)


        ## DROP SHADOW EFFECT
        self.shadow = QGraphicsDropShadowEffect(self)
        self.shadow.setBlurRadius(20)
        self.shadow.setXOffset(0)
```

```
        self.shadow.setYOffset(0)
        self.shadow.setColor(QColor(0, 0, 0, 60))
        self.ui.dropShadowFrame.setGraphicsEffect(self.shadow)

        ## QTIMER ==> START
        self.timer = QtCore.QTimer()
        self.timer.timeout.connect(self.progress)
        # TIMER IN MILLISECONDS
        self.timer.start(35)

        # CHANGE DESCRIPTION

        # Initial Text
        self.ui.label_description.setText("<strong>WELCOME</strong> TO RPGS!")

        # Change Texts
        QtCore.QTimer.singleShot(3500, lambda: self.ui.label_description.setText("<strong>LOADING</strong> DATABASE"))
        QtCore.QTimer.singleShot(5000, lambda: self.ui.label_description.setText("<strong>LOADING</strong> USER INTERFACE"))
        QtCore.QTimer.singleShot(6500, lambda: self.ui.label_description.setText("<strong>LOADING</strong> Scripts"))


        ## SHOW ==> MAIN WINDOW
        ########################################################################
        self.show()
        ## ==> END ##

    ## ==> APP FUNCTIONS
    ########################################################################
    def progress(self):

        global counter

        # SET VALUE TO PROGRESS BAR
        self.ui.progressBar.setValue(counter)

        # CLOSE SPLASH SCREE AND OPEN APP
        if counter > 100:
            # STOP TIMER
            self.timer.stop()

            # SHOW MAIN WINDOW
            import RPGS

            # CLOSE SPLASH SCREEN
            self.close()

        # INCREASE COUNTER
        counter += 0.65



if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = SplashScreen()
    sys.exit(app.exec_())

```

(loading bar code)

```python
# -*- coding: utf-8 -*-

################################################################################
## Form generated from reading UI file 'splash_screenXBSmkq.ui'
##
## Created by: Qt User Interface Compiler version 5.14.1
##
## WARNING! All changes made in this file will be lost when recompiling UI file!
################################################################################

from PyQt5.QtCore import (QCoreApplication, QMetaObject, QObject, QPoint,
    QRect, QSize, QUrl, Qt)
from PyQt5.QtGui import (QBrush, QColor, QConicalGradient, QCursor, QFont,
    QFontDatabase, QIcon, QLinearGradient, QPalette, QPainter, QPixmap,
    QRadialGradient)
from PyQt5.QtWidgets import *


class Ui_SplashScreen(object):
    def setupUi(self, SplashScreen):
        if SplashScreen.objectName():
            SplashScreen.setObjectName(u"SplashScreen")
        SplashScreen.resize(680, 400)
        self.centralwidget = QWidget(SplashScreen)
        self.centralwidget.setObjectName(u"centralwidget")
        self.verticalLayout = QVBoxLayout(self.centralwidget)
        self.verticalLayout.setSpacing(0)
        self.verticalLayout.setObjectName(u"verticalLayout")
        self.verticalLayout.setContentsMargins(10, 10, 10, 10)
        self.dropShadowFrame = QFrame(self.centralwidget)
        self.dropShadowFrame.setObjectName(u"dropShadowFrame")
        self.dropShadowFrame.setStyleSheet(u"QFrame {    \n"
"    background-color: rgb(56, 58, 89);  \n"
"    color: rgb(220, 220, 220);\n"
"    border-radius: 10px;\n"
"}")
        self.dropShadowFrame.setFrameShape(QFrame.StyledPanel)
        self.dropShadowFrame.setFrameShadow(QFrame.Raised)
        self.label_title = QLabel(self.dropShadowFrame)
        self.label_title.setObjectName(u"label_title")
        self.label_title.setGeometry(QRect(0, 90, 661, 61))
        font = QFont()
        font.setFamily(u"Segoe UI")
```

```
        font.setPointSize(20)
        self.label_title.setFont(font)
        self.label_title.setStyleSheet(u"color: rgb(254, 121, 199);")
        self.label_title.setAlignment(Qt.AlignCenter)
        self.label_description = QLabel(self.dropShadowFrame)
        self.label_description.setObjectName(u"label_description")
        self.label_description.setGeometry(QRect(0, 150, 661, 31))
        font1 = QFont()
        font1.setFamily(u"Segoe UI")
        font1.setPointSize(14)
        self.label_description.setFont(font1)
        self.label_description.setStyleSheet(u"color: rgb(98, 114, 164);")
        self.label_description.setAlignment(Qt.AlignCenter)
        self.progressBar = QProgressBar(self.dropShadowFrame)
        self.progressBar.setObjectName(u"progressBar")
        self.progressBar.setGeometry(QRect(50, 280, 561, 23))
        self.progressBar.setStyleSheet(u"QProgressBar {\n"
"    \n"
"    background-color: rgb(98, 114, 164);\n"
"    color: rgb(200, 200, 200);\n"
"    border-style: none;\n"
"    border-radius: 10px;\n"
"    text-align: center;\n"
"}\n"
"QProgressBar::chunk{\n"
"    border-radius: 10px;\n"
"    background-color: qlineargradient(spread:pad, x1:0, y1:0.511364, x2:1, y2:0.523, stop:0 rgba(254, 121, 199, 255), stop:1 rgba(170, 85, 255, 255));\n"
"}")
        self.progressBar.setValue(24)
        self.label_loading = QLabel(self.dropShadowFrame)
        self.label_loading.setObjectName(u"label_loading")
        self.label_loading.setGeometry(QRect(0, 320, 661, 21))
        font2 = QFont()
        font2.setFamily(u"Segoe UI")
        font2.setPointSize(12)
        self.label_loading.setFont(font2)
        self.label_loading.setStyleSheet(u"color: rgb(98, 114, 164);")
        self.label_loading.setAlignment(Qt.AlignCenter)
        self.label_credits = QLabel(self.dropShadowFrame)
        self.label_credits.setObjectName(u"label_credits")
        self.label_credits.setGeometry(QRect(20, 350, 621, 21))
        font3 = QFont()
        font3.setFamily(u"Segoe UI")
        font3.setPointSize(10)
        self.label_credits.setFont(font3)
        self.label_credits.setStyleSheet(u"color: rgb(98, 114, 164);")
        self.label_credits.setAlignment(Qt.AlignRight|Qt.AlignTrailing|Qt.AlignVCenter)

        self.verticalLayout.addWidget(self.dropShadowFrame)

        SplashScreen.setCentralWidget(self.centralwidget)

        self.retranslateUi(SplashScreen)

        QMetaObject.connectSlotsByName(SplashScreen)
    # setupUi

    def retranslateUi(self, SplashScreen):
        SplashScreen.setWindowTitle(QCoreApplication.translate("SplashScreen", u"MainWindow", None))
        self.label_title.setText(QCoreApplication.translate("SplashScreen", u" RealSense Physiotherapy Guidance System", None))
        self.label_description.setText(QCoreApplication.translate("SplashScreen", u"<strong>APP</strong> DESCRIPTION", None))
        self.label_loading.setText(QCoreApplication.translate("SplashScreen", u"loading...", None))
    # retranslateUi

```

(RPGS UI code)

```python
import sys
import os
import json

from PyQt5 import uic, QtWidgets, QtCore
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *
from PyQt5.QtGui import *
import webbrowser # to open pdf files
from PyQt5.QtGui import QMovie, QPixmap # for the gif


qtCreatorFile = "Main_GUI.ui"


class Ui(QtWidgets.QMainWindow):

    def __init__(self):
        super(Ui, self).__init__()
        uic.loadUi(qtCreatorFile, self)

        # REMOVE TITLE BAR
        self.setWindowFlag(QtCore.Qt.FramelessWindowHint)
        self.setAttribute(QtCore.Qt.WA_TranslucentBackground)

        def moveWindow(event):
            # IF LEFT CLICK MOVE WINDOW
            if event.buttons() == Qt.LeftButton:
                self.move(self.pos() + event.globalPos() - self.dragPos)
                self.dragPos = event.globalPos()
                event.accept()

