Digitalization of the Physical Performance Test and Training

Master Thesis

In partial fulfillment of the requirements for the degree

"Master of Science in Engineering"

Study program:

Mechatronics & Smart Technologies - Electrical Engineering

Management Center Innsbruck

Supervisor:

Prof. Yeongmi Kim, PhD

Prof. Dr. Daniel Wiznia, MD

Prof. Dr. Rummana Aslam, MBBS

Author:

Ing. Wolfgang Grosek, BSc

2110620001

Declaration in Lieu of Oath

„I hereby declare, under oath, that this master thesis has been my independent work and has not been aided with any prohibited means. I declare, to the best of my knowledge and belief, that all passages taken from published and unpublished sources or docments have been reproduced whether as original, slightlychanged or in thought, have been mentioned as such at the corresponding placesof the thesis, by citation, where the extend of the original quotes is indicated.“

Canmore, 24.09.2023

Place, Date

Signature

Acknowledgement

I would like to express my deepest appreciation to my supervisors and mentors at Yale University and MCI. Starting with Prof. Yeongmi Kim, PhD, who helped me to define my project idea right from the beginning and also helped me a lot throughout the course of the project. Prof. Dr. Daniel Wiznia, MD, Prof. Dr. Rummana Aslam, MBBS, Larry Wilen, Ph.D., and Vincent Wilczynski were crucial for the success of the project and helped me wherever they could. I could always count on them and it was a pleasure and honor to work with them at Yale University.
I also had great pleasure of working with Necolle Morgado-Vega, DO and her team at YNHH Rehabilitation Center. They supported me a lot during the clinical trial and helped me to use the sensors on patients in their Rehabilitation Center.
Special thanks to my friends all over the world, who I met during my stay in New Haven and friends and family, who visited me during that time. They truly helped me to enjoy my stay to the fullest and helped me to balance the stressful academic tasks.

Abstract

The measurement of the physical functions of the human body is of big interest, especially after a surgery, after an accident and for disabled or elderly people. After special training or rehabilitation, the progress can be analyzed via measurement. The Modified Physical Performance Test consists of different tasks and with a point system the performance of physical functions can be determined. However, until now this test is carried out with a supervisor and a stopwatch. This leads to inaccuracy, aberrance and missing data.

The aim of the digitalization of the Modified Physical Performance Test is to provide objective test results. Wearable sensors, smart objects and computer software were developed to measure the body position and movement. This will help to determine patient progress, trigger timers and calculate a test score. All these factors will eliminate inaccuracy and aberrance and facilitate the task of the supervisor. A training mode was developed, which motivates the patient to further train their balance. This training mode will help to improve future test scores.

A clinical trial with ten patients of the Yale rehabilitation unit in Milford hospital was conducted. The outcome of the work showed a great correlation between the digital test score and the evaluation of specialists in the corresponding field of rehabilitation via the traditional Modified Physical Performance Test score.

It was shown that the proposed Digitalization of the Physical Performance Test might be used for various patient groups. The test procedure is feasible and offers numerous advantages compared to the traditional test.

Keywords: IMU, joint replacement, orthopedics, Parkinson's disease, Physical Performance Test, rehabilitation, stroke, wearable.

Contents

1. Introduction

1.1. Problem Statement and Motivation

The measurement of the physical functions of the human body is of big interest, especially after a surgery, after an accident and for disabled or elderly people. After special training or rehabilitation, the progress can be analyzed via measurement. The Modified Physical Performance Test (MPPT) consists of different tasks and with a point system the performance of physical functions can be determined. However, until now this test is carried out with a supervisor and a stopwatch. This leads to inaccuracy, aberrance and missing data.

1.2. Aim of the Project

The aim of the digitalization of the Modified Physical Performance Test is to standardize the test results. With the help of wearable sensors, smart objects and computer software, it is possible to measure the body position and movement. This will help to determine patient progress, trigger timers and calculate a test score. All these factors will eliminate inaccuracy and aberrance and facilitate the task of the supervisor.

1.3. Objective

The main objective of the thesis is the creation of a test to measure the physical performance of patients. The population group consists of different patients, who all struggle to do tasks of daily living. The sub tasks of the test measure different body functions and can give an overview over the areas that the patient struggles with. The end result of the test is a percentage score, that helps to show the abilities of the patient. The test can be used over time to show rehabilitation progress.
After a literature research, the needed sensors for an evaluation of the patient movement have to be defined. The selected sensor hardware has to be combined with an microcontroller and a power source and must be embedded in a case. Additionally, a device to measure balance must be developed. The measurements of the sensors are evaluated at the end of the thesis.
The wearable sensor of the microcontroller has to capture the body movement of the

patient and send the measured values to the PC software. The PC software must evaluate those measured body movements of the patient. Additionally, the read data and the test score should be saved to a file. In that way further data analysis can be carried out.