        # SET TITLE BAR
        self.title_bar.mouseMoveEvent = moveWindow

        # CLOSE
        self.btn_close.clicked.connect(lambda: self.close())

        # MINIMIZE
        self.btn_minimize.clicked.connect(lambda: self.showMinimized())

        # change the program's main window name
        self.setWindowTitle('RPGS')
```

```
44
45          # Just for [3] label on top left
46          self.label_text = ""
47
48          # Prevent resizing
49          self.setFixedSize(self.size())
50
51
52          # Login page
53          self.stackedWidget.setCurrentIndex(0)
54          self.Username_field.returnPressed.connect(self.Login_btn.click) # enter key act as clicking the login button
55          self.Password_field.returnPressed.connect(self.Login_btn.click)
56          self.Login_btn.clicked.connect(self.loginfunction) # main login function
57
58          # Choose exercise page (PATIENT) [1]
59          self.Exercise_ChooseExercise.addItem("Half Arm Raise [2D]") # add the items to be selectable
60          self.Exercise_ChooseExercise.addItem("Half Arm Raise [3D]")
61          self.Exercise_ChooseExercise.addItem("Half Arm Raise [3D + Arduino]")
62          self.Exercise_ChooseExercise.itemClicked.connect(self.Exercise_Display) # run video when exercise selected
63          self.Exercise_NextBtn.clicked.connect(self.run_exercise) # run the py script depending on what user chose
64          self.Logout_btn.clicked.connect(self.logout)
65
66          # Choose exercise page (DEVELOPER) [2]
67          self.Exercise_ChooseExercise_2.addItem("Half Arm Raise [2D]") # add the items to be selectable
68          self.Exercise_ChooseExercise_2.addItem("Half Arm Raise [3D]")
69          self.Exercise_ChooseExercise_2.addItem("Half Arm Raise [3D + Arduino]")
70          self.Logout_btn_2.clicked.connect(self.logout)
71          self.edit_exercise_btn.clicked.connect(self.click_to_label_variable) # go to next page while changing label text
72
73          # Dev chosen exercise page [3]
74          self.stackedWidget_edit.setCurrentIndex(0)
75
76          with open(r"D:\STUDIES\YEAR 3 - SEM 1\ESP3902\openpose-master\build\examples\tutorial_api_python\Dev_Json\start.json") as start_json:
77              start = json.load(start_json)
78              for start_123 in start['start_123']:
79                  self.start_123_angle.setText(str(start_123["angle"]))
80                  self.start_123_upper.setText(str(start_123["upper"]))
81                  self.start_123_lower.setText(str(start_123["lower"]))
82              for start_234 in start['start_234']:
83                  self.start_234_angle.setText(str(start_234["angle"]))
84                  self.start_234_upper.setText(str(start_234["upper"]))
85                  self.start_234_lower.setText(str(start_234["lower"]))
86
```

```
        with open(r"D:\STUDIES\YEAR 3 - SEM 1\ESP3902\openpose-master\build\examples\tutorial_api_python\Dev_Json\end.json") as end_json:
            end = json.load(end_json)
            for end_234 in end['end_234']:
                self.end_234_angle.setText(str(end_234["angle"]))
                self.end_234_upper.setText(str(end_234["upper"]))
                self.end_234_lower.setText(str(end_234["lower"]))
            for end_814 in end['end_814']:
                self.end_814_angle.setText(str(end_814["angle"]))
                self.end_814_upper.setText(str(end_814["upper"]))
                self.end_814_lower.setText(str(end_814["lower"]))
            for end_13 in end['end_13']:
                self.end_13_angle.setText(str(end_13["angle"]))
                self.end_13_upper.setText(str(end_13["upper"]))
                self.end_13_lower.setText(str(end_13["lower"]))
            for end_14 in end['end_14']:
                self.end_14_angle.setText(str(end_14["angle"]))
                self.end_14_upper.setText(str(end_14["upper"]))
                self.end_14_lower.setText(str(end_14["lower"]))

        with open(r"D:\STUDIES\YEAR 3 - SEM 1\ESP3902\openpose-master\build\examples\tutorial_api_python\Dev_Json\repetition.json") as rep_json:
            rep = json.load(rep_json)
            self.Number_of_reps.setValue(int(rep['repetition'][0]['repetition_number']))


        self.start_pose_edit.clicked.connect(lambda: self.stackedWidget_edit.setCurrentIndex(0))
        self.end_pose_edit.clicked.connect(lambda: self.stackedWidget_edit.setCurrentIndex(1))
        self.start_save_btn.clicked.connect(self.save_start)
        self.end_save_btn.clicked.connect(self.save_end)
        self.dev_edit_back_btn.clicked.connect(lambda: self.stackedWidget.setCurrentIndex(2))


        # For the menu bar
        self.btn_help.clicked.connect(self.Help)


        # DONT TOUCH
        self.show()

#-----------------------------------------------------------------------------------------------------------------------------------------------------------

    # FUNCTIONS FROM HERE ONWARDS
    def loginfunction(self):
        username = self.Username_field.text()
```

```
        password = self.Password_field.text()
        self.Username_field.clear()
        self.Password_field.clear()
        if(username == "patient" and password == "patient"):
            self.stackedWidget.setCurrentIndex(1)
        elif(username== "dev" and password == "dev"):
            self.stackedWidget.setCurrentIndex(2)
        elif(username == "" and password == ""):
            return
        else:
            error_screen = QMessageBox()
            error_screen.setWindowTitle("Wrong Login")
            error_screen.setIcon(QMessageBox.Warning)
            error_screen.setText("Wrong Username or Password")
            x = error_screen.exec_()


    def Exercise_Display(self):
        if(self.Exercise_ChooseExercise.currentRow() == 0 or 1 or 2):
            movie = QMovie('instructional-video-raw.gif')
            self.Exercise_DisplayExercise.setMovie(movie)
            movie.start()


    def run_exercise(self):
        listItems = self.Exercise_ChooseExercise.selectedItems()
        if not listItems:
            return
        else:
            exercise_chosen = self.Exercise_ChooseExercise.currentItem().text()
            if(exercise_chosen == "Half Arm Raise [2D]"):
                exec(open("realsense_OP_Half_arm_raise_2D.py").read())
            elif(exercise_chosen == "Half Arm Raise [3D]"):
                exec(open("realsense_OP_Half_arm_raise_3D_V2.py").read())
            elif(exercise_chosen == "Half Arm Raise [3D + Arduino]"):
                exec(open("realsense_OP_Half_arm_raise_3D_arduino_V4(for GUI).py").read())


    def logout(self):
        logout_screen = QMessageBox()
        user_click = logout_screen.question(self,'Log Out', "Are You Sure You Want To Log Out?", logout_screen.Yes | logout_screen.No)
        if(user_click == logout_screen.Yes):
            self.stackedWidget.setCurrentIndex(0)
```

```
173
174
175     def Help(self):
176         # CHANGE THE PATH ACORDING TO THE CORRECT PDF
177         webbrowser.open_new('instruction manual.pdf')
178
179
180     def click_to_label_variable(self):
181         listItems = self.Exercise_ChooseExercise_2.selectedItems()
182         if not listItems:
183             return
184         else:
185             self.label_text = self.Exercise_ChooseExercise_2.currentItem().text()
186             self.stackedWidget.setCurrentIndex(3)
187             self.Dev_exercise_label.setText(self.label_text)
188
189
190     def save_start(self):
191         save = QMessageBox()
192         user_click = save.question(self,'Starting Pose Save?', "Are You Sure You Want To Save The Conditions?", save.Yes | save.No)
193         if(user_click == save.Yes):
194             data={}
195             data['start_123'] = []
196             data['start_123'].append({
197                 'angle': int(self.start_123_angle.text()),
198                 'upper': int(self.start_123_upper.text()),
199                 'lower': int(self.start_123_lower.text())
200             })
201             data['start_234'] = []
202             data['start_234'].append({
203                 'angle': int(self.start_234_angle.text()),
204                 'upper': int(self.start_234_upper.text()),
205                 'lower': int(self.start_234_lower.text())
206             })
207
208             rep_data={}
209             rep_data['repetition']=[]
210             rep_data['repetition'].append({
211                 'repetition_number': int(self.Number_of_reps.value()),
212             })
213
214             with open(r"D:\STUDIES\YEAR 3 - SEM 1\ESP3902\openpose-master\build\examples\tutorial_api_python\Dev_Json\start.json", 'w') as outfile:
215                 json.dump(data, outfile)
```

```
216
217             with open(r"D:\STUDIES\YEAR 3 - SEM 1\ESP3902\openpose-master\build\examples\tutorial_api_python\Dev_Json\repetition.json", 'w') as outfile:
218                 json.dump(rep_data, outfile)
219
220             saved = QMessageBox()
221             saved.setWindowTitle("Success")
222             saved.setText("Starting Pose Saved")
223             x = saved.exec_()
224
225
226     def save_end(self):
227         save = QMessageBox()
228         user_click = save.question(self,'End Pose Save?', "Are You Sure You Want To Save The Conditions?", save.Yes | save.No)
229         if(user_click == save.Yes):
230             data={}
231             data['end_234'] = []
232             data['end_234'].append({
233                 'angle': int(self.end_234_angle.text()),
234                 'upper': int(self.end_234_upper.text()),
235                 'lower': int(self.end_234_lower.text())
236             })
237             data['end_814'] = []
238             data['end_814'].append({
239                 'angle': int(self.end_814_angle.text()),
240                 'upper': int(self.end_814_upper.text()),
241                 'lower': int(self.end_814_lower.text())
242             })
243             data['end_13'] = []
244             data['end_13'].append({
245                 'angle': int(self.end_13_angle.text()),
246                 'upper': int(self.end_13_upper.text()),
247                 'lower': int(self.end_13_lower.text())
248             })
249             data['end_14'] = []
250             data['end_14'].append({
251                 'angle': int(self.end_14_angle.text()),
252                 'upper': int(self.end_14_upper.text()),
253                 'lower': int(self.end_14_lower.text())
254             })
255
256             rep_data={}
257             rep_data['repetition']=[]
258             rep_data['repetition'].append({
```

```
258             rep_data['repetition'].append({
259                 'repetition_number': int(self.Number_of_reps.value()),
260             })
261
262             with open(r"D:\STUDIES\YEAR 3 - SEM 1\ESP3902\openpose-master\build\examples\tutorial_api_python\Dev_Json\end.json", 'w') as outfile:
263                 json.dump(data, outfile)
264
265             with open(r"D:\STUDIES\YEAR 3 - SEM 1\ESP3902\openpose-master\build\examples\tutorial_api_python\Dev_Json\repetition.json", 'w') as outfile:
266                 json.dump(rep_data, outfile)
267
268             saved = QMessageBox()
269             saved.setWindowTitle("Success")
270             saved.setText("End Pose Saved")
271             x = saved.exec_()
272
273
274     def mousePressEvent(self, event):
275         self.dragPos = event.globalPos()
276
277
278
279
280
281
282 app = QtWidgets.QApplication(sys.argv)
283 window = Ui()
284 # app.exec_()
285 app.exec_()
286
```

Appendix II
(Source code)

```python
# Credit: https://github.com/IntelRealSense/librealsense/issues/1904#issuecomment-398434434
import pyrealsense2 as rs
import numpy as np
import cv2
import os
import sys
from sys import platform
import argparse
import json
import math
import csv
import serial
from serial import *

# Realsense initialisation ------------------------------------------------------------------------------------------------------------------------------------
# Configure depth and color streams
pipeline = rs.pipeline()

#Create a config and configure the pipeline to stream
config = rs.config()
config.enable_stream(rs.stream.depth, 640, 360, rs.format.z16, 30)
config.enable_stream(rs.stream.color, 640, 360, rs.format.bgr8, 30)

# Start streaming
pipe_profile = pipeline.start(config)

# Getting the depth sensor's depth scale (see rs-align example for explanation)
depth_sensor = pipe_profile.get_device().first_depth_sensor()
depth_scale = depth_sensor.get_depth_scale()

# Create an align object
# rs.align allows us to perform alignment of depth frames to others frames
# The "align_to" is the stream type to which we plan to align depth frames.
align_to = rs.stream.color
align = rs.align(align_to)
# RealSense end initialisation --------------------------------------------------------------------------------------------------------------------------------



# openpose initialisation -------------------------------------------------------------------------------------------------------------------------------------
# Import Openpose
dir_path = os.path.dirname(os.path.realpath(__file__))

```

```
# Change these variables to point to the correct folder (Release/x64 etc.)
sys.path.append(dir_path + '/../../python/openpose/Release');
os.environ['PATH']  = os.environ['PATH'] + ';' + dir_path + '/../../x64/Release;' +  dir_path + '/../../bin;'
import pyopenpose as op


# Flags
parser = argparse.ArgumentParser()
# parser.add_argument("--no_display", default=True, help="Enable to disable the visual display.")
args = parser.parse_known_args()

# !!To use openpose in realsense, i need to be able to read image from a folder, then output the json to another folder,
# then realsense use that folder to import the json files
#
# Custom Params (refer to include/openpose/flags.hpp for more parameters)
params = dict()
params["model_folder"] = "../../../models/" # DO NOT CHANGE
params["display"] = 0
params["render_pose"] = 0
params["number_people_max"] = 1
params["write_json"] = "Test_input_images_json" # I think this is still needed. Can try removing it.


# Add others in path?
for i in range(0, len(args[1])):
    curr_item = args[1][i]
    if i != len(args[1])-1: next_item = args[1][i+1]
    else: next_item = "1"
    if "--" in curr_item and "--" in next_item:
        key = curr_item.replace('-','')
        if key not in params:  params[key] = "1"
    elif "--" in curr_item and "--" not in next_item:
        key = curr_item.replace('-','')
        if key not in params: params[key] = next_item

# Start openpose
opWrapper = op.WrapperPython(0) # DONT TOUCH THE VALUE
opWrapper.configure(params)
opWrapper.start()
# openpose end initialisation ----------------------------------------------------------------------------------------------------------------------------------------------------------
```

```
# Guidance skeleton initialisation ----------
# END POSE
with open("D:/STUDIES/YEAR 3 - SEM 1/ESP3902/openpose-master/build/examples/tutorial_api_python/Pose Database/half_T_pose_end.json") as g:
    guidance_skele_json = json.loads(g.read())

guidance_skele_part_end = guidance_skele_json["part_candidates"]

#STARTING POSE
with open("D:/STUDIES/YEAR 3 - SEM 1/ESP3902/openpose-master/build/examples/tutorial_api_python/Pose Database/half_T_pose_start.json") as g:
    guidance_skele_json = json.loads(g.read())

guidance_skele_part_start = guidance_skele_json["part_candidates"]
# Guidance skeleton end initialisation ----------



# MATH initialisation for GUIDANCE angles ----------

with open(r"D:\STUDIES\YEAR 3 - SEM 1\ESP3902\openpose-master\build\examples\tutorial_api_python\Dev_Json\start.json") as start_json:
    start = json.load(start_json)
    for start_123 in start['start_123']:
        guidance_start_angle123 = start_123["angle"]
        start_angle123_upperAllowance = start_123["upper"]
        start_angle123_lowerAllowance = start_123["lower"]
    for start_234 in start['start_234']:
        guidance_start_angle234 = start_234["angle"]
        start_angle234_allowance = start_234["lower"]

with open(r"D:\STUDIES\YEAR 3 - SEM 1\ESP3902\openpose-master\build\examples\tutorial_api_python\Dev_Json\end.json") as end_json:
    end = json.load(end_json)
    for end_234 in end['end_234']:
        guidance_end_angle234 = end_234["angle"]
        end_angle234_allowance = end_234["lower"]
    for end_814 in end['end_814']:
        guidance_end_angle814 = end_814["angle"]
        end_angle814_upperAllowance = end_814["upper"]
        end_angle814_lowerAllowance = end_814["lower"]
    for end_14 in end['end_14']:
        guidance_planar_angle_chest_14 = end_14["angle"]
        planar_angle_chest_14_upperAllowance = end_14["upper"]
        planar_angle_chest_14_lowerAllowance = end_14["lower"]
    for end_13 in end['end_13']:
        guidance_planar_angle_chest_13 = end_13["angle"]
```

```python
        planar_angle_chest_13_upperAllowance = end_13["upper"]
        planar_angle_chest_13_lowerAllowance = end_13["lower"]

with open(r"D:\STUDIES\YEAR 3 - SEM 1\ESP3902\openpose-master\build\examples\tutorial_api_python\Dev_Json\repetition.json") as rep_json:
    rep = json.load(rep_json)
    rep_value = rep['repetition'][0]['repetition_number']
# MATH end initialisation for GUIDANCE angles -----------------------------------------------------------------------------------------------------------------------



# Arduino read from serial initialisation -----------------------------------------------------------------------------------------------------------------------
ser = serial.Serial('COM7', 9600)  # RMB TO CHANGE THE COM PORT BASE ON YOUR SETUP!
ser.xonxoff=1
ser.flushInput()
# Arduino read from serial initialisation end -------------------------------------------------------------------------------------------------------------------



# Drawing config ----------------------------------    *Realsense use BGR instead of RGB like a normal person. weirdo.
line_thickness = 2                                #|
colour_red = (0, 0, 255)                          #|
colour_green = (0, 128, 0)                        #|
colour_blue = (255,0,0)                           #|
font = cv2.FONT_HERSHEY_TRIPLEX                   #|
#--------------------------------------------------


# global variables
curr_frame = 0
curr_pose = 0 # 0 is start, 1 is end
csv_frames = 0 # for CSV only
curr_rot = 0 # 0 is starting rotation, 1 is ending rotation part one, 2 is ending rotation part two
counter = 0 # for how many cycles completed
help_switch = 0 # the help screen is not toggled
bol_start = True # for the counter. Only after started then can increment.


# Streaming loop
try:
    while True:
        # Wait for a coherent pair of frames: depth and color
        frames = pipeline.wait_for_frames()   # frames.get_depth_frame() is a 640x360 depth image
        depth_frame = frames.get_depth_frame()
```

```
        color_frame = frames.get_color_frame()

        # Align the depth frame to color frame
        aligned_frames = align.process(frames)

        # Get aligned frames
        aligned_depth_frame = aligned_frames.get_depth_frame() # aligned_depth_frame is a 640x360 depth image
        color_frame = aligned_frames.get_color_frame()

        # Validate that both frames are valid
        if not aligned_depth_frame or not color_frame:
            continue

        # Intrinsics & Extrinsics
        depth_intrin = aligned_depth_frame.profile.as_video_stream_profile().intrinsics
        color_intrin = color_frame.profile.as_video_stream_profile().intrinsics
        depth_to_color_extrin = aligned_depth_frame.profile.get_extrinsics_to(color_frame.profile)

        # Convert images to numpy arrays
        depth_image = np.asanyarray(aligned_depth_frame.get_data())
        color_image = np.asanyarray(color_frame.get_data())

        # save the colour image into a folder in path
        path = "D:/STUDIES/YEAR 3 - SEM 1/ESP3902/openpose-master/build/examples/tutorial_api_python/Realsense_colour_images"
        cv2.imwrite(os.path.join(path , str(curr_frame) + '_colour_frame.png'), color_image)

        # OpenPose
        imagepath = r'D:\STUDIES\YEAR 3 - SEM 1\ESP3902\openpose-master\build\examples\tutorial_api_python\Realsense_colour_images\0_colour_frame.png'
        datum = op.Datum()
        imageToProcess = cv2.imread(imagepath)
        datum.cvInputData = imageToProcess
        opWrapper.emplaceAndPop([datum])
        np.userskele_json_part = datum.poseKeypoints
        # if(np.userskele_json_part.ndim > 0):   # !!IMPORTANT!! MAKE SURE THERE IS THIS CONDITION TO MAKE THE VARIABLE HAVE PROPER KEYPOINT
        #     print(np.userskele_json_part[0][0][0])
        # if(datum.poseKeypoints.ndim > 0):      # not having to copy the array over to a new variable works as well
        #     print(datum.poseKeypoints[0][4])