1.4. Method and Planned Strategy

The planned strategy in order to fulfill the project is described by the following points: Literature and market research, Definition of the specifications and test cases, Selecting sensors and creation of hardware, Software development, and Testing the implementation and further improvements.

The Literature and Market research shows the current situation and approach in terms of rehabilitation digitalization and the different ways of using rehabilitation tests for patients. The definition of the specifications defines the tests used for the digital rehabilitation test and defines the measured quantities in each sub task. A definition of the used sensors and microcontrollers is crucial for the success of the project. The hardware for the wearables and balance board is defined. This design process includes the development of prototypes and a final wearable design. In order to detect the patient movements and calculate the test score, computer software is developed. The software has to be developed for the wearables and for the PC program. The final step includes the testing of the project and mentions further improvements. The project is part of a clinical study in a hospital.

1.5. Thesis Structure

The chapters of the thesis describe the following content:

Chapter 2 gives an overview over state of the art in medical tests and digital rehabilitation.
Chapter 3 describes the realization process of the project and lists the different observations and used methods.
Chapter 4 gives an overlook over the results and evaluates the findings.
Chapter 5 talks about the main results, and explains the interpretation.
Chapter 6 gives a summary over the key points of the thesis.

2. Literature, Basics

2.1. Assessment in Rehabilitation

In order to determine the physical status of a patient, it is important to get a test score. In that way, the physical functions of the patient can be determined. Another benefit is that the patients score can be compared over an extended time frame. In that way rehabilitation success can be tracked. The most common test procedures are covered in the following sections.

2.1.1. Berg Balance Test

The Berg Balance Test is used in the clinical environment to determine a patient's balance and especially the fall risk. The test consists of the following balance and functional mobility sub tasks: Sitting to standing, Standing unsupported, Sitting with back unsupported and feet supported, Standing to sitting, Transfers, Standing unsupported with eyes closed, Standing unsupported with feet together, Reaching forward with outstretched arm while standing, Picking up object from floor from a standing position, Turning to look behind over left and right shoulders while standing, Turn 360 degrees, Placing alternate foot on step while standing unsupported, Standing unsupported one foot on front, and Standing on one leg [1].

As it can be seen, the sub tasks involve everyday tasks and is aimed for elderly patients and individuals with mobility impairments. The individual sub tasks are all graded on a scale of 0 to 4. A score of 0 means the patient is not able to fulfill the task. A score of 4 means the patient can perform the task without any assistance. At the end of the assessment a final test score is calculated by summing up the sub task scores. The end score is interpreted as:

- 0 to 20: High fall risk
- 21 to 40: Moderate fall risk
- 41 to 56: Low fall risk

It is important to note, that some of the sub tasks are objective e.g. by time measurements. However, some of the sub tasks are highly subjective, as it involves the assessment of the rehabilitation worker. Therefore a standardized measurement behavior is not guaranteed. [1]

2.1.2. Knee Injury and Osteoarthritis Outcome Score

The Knee Injury and Osteoarthritis Outcome Score (KOOS) describes the quality of life for a patient with knee related symptoms. The principle behind this test is a self-reported questionnaire. The patients has to answer questions regarding pain, symptoms, function in daily living and sport, and quality of life. At the end of the test a score from 0 to 100 percent is received. The test helps to get a comprehensive knee assessment and offers a good overall evaluation. However, the honesty of the patient is important, as the test can be manipulated by wrong answers. Additionally the test is only question based, therefore there is no physical test aspect involved. [2]

2.1.3. MPPT

The MPPT is an important tool to measure the physical abilities of patients after surgeries, strokes, Parkinson's and neurological diseases, chronic pain, as well as movement and gait disorders. The different tasks of the MPPT can be seen in figure 2.1. However, the measurement with pen, paper and a stopwatch is the current approach to carry out this test. This results in a lot of limitations [3].