        # Stack both images horizontally, but now i only want the colour_image
        images = color_image

################################################################################################################################################################
```

```
216
217         # user skeleton calculation  # NEW VERSION (DATUM)
218         if(datum.poseKeypoints.ndim > 0):
219             # 123
220             if(datum.poseKeypoints[0][2].all and datum.poseKeypoints[0][1].all and datum.poseKeypoints[0][3].all):
221                 if(any(datum.poseKeypoints[0][2]) and any(datum.poseKeypoints[0][1]) and any(datum.poseKeypoints[0][3])):
222                     # print("\n123")
223                     AB = [datum.poseKeypoints[0][1][0] - datum.poseKeypoints[0][2][0], datum.poseKeypoints[0][1][1] - datum.poseKeypoints[0][2][1]]
224                     AC = [datum.poseKeypoints[0][3][0] - datum.poseKeypoints[0][2][0], datum.poseKeypoints[0][3][1] - datum.poseKeypoints[0][2][1]]
225                     # dot_prdt = np.array(AB) @ np.array(AC)
226                     dot_prdt = np.dot(AB, AC)
227                     # print("dot_prdt: ", dot_prdt)
228                     len_AB = math.sqrt(AB[0]**2 + AB[1]**2)
229                     len_AC = math.sqrt(AC[0]**2 + AC[1]**2)
230                     Length = len_AB * len_AC
231                     # print("Length: ", Length)
232                     if(len_AB!=0 and len_AC!=0):
233                         divided_value = np.divide(dot_prdt,Length)
234                         # print("divided_value: ", divided_value)
235                         if(divided_value<1 and divided_value>-1):
236                             user_angle123 = math.degrees(math.acos(divided_value))
237                             # print("user_angle123: ", user_angle123)
238                         else:
239                             user_angle123=0
240                 else:
241                     user_angle123 = 0
242             else:
243                 user_angle123 = 0 # needed else will not work. So in case when either 1 of the parts is null, then just let the angle =0
244
245             # 234
246             if(datum.poseKeypoints[0][2].all and datum.poseKeypoints[0][3].all and datum.poseKeypoints[0][4].all):
247                 if(any(datum.poseKeypoints[0][2]) and any(datum.poseKeypoints[0][3]) and any(datum.poseKeypoints[0][4])):
248                     # print("\n234")
249                     AB = [datum.poseKeypoints[0][2][0] - datum.poseKeypoints[0][3][0], datum.poseKeypoints[0][2][1] - datum.poseKeypoints[0][3][1]]
250                     AC = [datum.poseKeypoints[0][4][0] - datum.poseKeypoints[0][3][0], datum.poseKeypoints[0][4][1] - datum.poseKeypoints[0][3][1]]
251                     # dot_prdt = np.array(AB) @ np.array(AC)
252                     dot_prdt = np.dot(AB, AC)
253                     # print("dot_prdt: ", dot_prdt)
254                     len_AB = math.sqrt(AB[0]**2 + AB[1]**2)
255                     len_AC = math.sqrt(AC[0]**2 + AC[1]**2)
256                     Length = len_AB * len_AC
257                     # print("Length: ", Length)
258                     if(len_AB!=0 and len_AC!=0):
```

```
                    divided_value = np.divide(dot_prdt,Length)
                    # print("divided_value: ", divided_value)
                    if(divided_value<1 and divided_value>-1):
                        # user_angle234 = math.degrees(math.acos(divided_value))
                        user_angle234 = math.degrees(math.acos(divided_value))
                        # print("user_angle234: ", user_angle234)
                    else:
                        user_angle234=0
                else:
                    user_angle234=0

                # print(user_angle234)
            else:
                user_angle234 = 0
        else:
            user_angle234 = 0 # needed else will not work. So in case when either 1 of the parts is null, then just let the angle =0

        # 814
        if(datum.poseKeypoints[0][8].all and datum.poseKeypoints[0][1].all and datum.poseKeypoints[0][4].all):
            if(any(datum.poseKeypoints[0][2]) and any(datum.poseKeypoints[0][1]) and any(datum.poseKeypoints[0][4])):
                # print("\n813")
                AB = [datum.poseKeypoints[0][8][0] - datum.poseKeypoints[0][1][0], datum.poseKeypoints[0][8][1] - datum.poseKeypoints[0][1][1]]
                AC = [datum.poseKeypoints[0][4][0] - datum.poseKeypoints[0][1][0], datum.poseKeypoints[0][4][1] - datum.poseKeypoints[0][1][1]]
                # dot_prdt = np.array(AB) @ np.array(AC)
                dot_prdt = np.dot(AB, AC)
                # print("dot_prdt: ", dot_prdt)
                len_AB = math.sqrt(AB[0]**2 + AB[1]**2)
                len_AC = math.sqrt(AC[0]**2 + AC[1]**2)
                Length = len_AB * len_AC
                # print("Length: ", Length)
                if(len_AB!=0 and len_AC!=0):
                    divided_value = np.divide(dot_prdt,Length)
                    # print("divided_value: ", divided_value)
                    if(divided_value<1 and divided_value>-1):
                        user_angle814 = math.degrees(math.acos(divided_value))
                        # print("user_angle814: ", user_angle814)
                    else:
                        user_angle814=0
                else:
                    user_angle814 = 0
            else:
                user_angle814 = 0 # needed else will not work. So in case when either 1 of the parts is null, then just let the angle =0

        # Normal of 1-9-12 @ 1-4 @ 1-3
        if(datum.poseKeypoints[0][1].all and datum.poseKeypoints[0][9].all and datum.poseKeypoints[0][12].all and datum.poseKeypoints[0][4].all and datum.poseKeypoints[0][3].all):
            if(any(datum.poseKeypoints[0][1]) and any(datum.poseKeypoints[0][9]) and any(datum.poseKeypoints[0][12]) and any(datum.poseKeypoints[0][4]) and any(datum.poseKeypoints[0][3])):
                depth_keypoint1 = aligned_depth_frame.get_distance(int(datum.poseKeypoints[0][1][0]), int(datum.poseKeypoints[0][1][1])) #A
                depth_keypoint9 = aligned_depth_frame.get_distance(int(datum.poseKeypoints[0][9][0]), int(datum.poseKeypoints[0][9][1])) #B
                depth_keypoint12 = aligned_depth_frame.get_distance(int(datum.poseKeypoints[0][12][0]), int(datum.poseKeypoints[0][12][1])) #C
                depth_keypoint3 = aligned_depth_frame.get_distance(int(datum.poseKeypoints[0][3][0]), int(datum.poseKeypoints[0][3][1])) #D
                depth_keypoint4 = aligned_depth_frame.get_distance(int(datum.poseKeypoints[0][4][0]), int(datum.poseKeypoints[0][4][1])) #E

                point1 = rs.rs2_deproject_pixel_to_point(color_intrin, [int(datum.poseKeypoints[0][1][0]), int(datum.poseKeypoints[0][1][1])], depth_keypoint1)
                point9 = rs.rs2_deproject_pixel_to_point(color_intrin, [int(datum.poseKeypoints[0][9][0]), int(datum.poseKeypoints[0][9][1])], depth_keypoint9)
                point12 = rs.rs2_deproject_pixel_to_point(color_intrin, [int(datum.poseKeypoints[0][12][0]), int(datum.poseKeypoints[0][12][1])], depth_keypoint12)
                point4 = rs.rs2_deproject_pixel_to_point(color_intrin, [int(datum.poseKeypoints[0][4][0]), int(datum.poseKeypoints[0][4][1])], depth_keypoint4)
                point3 = rs.rs2_deproject_pixel_to_point(color_intrin, [int(datum.poseKeypoints[0][3][0]), int(datum.poseKeypoints[0][3][1])], depth_keypoint3)

                len_AB = math.sqrt(math.pow(point1[0] - point9[0], 2) + math.pow(point1[1] - point9[1],2) + math.pow(point1[2] - point9[2], 2)) #1-9
                len_AC = math.sqrt(math.pow(point1[0] - point12[0], 2) + math.pow(point1[1] - point12[1],2) + math.pow(point1[2] - point12[2], 2)) #1-12
                len_AE = math.sqrt(math.pow(point1[0] - point4[0], 2) + math.pow(point1[1] - point4[1],2) + math.pow(point1[2] - point4[2], 2)) #1-4
                len_AD = math.sqrt(math.pow(point1[0] - point3[0], 2) + math.pow(point1[1] - point3[1],2) + math.pow(point1[2] - point3[2], 2)) #1-3