Modified Physical Performance Test
Scoring Sheet

Name:		Date:		
		Time (s)	**Scoring**	**Score**
1	Lift a book and put it on a shelf Book: PDR 1988 5.5 lbs Bed height 59 cm Shelf height 118 cm All sitting with feet on the floor		≤ 2 sec = 4 2.1. - 4 sec = 3 4.1 - 6 sec = 2 >6 sec = 1 unable = 0	Trial 1: Trial 2:
2	Put on and remove a jacket. 1. Standing 2. Use of bath robe, button down shirt, hospital gown		<10 sec = 4 10.1 -15 sec = 3 15.1 - 20 sec = 2 >20 sec = 1 unable = 0	Trial 1: Trial 2:
3	Pick up nickel from floor		≤ 2 sec = 4 2.1 - 4 sec = 3 4.1 - 6 sec = 2 >6 sec = 1 unable = 0	Trial 1: Trial 2:
4	50-foot walk test (3.28 feet/meter) 15.24 meters <15 sec = 3.33 feet/sec or 1.0m/sec		≤15 sec = 4 15.1- 20 sec = 3 20.1- 25 sec = 2 >25 sec = 1 unable = 0	Trial 1: Trial 2:
5	Climb one flight of stairs		≤5 sec = 4 5.1 - 10 sec = 3 10.1 - 15 sec = 2 >15 sec = 1 unable = 0	Trial 1: Trial 2:
6	Chair Rise (5x) without arms		<11 sec = 4 11.1-13.9 sec = 3 14-16.9 sec = 2 >17 sec = 1 unable = 0	Trial 1: Trial 2:
7	Climb 4 flights of stairs		Number of flights of stairs up and down (maximum of 4) Unable = 0	
8	Turn 360 degrees in direction of choice		Discontinuous steps = 0 Continuous steps = 2 Unsteady (grabs, staggers) = 0 Steady = 2	Discontinuous Continuous
9	Standing Balance in Full Tandem and Semi-Tandem	Full Tandem / Semi-tandem / Side-by-side 4 10s 10s 10s 3 3-9s 10s 10s 2 0-2s 10s 10s 1 Unable 0-9s 10s 0 Unable 0-9s 0-9s		Side by side: Semi-tandem: Full tandem:
	TOTAL SCORE maximum 36			

Figure 2.1.: A list of the composition of the Modified Physical Performance Test. The test in this form consists of 9 sub-tasks [4].

The first limitation is the inaccuracy in those manual tests. The evaluator must watch the performance of the patient very closely and measure the time too. This results in a very inaccurate measurement. It is also not guaranteed, that the evaluator judges the patient in a correct way every time the test is taken and human errors can occur. Another limitation is an inconsistency between evaluators. In this way no standardized way of measurement is guaranteed.
Also, the evaluator can only measure the patient's movements in a very subjective way. The patient's movement can seem very good overall, but individual movements of body parts are hard to judge. For a better detection of the joint position the patient often must change his clothing and wear hospital clothes.
Storing, comparing and analyzing the data is causing further inaccuracies. The data is measured by hand and then transferred to a spreadsheet. This is not ideal and can lead to missing or incorrect data. The measurement is not continuous and leads to a lack of data points and less frequent assessment.

2.2. Digital Assessment in Rehabilitation

There is a big focus on the digitalization of patient assessment and digitalization for training purposes. The following section focuses on these topics and list various examples.

The two biggest focuses of current digital patient assessment lay on gait tracking and balance tracking.
Examples for gait tracking are two different projects of MCI. Both projects focus on the analysis of gait while walking and/or walking stairs. They consist of a wearable device mounted to the feet in combination with an Android app. The assessment results in test scores with sub scores and further details, such as walked distance, number of steps and gait ratio.
Another example for patient assessment is a balance board designed at Yale University. The balance board can be seen in Figure 2.2. The force sensors in combination with an Arduino microcontroller are used in the balance board to determine the position of the patient on the floor. The board is designed to measure the center of gravity of the patient in a wide range. The assessment test consists of a static balance test and a dynamic balance mode. In this dynamic mode the patient's center of gravity has to follow a balance area on screen which moves dynamically. The test results in a percentage balance score.

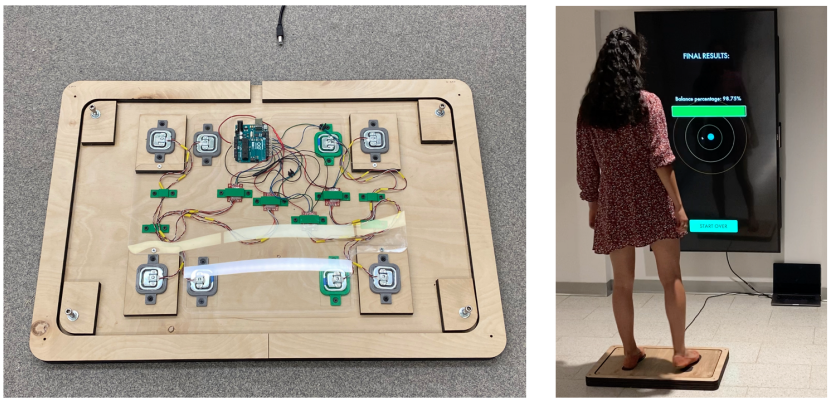

Figure 2.2.: The balance board at Yale university, including the board components and the assessment test procedure. Image permitted by R. Aslam, Yale University.

A completely different approach to digital patient assessment is shown in [5]. The robot developed in the project is used for the evaluation of patients. During the assessment patients carry out the timed up and go test. This test involves the patient standing up from seated position, walking three meters, turning around and walking back to the chair, and sitting down again. The robot evaluates the patient's lower limbs motion. Additionally the robot follows the patient during the test and is also interacting with the patient. At the end of the test the outcome of the evaluation is verified together with the healthcare worker.

In order to measure gait and track steps, the Inertial Measurement Unit (IMU) is a great tool. Especially the included gyroscope and the accelerometer offer the needed capabilities to measure the leg angle and leg acceleration. A typical gait cycle can be seen in figure 2.3.

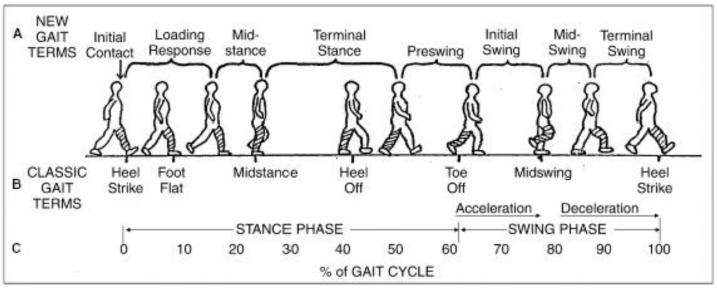

Figure 2.3.: An overview over the different components of the gate cycle [6].

As it can be seen in figure 2.3, the highlighted leg changes global position and also the relation to the rest of the body changes. This change reflects in a change of leg angle and a change of leg acceleration. A measurement of those two quantities can give an overview of the current status of the leg in the gait cycle. A possible example approach would be, if the leg angle reaches a certain value and at the same time acceleration in forward direction of the leg is detected, a step is initialized and the initial swing phase of the leg starts. If the angle of the leg goes to $0°$ and no acceleration is measured, the

loading response of the gait can be detected.

Detailed literature for an advanced control algorithm to evaluate stair running performance is shown in [7]. An option to measure foot motion tracking with the use of a convolutional neural network and a six axis IMU is presented in [8]. The potential to calculate an MPPT score automatically while wearing IMU sensors is proposed in [9]. The system uses machine learning for the score prediction. An approach for an exercise in a home based way is presented in [10]. It also focuses on protein intake and persuasive technology for older adults. The approach of a home based High-Intensity Interval Training for Parkinson's patients is presented in [11]. The project aims to be feasible and safe and proves this claims by data collection. An idea for low appendicular lean mass detection in older adults is explained by [12]. The method is called bio-electrical impedance analysis. The validity of low appendicular lean mass detection by using this method is not the best. However the process is able to detect low muscle mass in adults.

The use of technology and sensors in physical training is of big importance. It helps to store, compare and standardize results and makes improvements visible. This approach is used by Yale university in various projects and products.
One of those products is the Bulldog RepBox, which can be seen in a training environment in figure 2.4 [13]. The product is used to count repetitions while doing different kinds of workouts such as sit ups, push ups, squats, etc. A sensor in the product can measure the distance between the person and the RepBox and therefore count repetitions. The distance between sensor and person can be calibrated, in order to match the type of exercise.

Figure 2.4.: The Yale Bulldog RepBox used in a training session. [13]

Another project at Yale University which uses sensors in physical training is the Digital Dot Drill Assessment. This project uses a traditional dot drill mat equipped with force sensors, which can be seen in figure 2.5.

Figure 2.5.: The Yale Digital Dot Drill Assessment mat used for training sessions.

The Dot Drill Assessment is usually used for training sessions with football or basketball players. The player has to jump different types of patterns on the mat. The time to do this exercise is taken by the coach. However, the player performs the exercise which such a high speed that makes an evaluation of the foot position difficult. The digital version helps to detect if the player hits the dots and also takes the time automatically.

3. System

3.1. Hardware

A microcontroller is an integrated circuit which is built in a compact form factor. It usually consists of a processor, inputs/outputs, a memory chip and a clock generator. It is used to control processes or devices in different applications and perform defined tasks. Microcontrollers are used in various embedded systems and control engineering tasks and other fields such as: Automotive, Industrial Automation, Consumer Electronics, Internet of Things (IoT), Aerospace, and Medical Devices.

Microcontrollers are programmed either in general programming languages such as C and C++ or tools as e.g. Arduino IDE. There are different controllers available such as Arduino, Raspberry Pi, AVR or ESP32. Choosing the right microcontroller is very specific to the project. It mostly depends on the available peripherals, power consumption, cost and tools for development. Depending on the project it could be of big interest to choose a microcontroller with Bluetooth and/or WIFI support. Otherwise, wireless communication with e.g. a PC is not easily possible without additional hardware or shields.