                AB = [point9[0] - point1[0], point9[1] - point1[1], depth_keypoint9-depth_keypoint1]
                AC = [point12[0] - point1[0], point12[1] - point1[1], depth_keypoint12-depth_keypoint1]
                AE = [point4[0] - point1[0], point4[1] - point1[1], depth_keypoint4-depth_keypoint1]
                AD = [point3[0] - point1[0], point3[1] - point1[1], depth_keypoint3-depth_keypoint1]

                if(len_AB != 0 and len_AC != 0 and len_AE != 0 and len_AD != 0):
                    cross_prdt = np.cross(AB, AC)
                    len_cross_prdt=math.sqrt(math.pow(cross_prdt[0], 2) + math.pow(cross_prdt[1], 2) + math.pow(cross_prdt[2], 2))
                    norm_cross_prdt = np.divide(cross_prdt, len_cross_prdt)
                    norm_dot_prdt = np.divide((np.array(norm_cross_prdt) @ np.array(AE)), (len_AE))
                    planar_angle_chest_14 = math.degrees(math.acos(norm_dot_prdt))
                    norm_dot_prdt = np.divide((np.array(norm_cross_prdt) @ np.array(AD)), (len_AD))
                    planar_angle_chest_13 = math.degrees(math.acos(norm_dot_prdt))

                    # print("planar: ", planar_angle_chest)

                # print(planar_angle_chest)

                # distance away from camera warning
                # print(depth_keypoint1)
                if(depth_keypoint1>3.0):
                    images = cv2.rectangle(images, (5, 340) ,(140, 360) , (255, 255, 255), -1)
                    cv2.putText(images,'Move forward',(7, 355),font, 0.5,(0, 0, 255),1,cv2.LINE 4)
```

```
                    bol_wrong = True
                elif(depth_keypoint1<2.0):
                    # print("AJKLSHDLAJSDLKJASLKDJALKSDJLAKSJDLKAJSDLKAJSDLKJASLKDJALKSJDLKASJDLKAJSLKDJALSKJDALKSJDLKAJSDLKAJSDLKJASe")
                    images = cv2.rectangle(images, (5, 340) ,(140, 360) , (255, 255, 255), -1)
                    cv2.putText(images,"Move backward",(7, 355),font, 0.5,(0, 0, 255),1,cv2.LINE_4)
                    # print("asdasdasdasdasdasdasdasdasdas")
                    bol_wrong = True
                else:
                    bol_wrong = False
        else:
            user_angle123 = 0
            user_angle234 = 0
            user_angle814 = 0
            planar_angle_chest_14 = 0
            planar_angle_chest_13 = 0

        # print("123:",user_angle123)
        # print("234:",user_angle234)
        # print("814:",user_angle814)
        # print("14:",planar_angle_chest_14)
        # print("13:",planar_angle_chest_13)



        # User rotation axis. 0-2 in terms of x,y,z
        ser_bytes = ser.readline()
        # print(ser_bytes)
        decoded_bytes = ser_bytes[0:len(ser_bytes)].decode("utf-8").strip('\n').split(',')
        # print(decoded_bytes)
        if"\x00" not in decoded_bytes[0] and decoded_bytes[0]!='':
        # if decoded_bytes[0].isdecimal() == True:
            float_list_values = [float(i) for i in decoded_bytes]
            highest_list_values = max(float_list_values, key=abs)
            highest_axis_index = float_list_values.index(highest_list_values)
            # print(highest_list_values)
            # print("rot index:",highest_axis_index)
        else:
            highest_axis_index = 2 # since we not using z-axis
            highest_list_values = 0 # doesnt matter what i put
        # problem with not reading the latest data from arduiono. it reads the top in a ever-filling container of data from arduino.
        # tried delay => fps worst
        # tried https://stackoverflow.com/questions/1093598/pyserial-how-to-read-the-last-line-sent-from-a-serial-device => doesnt work
        # tried https://stackoverflow.com/questions/1093598/pyserial-how-to-read-the-last-line-sent-from-a-serial-device => doesnt work
```

```


    # User skeleton drawing  # NEW VERSION (DATUM)
    if(datum.poseKeypoints.ndim > 0):
        for i in range(15):
            if(datum.poseKeypoints[0][i].all):  # need to .all to make sure that that keypoint returns true for it to be drawn
                if(any(datum.poseKeypoints[0][i])):
                    cv2.circle(images, (int(datum.poseKeypoints[0][i][0]),int(datum.poseKeypoints[0][i][1])), 5, colour_blue, thickness=line_thickness)


    # Guidance skeleton drawing (Line version)
    if(curr_pose == 1): # change to end pose
        if((user_angle234<guidance_end_angle234-end_angle234_allowance) or (user_angle814<guidance_end_angle814-end_angle814_lowerAllowance or user_angle814>guidance_end_angle814+end_angle814_upperAllowance) or
            (planar_angle_chest_13<guidance_planar_angle_chest_13-planar_angle_chest_13_lowerAllowance or planar_angle_chest_13>guidance_planar_angle_chest_13+planar_angle_chest_13_upperAllowance) or
            (planar_angle_chest_14<guidance_planar_angle_chest_14-planar_angle_chest_14_lowerAllowance or planar_angle_chest_14>guidance_planar_angle_chest_14+planar_angle_chest_14_upperAllowance) or
            (curr_rot==1 and (highest_axis_index!=0 or highest_list_values<0)) or (curr_rot==2 and (highest_axis_index!=0 or highest_list_values>0)) or
            bol_wrong == True): # outside tolerance, red colour
            #0 - 1
            cv2.line(images, (int(guidance_skele_part_end[0]['0'][0]),int(guidance_skele_part_end[0]['0'][1])), (int(guidance_skele_part_end[0]['1'][0]),int(guidance_skele_part_end[0]['1'][1])), colour_red, thickness=line_thickness)
            #1 - 2
            cv2.line(images, (int(guidance_skele_part_end[0]['2'][0]),int(guidance_skele_part_end[0]['2'][1])), (int(guidance_skele_part_end[0]['1'][0]),int(guidance_skele_part_end[0]['1'][1])), colour_red, thickness=line_thickness)
            #1 - 5
            cv2.line(images, (int(guidance_skele_part_end[0]['5'][0]),int(guidance_skele_part_end[0]['5'][1])), (int(guidance_skele_part_end[0]['1'][0]),int(guidance_skele_part_end[0]['1'][1])), colour_red, thickness=line_thickness)
            #1 - 8
            cv2.line(images, (int(guidance_skele_part_end[0]['8'][0]),int(guidance_skele_part_end[0]['8'][1])), (int(guidance_skele_part_end[0]['1'][0]),int(guidance_skele_part_end[0]['1'][1])), colour_red, thickness=line_thickness)
            #2 - 3
            cv2.line(images, (int(guidance_skele_part_end[0]['2'][0]),int(guidance_skele_part_end[0]['2'][1])), (int(guidance_skele_part_end[0]['3'][0]),int(guidance_skele_part_end[0]['3'][1])), colour_red, thickness=line_thickness)
            #3 - 4
            cv2.line(images, (int(guidance_skele_part_end[0]['3'][0]),int(guidance_skele_part_end[0]['3'][1])), (int(guidance_skele_part_end[0]['4'][0]),int(guidance_skele_part_end[0]['4'][1])), colour_red, thickness=line_thickness)
            #5 - guidance_skele_part_end
            cv2.line(images, (int(guidance_skele_part_end[0]['5'][0]),int(guidance_skele_part_end[0]['5'][1])), (int(guidance_skele_part_end[0]['6'][0]),int(guidance_skele_part_end[0]['6'][1])), colour_red, thickness=line_thickness)
            #6 - 7
            cv2.line(images, (int(guidance_skele_part_end[0]['6'][0]),int(guidance_skele_part_end[0]['6'][1])), (int(guidance_skele_part_end[0]['7'][0]),int(guidance_skele_part_end[0]['7'][1])), colour_red, thickness=line_thickness)
            #8 - 9
            cv2.line(images, (int(guidance_skele_part_end[0]['8'][0]),int(guidance_skele_part_end[0]['8'][1])), (int(guidance_skele_part_end[0]['9'][0]),int(guidance_skele_part_end[0]['9'][1])), colour_red, thickness=line_thickness)
            #8 - 12
            cv2.line(images, (int(guidance_skele_part_end[0]['8'][0]),int(guidance_skele_part_end[0]['8'][1])), (int(guidance_skele_part_end[0]['12'][0]),int(guidance_skele_part_end[0]['12'][1])), colour_red, thickness=line_thickness)
            #9 - 10
            cv2.line(images, (int(guidance_skele_part_end[0]['9'][0]),int(guidance_skele_part_end[0]['9'][1])), (int(guidance_skele_part_end[0]['10'][0]),int(guidance_skele_part_end[0]['10'][1])), colour_red, thickness=line_thickness)
            #10 - 11
            cv2.line(images, (int(guidance_skele_part_end[0]['10'][0]),int(guidance_skele_part_end[0]['10'][1])), (int(guidance_skele_part_end[0]['11'][0]),int(guidance_skele_part_end[0]['11'][1])), colour_red, thickness=line_thickness)
            #12 - 13
            cv2.line(images, (int(guidance_skele_part_end[0]['12'][0]),int(guidance_skele_part_end[0]['12'][1])), (int(guidance_skele_part_end[0]['13'][0]),int(guidance_skele_part_end[0]['13'][1])), colour_red, thickness=line_thickness)
            #13 - 14
            cv2.line(images, (int(guidance_skele_part_end[0]['13'][0]),int(guidance_skele_part_end[0]['13'][1])), (int(guidance_skele_part_end[0]['14'][0]),int(guidance_skele_part_end[0]['14'][1])), colour_red, thickness=line_thickness)