A good overall understanding of the IMU hardware can be gained in [14]. To detect the angular rate and orientation of a body, the IMU uses the sensor values of accelerometers, gyroscopes, and sometimes magnetometers. They all deliver sensor values on their own, but the real strength of the IMU lies in the combination of those sensor values. When combining the three sensor values, e.g. to measure the angle with a complementary filter or a Kalman filter, the accuracy of the measured angle is increased [14][15]. The most important measured values of the IMU are the angular velocity and the acceleration. The values can be measured directly by the IMU components. However, they need a filter as they are not very accurate. The measurement of linear velocity is also possible, because the integral of linear acceleration over time is the change in velocity. However, this may already display inaccuracy, because of additional errors and integration of noise.
IMUs are used to maneuver vehicles such as motorcycles, aircraft and spacecraft. In recent years IMUs are also used in consumer electronics such as smartphones and fitness trackers.

Force sensors are used to measure the force applied to the sensor surface area. The working principle is the conversion of physical force into electrical quantities, which can

be measured by e.g. the inputs of a micro controller. The most common type of force sensor is the strain gauge load cell. It consists of thin conductive stripes, also known as strain gauges. If force is applied to this type of sensor, they start to bend. This bend off the strain gauges results in a change of electrical resistance. This change can be used to detect the applied force.
Force sensors are used in a variety of different fields such as: Medical Devices, Consumer Electronics, Aerospace, Robotics, Material Science, and Automotive.

Important factors for choosing a specific force sensor are the linearity, temperature stability, sensitivity and range of measurement. It is also important to calibrate the force sensor to guarantee an accurate measurement result.

3D printing describes the creation of a three dimensional object out of various materials. The used materials are plastic, resin, metal, powder, or carbon fiber. The object is created from a digital model and printed layer by layer. A slicing software is used in order to convert the digital 3D model, usually created in a Computer Aided Design (CAD) program, into G-code which the 3D printer can read. There are different versions of 3D printers. However, Delta and Cartesian printers are the most popular ones.
The material and printer decision is highly dependent on the use case. Different use cases of 3D printers include prototyping, product development and education. With the help of 3D printers objects can be created in a rapid way and highly customized way.

3.2. Digital MPPT Adaptation

Adaptations of the traditional MPPT have been made, in order to define the most important test, cover all movements needed in everyday life, and include additional sensor data. This helps to get a more objective and detailed score compared to the traditional analog MPPT. The sub tests chosen and adapted from the traditional MPPT are the following: 10 step walk, 4 step stair walk, Turn 360 degree, Chair Rise (5x) without arms, Pick up object while sitting, Pick up object while standing, Balance with feet together, and Balance while in tandem position. In comparison to the traditional MPPT, some tasks were removed or modified. The $15.24\,\mathrm{m}$ walking test was changed to 10 steps, as this is about half the distance of the traditional MPPT distance and more feasible for the patients. The steps are also easier to evaluate for the rehabilitation workers. Lifting up a book and putting it on a shelf and picking up a nickel from the floor was changed to picking up the sensor object while sitting/standing, to get sensor data via the object. Climbing stairs is especially hard, so one flight of stairs was changed to 4 steps, as the 4 steps are already part of the clinic rehabilitation at Yale hospital. Therefore the climb of 4 flights of stairs was removed, after feedback of the Yale rehabilitation workers. The put on and remove a jacket sub-task of the traditional MPPT was removed, as this task can not be measured in a reliable way by the IMU sensors.

Those tests cover the most important body movement groups and a variety of different movements of everyday life. The tests and the different metrics are chosen together

with rehabilitation workers of the Yale university hospital. Out of the eight tests, the first six use the IMU wearable and the last two use the new balance board adaption. The IMU wearable position of each task will be covered in 3.3.

3.3. MPPT Sub-Task Definitions

This section has a focus on the software of the digital MPPT sub tasks and also describes characteristics used in general. One main feature of the software, which is written in Python, is the graphical user interface. The GUI can be seen in figure 3.1. Figure 3.1 shows detailed information of the sub task results and also the result of the total test. The GUI features a calibration button, which is pressed once the patient is ready to perform the test and the sensors are placed on the body. This results in a definition of the current wearable position as the position zero. A colored field next to the sub task name shows the status of the current sub task. A red field shows if a calibration is needed, a yellow light shows if a test is in progress, and a green light shows that a test is finished.

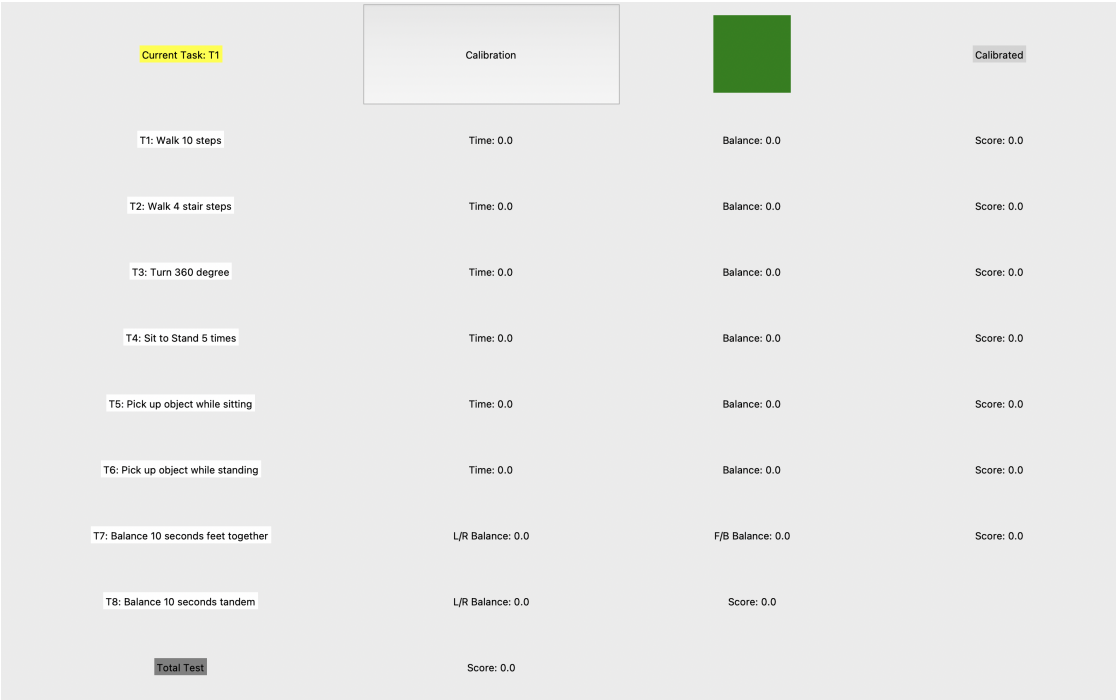

Figure 3.1.: The graphical user interface of the digital MPPT.

Just as the software for the wearable, the PC software also features a Message Queuing Telemetry Transport (MQTT) element. In the case of the PC software it is a MQTT receiver, which receives the wearable string from the MQTT broker. The received string is split into the individual variables and used in the program code. The variables are additionally logged into a text file and saved to the PC hard drive.

A big factor of the software is the conversion of the quaternions into Euler angles. It is important to note that the original coordinate system of the IMU in the wearable is based on the earth magnetic north pole. This means that the z-axis of the wearable seen in figure 3.2 matches with the gravitational force direction of the earth. If the wearable is tilted around a axis and mounted on the body, this rotation has to be taken into consideration. The quaternion rotation around the x-axis is characterized by equation 3.1, equation 3.2, equation 3.3, and equation 3.4. The quaternion rotation around the y-axis is characterized by equation 3.5, equation 3.6, equation 3.7, and equation 3.8. The quaternion rotation around the z-axis is characterized by equation 3.9, equation 3.10, equation 3.11, and equation 3.12.

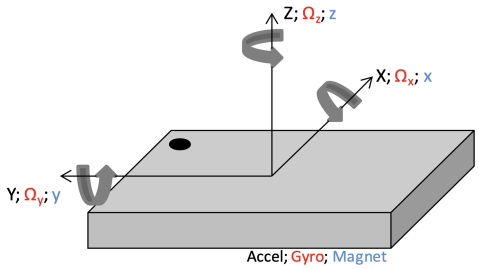

Figure 3.2.: The axes description of the BNO055.[16]

$$q_0 = \frac{\sqrt{2}}{2} \cdot q_{0prime} - \frac{\sqrt{2}}{2} \cdot q_{1prime} \tag{3.1}$$

$$q_1 = \frac{\sqrt{2}}{2} \cdot q_{1prime} + \frac{\sqrt{2}}{2} \cdot q_{0prime} \tag{3.2}$$

$$q_2 = \frac{\sqrt{2}}{2} \cdot q_{2prime} + \frac{\sqrt{2}}{2} \cdot q_{3prime} \tag{3.3}$$

$$q_3 = \frac{\sqrt{2}}{2} \cdot q_{3prime} - \frac{\sqrt{2}}{2} \cdot q_{2prime} \tag{3.4}$$

$$q_0 = \frac{\sqrt{2}}{2} \cdot q_{0prime} - \frac{\sqrt{2}}{2} \cdot q_{2prime} \tag{3.5}$$

$$q_1 = \frac{\sqrt{2}}{2} \cdot q_{1prime} - \frac{\sqrt{2}}{2} \cdot q_{3prime} \tag{3.6}$$