            # Error warnings
            #rotation
            if(curr_rot==1 and (highest_axis_index!=0 or highest_list_values<0)):
                # Draw a rectangle
                images = cv2.rectangle(images, (0, 0), (213, 70), (255, 255, 255), -1)  # frame to draw, top left, bot right, colour, -1
                cv2.putText(images,'Rotate your hand until',(5, 15),font, 0.5,(0, 0, 255),1,cv2.LINE_4)
```

```
                    cv2.putText(images,'your palm faces the',(5, 40),font, 0.5,(0, 0, 255),1,cv2.LINE_4)
                    cv2.putText(images,'floor',(5, 65),font, 0.5,(0, 0, 255),1,cv2.LINE_4)
                elif(curr_rot==2 and (highest_axis_index!=0 or highest_list_values>0)):
                    # Draw a rectangle
                    images = cv2.rectangle(images, (0, 0), (213, 70), (255, 255, 255), -1)
                    cv2.putText(images,'Rotate your hand until',(5, 15),font, 0.5,(0, 0, 255),1,cv2.LINE_4)
                    cv2.putText(images,'your palm faces the',(5, 40),font, 0.5,(0, 0, 255),1,cv2.LINE_4)
                    cv2.putText(images,'ceiling',(5, 65),font, 0.5,(0, 0, 255),1,cv2.LINE_4)
                # elbow
                if(user_angle234<guidance_end_angle234-end_angle234_allowance):
                    images = cv2.rectangle(images, (225, 340) , (425, 360) , (255, 255, 255), -1)
                    cv2.putText(images,'Straighten your arm',(230, 355),font, 0.5,(0, 0, 255),1,cv2.LINE_4)
                # up, down
                if(user_angle814<guidance_end_angle814-end_angle814_lowerAllowance):
                    images = cv2.rectangle(images, (228, 0) , (405, 20) , (255, 255, 255), -1)
                    cv2.putText(images,'Move your arm up',(232, 15),font, 0.5,(0, 0, 255),1,cv2.LINE_4)
                elif (user_angle814>guidance_end_angle814+end_angle814_upperAllowance):
                    images = cv2.rectangle(images, (216, 0) , (406, 20) , (255, 255, 255), -1)
                    cv2.putText(images,'Move your arm down',(218, 15),font, 0.5,(0, 0, 255),1,cv2.LINE_4)
                # front, back
                if(planar_angle_chest_13<guidance_planar_angle_chest_13-planar_angle_chest_13_lowerAllowance and planar_angle_chest_14<guidance_planar_angle_chest_14-planar_angle_chest_14_lowerAllowance):
                    images = cv2.rectangle(images, (410, 0) ,(640, 20) , (255, 255, 255), -1)
                    cv2.putText(images,'Move your arm backward',(413, 15),font, 0.5,(0, 0, 255),1,cv2.LINE_4)
                elif(planar_angle_chest_13>guidance_planar_angle_chest_13+planar_angle_chest_13_upperAllowance and planar_angle_chest_14>guidance_planar_angle_chest_14+planar_angle_chest_14_upperAllowance):
                    images = cv2.rectangle(images, (420, 0) ,(640, 20) , (255, 255, 255), -1)
                    cv2.putText(images,'Move your arm forward',(425, 15),font, 0.5,(0, 0, 255),1,cv2.LINE_4)
                else:
                    images = cv2.rectangle(images, (420, 0) ,(640, 20) , (255, 255, 255), -1)
                    cv2.putText(images,'Straighten your elbow',(425, 15),font, 0.5,(0, 0, 255),1,cv2.LINE_4)

            else: # within tolerance, green colour
                #0 - 1
                cv2.line(images, (int(guidance_skele_part_end[0]['0'][0]),int(guidance_skele_part_end[0]['0'][1])), (int(guidance_skele_part_end[0]['1'][0]),int(guidance_skele_part_end[0]['1'][1])), colour_green, thickness=line_thickness)
                #1 - 2
                cv2.line(images, (int(guidance_skele_part_end[0]['2'][0]),int(guidance_skele_part_end[0]['2'][1])), (int(guidance_skele_part_end[0]['1'][0]),int(guidance_skele_part_end[0]['1'][1])), colour_green, thickness=line_thickness)
                #1 - 5
                cv2.line(images, (int(guidance_skele_part_end[0]['5'][0]),int(guidance_skele_part_end[0]['5'][1])), (int(guidance_skele_part_end[0]['1'][0]),int(guidance_skele_part_end[0]['1'][1])), colour_green, thickness=line_thickness)
                #1 - 8
                cv2.line(images, (int(guidance_skele_part_end[0]['8'][0]),int(guidance_skele_part_end[0]['8'][1])), (int(guidance_skele_part_end[0]['1'][0]),int(guidance_skele_part_end[0]['1'][1])), colour_green, thickness=line_thickness)
                #2 - 3
                cv2.line(images, (int(guidance_skele_part_end[0]['2'][0]),int(guidance_skele_part_end[0]['2'][1])), (int(guidance_skele_part_end[0]['3'][0]),int(guidance_skele_part_end[0]['3'][1])), colour_green, thickness=line_thickness)
                #3 - 4
                cv2.line(images, (int(guidance_skele_part_end[0]['3'][0]),int(guidance_skele_part_end[0]['3'][1])), (int(guidance_skele_part_end[0]['4'][0]),int(guidance_skele_part_end[0]['4'][1])), colour_green, thickness=line_thickness)
                #5 - guidance_skele_part_end
                cv2.line(images, (int(guidance_skele_part_end[0]['5'][0]),int(guidance_skele_part_end[0]['5'][1])), (int(guidance_skele_part_end[0]['6'][0]),int(guidance_skele_part_end[0]['6'][1])), colour_green, thickness=line_thickness)
                #6 - 7
                cv2.line(images, (int(guidance_skele_part_end[0]['6'][0]),int(guidance_skele_part_end[0]['6'][1])), (int(guidance_skele_part_end[0]['7'][0]),int(guidance_skele_part_end[0]['7'][1])), colour_green, thickness=line_thickness)
                #8 - 9
                cv2.line(images, (int(guidance_skele_part_end[0]['8'][0]),int(guidance_skele_part_end[0]['8'][1])), (int(guidance_skele_part_end[0]['9'][0]),int(guidance_skele_part_end[0]['9'][1])), colour_green, thickness=line_thickness)
                #8 - 12
                cv2.line(images, (int(guidance_skele_part_end[0]['8'][0]),int(guidance_skele_part_end[0]['8'][1])), (int(guidance_skele_part_end[0]['12'][0]),int(guidance_skele_part_end[0]['12'][1])), colour_green, thickness=line_thickness)
                #9 - 10
                cv2.line(images, (int(guidance_skele_part_end[0]['9'][0]),int(guidance_skele_part_end[0]['9'][1])), (int(guidance_skele_part_end[0]['10'][0]),int(guidance_skele_part_end[0]['10'][1])), colour_green, thickness=line_thickness)
                #10 - 11
                cv2.line(images, (int(guidance_skele_part_end[0]['10'][0]),int(guidance_skele_part_end[0]['10'][1])), (int(guidance_skele_part_end[0]['11'][0]),int(guidance_skele_part_end[0]['11'][1])), colour_green, thickness=line_thickness)
                #12 - 13
                cv2.line(images, (int(guidance_skele_part_end[0]['12'][0]),int(guidance_skele_part_end[0]['12'][1])), (int(guidance_skele_part_end[0]['13'][0]),int(guidance_skele_part_end[0]['13'][1])), colour_green, thickness=line_thickness)
                #13 - 14
                cv2.line(images, (int(guidance_skele_part_end[0]['13'][0]),int(guidance_skele_part_end[0]['13'][1])), (int(guidance_skele_part_end[0]['14'][0]),int(guidance_skele_part_end[0]['14'][1])), colour_green, thickness=line_thickness)
                #CHANGE TO START POSE AFTER PART TWO OF ROTATION
                if(curr_rot == 1):
                    curr_rot=2
                elif(curr_rot == 2):
                    curr_pose=0
                    curr_rot=0