$$q_2 = \frac{\sqrt{2}}{2} \cdot q_{2prime} + \frac{\sqrt{2}}{2} \cdot q_{0prime} \tag{3.7}$$

$$q_3 = \frac{\sqrt{2}}{2} \cdot q_{3prime} + \frac{\sqrt{2}}{2} \cdot q_{1prime} \tag{3.8}$$

$$q_0 = \frac{\sqrt{2}}{2} \cdot q_{0prime} - \frac{\sqrt{2}}{2} \cdot q_{3prime} \tag{3.9}$$

$$q_1 = \frac{\sqrt{2}}{2} \cdot q_{1prime} + \frac{\sqrt{2}}{2} \cdot q_{2prime} \tag{3.10}$$

$$q_2 = \frac{\sqrt{2}}{2} \cdot q_{2prime} - \frac{\sqrt{2}}{2} \cdot q_{1prime} \tag{3.11}$$

$$q_3 = \frac{\sqrt{2}}{2} \cdot q_{3prime} + \frac{\sqrt{2}}{2} \cdot q_{0prime} \tag{3.12}$$

After the rotation the conversion from quaternions to Euler angles is needed. This is described by equation 3.13, equation 3.14, and equation 3.15.

$$\alpha = \frac{360}{2\pi} \cdot atan2(2 \cdot (q0 \cdot q1 + q2 \cdot q3), 1 - 2 \cdot (q1 \cdot q1 + q2 \cdot q2)) \tag{3.13}$$

$$\beta = \frac{360}{2\pi} \cdot asin(2 \cdot (q0 \cdot q2 + q3 \cdot q1)) \tag{3.14}$$

$$\gamma = \frac{360}{2\pi} \cdot atan2(2 \cdot (q0 \cdot q3 + q1 \cdot q2), 1 - 2 \cdot (q2 \cdot q2 + q3 \cdot q3)) \tag{3.15}$$

The sub task scoring system will take the traditional MPPT timescore into consideration and also a newly measured sensor parameter, also defined as balancescore. The sub scores are calculated as described by equation 3.16. The equation was defined together with the rehabilitation workers at Yale hospital, to still keep a bigger focus on the timing aspect of the MPPT and make smaller adjustments with the balancescore. Therefor the weight of the timescore was set to $75\,\%$, and the weight of the balancescore was set to $25\,\%$. This equation is true for all sub tasks except for the balance sub task, as this sub task is not time dependent. All scores are in percent, therefore 100 is the best value and 0 the worst value. At the end the mean of all sub tasks is calculated and results in a final test score.

$$score = timescore \cdot 0.75 + balancescore \cdot 0.25 \tag{3.16}$$

3.3.1. 10 Step Walk

For this sub-task, wearable 1 is placed on the outside of the lower left leg, like it can be seen in figure 3.3.

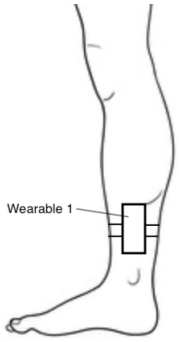

Figure 3.3.: The position of wearable 1 on the outside of the left leg.

Wearable 2 is placed on the outside of the lower right leg. The wearables are mounted with the velcro strap so that the wearable switch faces the same direction the patient walks. The logic of the step detection is described by figure 3.4. The units for the values in this flowchart and the following flowcharts are $^\circ$ for the leg angles and m/s^2 for the accelerations. The patient has to walk ten steps and the passed time is the total time needed. The timer starts automatically as the patient takes the first step. A timescore is generated which can be seen in table 3.1. Additionally timers are also started as soon as a feet leaves the ground. This moment can be detected if a leg angle $LegAngle$ bigger 10° and a leg acceleration $LegAcc$ bigger $2.5\,\mathrm{m/s^2}$ is detected at the same time. This corresponds to the gait behavior in figure 2.3. The time for right foot and left foot is calculated and the ratio between the times. This results in a balance score. The ratio percent calculation is defined by equation 3.17. If the balance score is under $25\,\%$ the patient gets a balance score of $25\,\%$, as the person is still able to perform the task.

Table 3.1.: The criteria for the timescore of sub task 1.

t_{total} / s	$timescore$ / %
< 8	100
> 8 - 10.5	75
> 10.5 - 13	50
> 13	25
-	0

$$balancescore = 100 - \left| \left(\frac{t_{total} - t_{left}}{t_{total} - t_{right}} - 1 \right) \cdot 100 \right| \tag{3.17}$$