        else: # starting pose
            if((user_angle123<guidance_start_angle123-start_angle123_lowerAllowance or user_angle123>guidance_start_angle123+start_angle123_upperAllowance) or user_angle234<guidance_start_angle234-start_angle234_allowance) or (highest_axis_index!=1 or highest_list_values<0) or
                bol_wrong ==  True): # outside tolerance, red colour
                #0 - 1
                cv2.line(images, (int(guidance_skele_part_start[0]['0'][0]),int(guidance_skele_part_start[0]['0'][1])), (int(guidance_skele_part_start[0]['1'][0]),int(guidance_skele_part_start[0]['1'][1])), colour_red, thickness=line_thickness)
                #1 - 2
                cv2.line(images, (int(guidance_skele_part_start[0]['2'][0]),int(guidance_skele_part_start[0]['2'][1])), (int(guidance_skele_part_start[0]['1'][0]),int(guidance_skele_part_start[0]['1'][1])), colour_red, thickness=line_thickness)
                #1 - 5
                cv2.line(images, (int(guidance_skele_part_start[0]['5'][0]),int(guidance_skele_part_start[0]['5'][1])), (int(guidance_skele_part_start[0]['1'][0]),int(guidance_skele_part_start[0]['1'][1])), colour_red, thickness=line_thickness)
                #1 - 8
                cv2.line(images, (int(guidance_skele_part_start[0]['8'][0]),int(guidance_skele_part_start[0]['8'][1])), (int(guidance_skele_part_start[0]['1'][0]),int(guidance_skele_part_start[0]['1'][1])), colour_red, thickness=line_thickness)
                #2 - 3
                cv2.line(images, (int(guidance_skele_part_start[0]['2'][0]),int(guidance_skele_part_start[0]['2'][1])), (int(guidance_skele_part_start[0]['3'][0]),int(guidance_skele_part_start[0]['3'][1])), colour_red, thickness=line_thickness)
                #3 - 4
                cv2.line(images, (int(guidance_skele_part_start[0]['3'][0]),int(guidance_skele_part_start[0]['3'][1])), (int(guidance_skele_part_start[0]['4'][0]),int(guidance_skele_part_start[0]['4'][1])), colour_red, thickness=line_thickness)
                #5 - 6
                cv2.line(images, (int(guidance_skele_part_start[0]['5'][0]),int(guidance_skele_part_start[0]['5'][1])), (int(guidance_skele_part_start[0]['6'][0]),int(guidance_skele_part_start[0]['6'][1])), colour_red, thickness=line_thickness)
                #6 - 7
                cv2.line(images, (int(guidance_skele_part_start[0]['6'][0]),int(guidance_skele_part_start[0]['6'][1])), (int(guidance_skele_part_start[0]['7'][0]),int(guidance_skele_part_start[0]['7'][1])), colour_red, thickness=line_thickness)
                #8 - 9
                cv2.line(images, (int(guidance_skele_part_start[0]['8'][0]),int(guidance_skele_part_start[0]['8'][1])), (int(guidance_skele_part_start[0]['9'][0]),int(guidance_skele_part_start[0]['9'][1])), colour_red, thickness=line_thickness)
                #8 - 12
                cv2.line(images, (int(guidance_skele_part_start[0]['8'][0]),int(guidance_skele_part_start[0]['8'][1])), (int(guidance_skele_part_start[0]['12'][0]),int(guidance_skele_part_start[0]['12'][1])), colour_red, thickness=line_thickness)
                #9 - 10
                cv2.line(images, (int(guidance_skele_part_start[0]['9'][0]),int(guidance_skele_part_start[0]['9'][1])), (int(guidance_skele_part_start[0]['10'][0]),int(guidance_skele_part_start[0]['10'][1])), colour_red, thickness=line_thickness)
                #10 - 11
                cv2.line(images, (int(guidance_skele_part_start[0]['10'][0]),int(guidance_skele_part_start[0]['10'][1])), (int(guidance_skele_part_start[0]['11'][0]),int(guidance_skele_part_start[0]['11'][1])), colour_red, thickness=line_thickness)
                #12 - 13
                cv2.line(images, (int(guidance_skele_part_start[0]['12'][0]),int(guidance_skele_part_start[0]['12'][1])), (int(guidance_skele_part_start[0]['13'][0]),int(guidance_skele_part_start[0]['13'][1])), colour_red, thickness=line_thickness)
                #13 - 14
                cv2.line(images, (int(guidance_skele_part_start[0]['13'][0]),int(guidance_skele_part_start[0]['13'][1])), (int(guidance_skele_part_start[0]['14'][0]),int(guidance_skele_part_start[0]['14'][1])), colour_red, thickness=line_thickness)

                #rotation
                if(highest_axis_index!=1 or highest_list_values<0):
                    images = cv2.rectangle(images, (0, 0), (235, 50), (255, 255, 255), -1)  # frame to draw, top left, bot right, colour, -1
                    cv2.putText(images,'Rotate your hand until',(5, 15),font, 0.5,(0, 0, 255),1,cv2.LINE_4)
                    cv2.putText(images,'your palm faces your leg',(5, 40),font, 0.5,(0, 0, 255),1,cv2.LINE_4)



            else: # within tolerance, green colour
                #0 - 1
                cv2.line(images, (int(guidance_skele_part_start[0]['0'][0]),int(guidance_skele_part_start[0]['0'][1])), (int(guidance_skele_part_start[0]['1'][0]),int(guidance_skele_part_start[0]['1'][1])), colour_green, thickness=line_thickness)
                #1 - 2
                cv2.line(images, (int(guidance_skele_part_start[0]['2'][0]),int(guidance_skele_part_start[0]['2'][1])), (int(guidance_skele_part_start[0]['1'][0]),int(guidance_skele_part_start[0]['1'][1])), colour_green, thickness=line_thickness)
```

```
                #1 - 5
                cv2.line(images, (int(guidance_skele_part_start[0]['5'][0]),int(guidance_skele_part_start[0]['5'][1])), (int(guidance_skele_part_start[0]['1'][0]),int(guidance_skele_part_start[0]['1'][1])), colour_green, thickness=line_thickness)
                #1 - 8
                cv2.line(images, (int(guidance_skele_part_start[0]['8'][0]),int(guidance_skele_part_start[0]['8'][1])), (int(guidance_skele_part_start[0]['1'][0]),int(guidance_skele_part_start[0]['1'][1])), colour_green, thickness=line_thickness)
                #2 - 3
                cv2.line(images, (int(guidance_skele_part_start[0]['2'][0]),int(guidance_skele_part_start[0]['2'][1])), (int(guidance_skele_part_start[0]['3'][0]),int(guidance_skele_part_start[0]['3'][1])), colour_green, thickness=line_thickness)
                #3 - 4
                cv2.line(images, (int(guidance_skele_part_start[0]['3'][0]),int(guidance_skele_part_start[0]['3'][1])), (int(guidance_skele_part_start[0]['4'][0]),int(guidance_skele_part_start[0]['4'][1])), colour_green, thickness=line_thickness)
                #5 - 6
                cv2.line(images, (int(guidance_skele_part_start[0]['5'][0]),int(guidance_skele_part_start[0]['5'][1])), (int(guidance_skele_part_start[0]['6'][0]),int(guidance_skele_part_start[0]['6'][1])), colour_green, thickness=line_thickness)
                #6 - 7
                cv2.line(images, (int(guidance_skele_part_start[0]['6'][0]),int(guidance_skele_part_start[0]['6'][1])), (int(guidance_skele_part_start[0]['7'][0]),int(guidance_skele_part_start[0]['7'][1])), colour_green, thickness=line_thickness)
                #8 - 9
                cv2.line(images, (int(guidance_skele_part_start[0]['8'][0]),int(guidance_skele_part_start[0]['8'][1])), (int(guidance_skele_part_start[0]['9'][0]),int(guidance_skele_part_start[0]['9'][1])), colour_green, thickness=line_thickness)
                #8 - 12
                cv2.line(images, (int(guidance_skele_part_start[0]['8'][0]),int(guidance_skele_part_start[0]['8'][1])), (int(guidance_skele_part_start[0]['12'][0]),int(guidance_skele_part_start[0]['12'][1])), colour_green, thickness=line_thickness)
                #9 - 10
                cv2.line(images, (int(guidance_skele_part_start[0]['9'][0]),int(guidance_skele_part_start[0]['9'][1])), (int(guidance_skele_part_start[0]['10'][0]),int(guidance_skele_part_start[0]['10'][1])), colour_green, thickness=line_thickness)
                #10 - 11
                cv2.line(images, (int(guidance_skele_part_start[0]['10'][0]),int(guidance_skele_part_start[0]['10'][1])), (int(guidance_skele_part_start[0]['11'][0]),int(guidance_skele_part_start[0]['11'][1])), colour_green, thickness=line_thickness)
                #12 - 13
                cv2.line(images, (int(guidance_skele_part_start[0]['12'][0]),int(guidance_skele_part_start[0]['12'][1])), (int(guidance_skele_part_start[0]['13'][0]),int(guidance_skele_part_start[0]['13'][1])), colour_green, thickness=line_thickness)
                #13 - 14
                cv2.line(images, (int(guidance_skele_part_start[0]['13'][0]),int(guidance_skele_part_start[0]['13'][1])), (int(guidance_skele_part_start[0]['14'][0]),int(guidance_skele_part_start[0]['14'][1])), colour_green, thickness=line_thickness)
                #CHANGE TO END POSE
                curr_pose = 1
                curr_rot = 1
                if(bol_start != True):
                    counter += 1

                bol_start = False

        csv_frames += 1

        # print("user_angle123= ", user_angle123)
        # print("user_angle234= ", user_angle234)
        # print("user_angle813= ", user_angle813)
        # print("planar_angle_chest= ", planar_angle_chest)
        # print("Arduino= ", highest_axis_index, " ", highest_list_values)

        key = cv2.waitKey(1)
        if key == ord('h') and help_switch == 0:
            help_switch = 1
        elif key == ord('h') and help_switch == 1:
            help_switch = 0

        # for the user guide
        cv2.putText(images,'press h for help', (490, 350), font, 0.5,  (0, 0, 255),  1,  cv2.LINE_4)

        # draw circle w/ line across
        images = cv2.circle(images, (600, 75), 36, (0, 0, 0), -1)
        images = cv2.line(images, (630,55), (578,100), (255,255,255), 2)
        cv2.putText(images,  str(rep_value),  (603, 95),  font, 0.6,  (255, 255, 255),  1,  cv2.LINE_4)
        cv2.putText(images,  str(counter),  (588, 68),  font, 0.6,  (255, 255, 255),  1,  cv2.LINE_4)

        # for the help screen
        if help_switch == 1:
            images = cv2.rectangle(images, (0,0), (640,360), (255, 255, 255), -1)
```

```
                cv2.putText(images, 'User guide:',(5,15), font, 0.4, (0, 0, 0), 1, cv2.LINE_4)
                images = cv2.line(images, (5,17), (85,17), (0,0,0), 1)
                cv2.putText(images, 'Blue',(5,30), font, 0.4, (255, 0, 0), 1, cv2.LINE_4)
                cv2.putText(images, 'circles represent your',  (40,30), font, 0.4,  (0, 0, 0),  1,  cv2.LINE_4)
                cv2.putText(images,  'skeleton.',  (205,30), font, 0.4,  (255, 0, 0),  1,  cv2.LINE_4)
                cv2.putText(images,  'The',  (5,45),  font, 0.4,  (0, 0, 0),  1,  cv2.LINE_4)
                cv2.putText(images,  'red',  (35,45), font, 0.4,  (0, 0, 255), 1,  cv2.LINE_4)
                cv2.putText(images, 'lines represent the', (65,45), font, 0.4, (0, 0, 0),  1,  cv2.LINE_4)
                cv2.putText(images, 'correct',  (205,45), font, 0.4, (0, 0, 255),  1, cv2.LINE_4)
                cv2.putText(images, 'pose.',  (265,45), font, 0.4, (0, 0, 0), 1, cv2.LINE_4)
                cv2.putText(images,  'If the',  (5,60), font, 0.4,  (0, 0, 0), 1,  cv2.LINE_4)
                cv2.putText(images,  'red',  (50,60), font, 0.4,  (0, 0, 255),  1,  cv2.LINE_4)
                cv2.putText(images, 'lines turn',  (80,60), font, 0.4,  (0, 0, 0),  1,  cv2.LINE_4)
                cv2.putText(images, 'green',  (160,60), font, 0.4, (34, 139, 34), 1,  cv2.LINE_4)
                cv2.putText(images,', correct pose is', (200,60), font, 0.4, (0, 0, 0), 1,  cv2.LINE_4)
                cv2.putText(images,  'achieved!',  (320,60), font, 0.4,  (0, 69, 255), 1,  cv2.LINE_4)
                cv2.putText(images,  'If not, warning messages will appear to tell you how to correct it:',  (5, 75),  font, 0.4, (0, 0, 0),  1, cv2.LINE_4)
                cv2.putText(images,  '- Top left: rotation of arms',  (5, 95),  font, 0.4,  (0, 0, 0),  1,  cv2.LINE_4)
                cv2.putText(images,  '- Top middle: up or down movement of the arms',  (5, 110),  font, 0.4,  (0, 0, 0),  1,  cv2.LINE_4)
                cv2.putText(images,  '- Top right: front or back movement of the arms',  (5, 125), font, 0.4,  (0, 0, 0), 1,  cv2.LINE_4)
                cv2.putText(images, '- Bottom left: how far you are from the camera', (5, 140), font, 0.4, (0, 0, 0), 1, cv2.LINE_4)
                cv2.putText(images,  '- Bottom middle: elbow corrections',  (5, 155),  font, 0.4,  (0, 0, 0), 1, cv2.LINE_4)
                cv2.putText(images, 'The number at the top right indicates how many times the exercise is completed.', (5, 175),  font, 0.4,  (0, 0, 0),  1, cv2.LINE_4)
                cv2.putText(images,  'How to:',  (5,200), font, 0.4,  (0, 0, 0),  1,  cv2.LINE_4)
                images = cv2.line(images, (5,202), (55,202), (0,0,0), 1)
                cv2.putText(images,  '1) Go through the gif in the selection screen before proceeding with the exercise.',  (5, 215),  font, 0.4,  (0, 0, 0),  1, cv2.LINE_4)
                cv2.putText(images,  '2) Stand 2 to 3m away from the camera.',  (5, 230),  font, 0.4,  (0, 0, 0),  1, cv2.LINE_4)
                cv2.putText(images,  '3) The correct pose is shown by the red skeleton in the middle of the screen.',  (5, 245),  font, 0.4,  (0, 0, 0),  1, cv2.LINE_4)
                cv2.putText(images,  '4) There is no need to align your skeleton to the red skeleton.',  (5, 260),  font, 0.4,  (0, 0, 0),  1, cv2.LINE_4)
                cv2.putText(images,  "5) Follow the red skeleton's pose.",  (5, 275),  font, 0.4,  (0, 0, 0),  1, cv2.LINE_4)
                cv2.putText(images,  '6) Any difference between your skeleton and the red skeleton will be shown as',  (5, 290),  font, 0.4,  (0, 0, 0),  1, cv2.LINE_4)
                cv2.putText(images,  'warnings around the screen.',  (5, 305),  font, 0.4,  (0, 0, 0),  1, cv2.LINE_4)
                cv2.putText(images,  '7) When one cycle is completed, the counter on the right will increment by one.',  (5, 320),  font, 0.4,  (0, 0, 0),  1, cv2.LINE_4)
                cv2.putText(images,  'press h to exit',  (525, 355),  font, 0.4,  (0, 0, 255),  1,  cv2.LINE_4)

            # make screen full screen
            cv2.namedWindow("RealSense", cv2.WND_PROP_FULLSCREEN)
            cv2.setWindowProperty("RealSense",cv2.WND_PROP_FULLSCREEN,cv2.WINDOW_FULLSCREEN)

```

```
        # Show images
        cv2.imshow("RealSense", images)
        # key = cv2.waitKey(1)
        # Press esc or 'q' to close the image window
        if key & 0xFF == ord('q') or key == 27:
            # sys.exit()
            # cv2.destroyAllWindows()
            cv2.destroyWindow("RealSense")
            break
        # elif key == ord('h') and help_switch == 0:
        #     help_switch = 1
        # elif key == ord('h') and help_switch == 1:
        #     help_switch = 0


finally:

    # Stop streaming
    pipeline.stop()

```

Appendix III
(Arduino code)

```
#include <Wire.h>

long accelX, accelY, accelZ;
float gForceX, gForceY, gForceZ;

long gyroX, gyroY, gyroZ;
float rotX, rotY, rotZ;

unsigned long previousMillis = 0;
const long interval = 125;

void setup() {
  Serial.begin(9600);
  Wire.begin();
  setupMPU();
}


void loop() {
  recordAccelRegisters();
  recordGyroRegisters();
  printData();
  delay(100);
}

void setupMPU(){
  Wire.beginTransmission(0b1101000); //This is the I2C address of the MPU (b1101000/b1101001 for AC0 low/high datasheet sec. 9.2)
  Wire.write(0x6B); //Accessing the register 6B - Power Management (Sec. 4.28)
  Wire.write(0b00000000); //Setting SLEEP register to 0. (Required; see Note on p. 9)
  Wire.endTransmission();

   Wire.beginTransmission(0b1101000); //I2C address of the MPU
   Wire.write(0x1B); //Accessing the register 1B - Gyroscope Configuration (Sec. 4.4)
   Wire.write(0x00000000); //Setting the gyro to full scale +/- 250deg./s
   Wire.endTransmission();
   Wire.beginTransmission(0b1101000); //I2C address of the MPU
   Wire.write(0x1C); //Accessing the register 1C - Acccelerometer Configuration (Sec. 4.5)
   Wire.write(0b00000000); //Setting the accel to +/- 2g
   Wire.endTransmission();
}

void recordAccelRegisters() {
  Wire.beginTransmission(0b1101000); //I2C address of the MPU
  Wire.write(0x3B); //Starting register for Accel Readings
  Wire.endTransmission();
  Wire.requestFrom(0b1101000,6); //Request Accel Registers (3B - 40)
  while(Wire.available() < 6);
  accelX = Wire.read()<<8|Wire.read(); //Store first two bytes into accelX
  accelY = Wire.read()<<8|Wire.read(); //Store middle two bytes into accelY
  accelZ = Wire.read()<<8|Wire.read(); //Store last two bytes into accelZ
  processAccelData();
}

void processAccelData(){
  gForceX = accelX / 16384.0;
  gForceY = accelY / 16384.0;
  gForceZ = accelZ / 16384.0;
}

void recordGyroRegisters() {
  Wire.beginTransmission(0b1101000); //I2C address of the MPU
```

```
  Wire.write(0x43); //Starting register for Gyro Readings
  Wire.endTransmission();
  Wire.requestFrom(0b1101000,6); //Request Gyro Registers (43 - 48)
  while(Wire.available() < 6);
  gyroX = Wire.read()<<8|Wire.read(); //Store first two bytes into accelX
  gyroY = Wire.read()<<8|Wire.read(); //Store middle two bytes into accelY
  gyroZ = Wire.read()<<8|Wire.read(); //Store last two bytes into accelZ
  processGyroData();
}

void processGyroData() {
  rotX = gyroX / (131.0*250);
  rotY = gyroY / (131.0*250);
  rotZ = gyroZ / (131.0*250);
}

void printData() {
      recordAccelRegisters();
      recordGyroRegisters();
    Serial.print(gForceX,3);
    Serial.print(",");
    Serial.print(gForceY,3);
    Serial.print(",");
    Serial.print(gForceZ,3);
    Serial.print("\n");
    delay(10);
  }
```