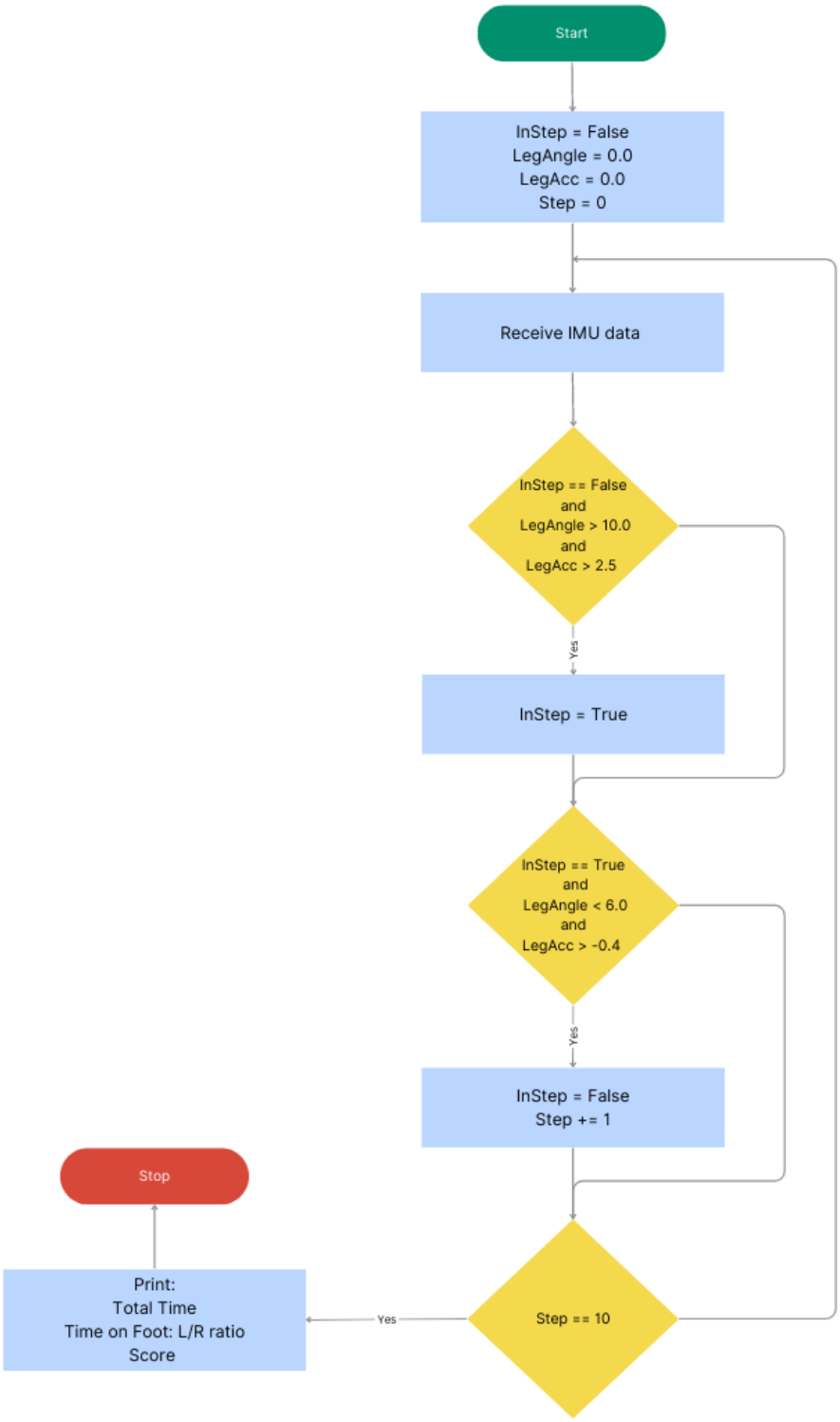

Figure 3.4.: The flowchart for the logic of sub task 1.

3.3.2. 4 Step Stair Walk

For this sub-task, wearable 1 is placed on the outside of the lower left leg. Wearable 2 is placed on the outside of the lower right leg. The wearables are mounted with the velcro strap so that the wearable switch faces the same direction the patient walks. The logic of the step detection is described by figure 3.5. The patient has to walk four steps on a stair and the passed time is the total time needed. The timer starts automatically as the patient takes the first step. The step start moment can be detected if a leg angle $LegAngle$ smaller $-10\,^\circ$ and a leg acceleration $LegAcc$ bigger $1.5\,\mathrm{m/s^2}$ is measured at the same time. A timescore is generated which can be seen in table 3.2. Additionally timers are also started as soon as a feet leaves the ground. This corresponds to the gait behavior in figure 2.3. The time for right foot and left foot is calculated and the ratio between the times. The ratio percent calculation is defined by equation 3.17. If the balance score is under $25\,\%$ the patient gets a balance score of $25\,\%$, as the person is still able to perform the task.

Table 3.2.: The criteria for the timescore of sub task 2.

t_{total} / s	$timescore$ / %
< 5	100
> 5 - 10	75
> 10 - 10	50
> 15	25
-	0

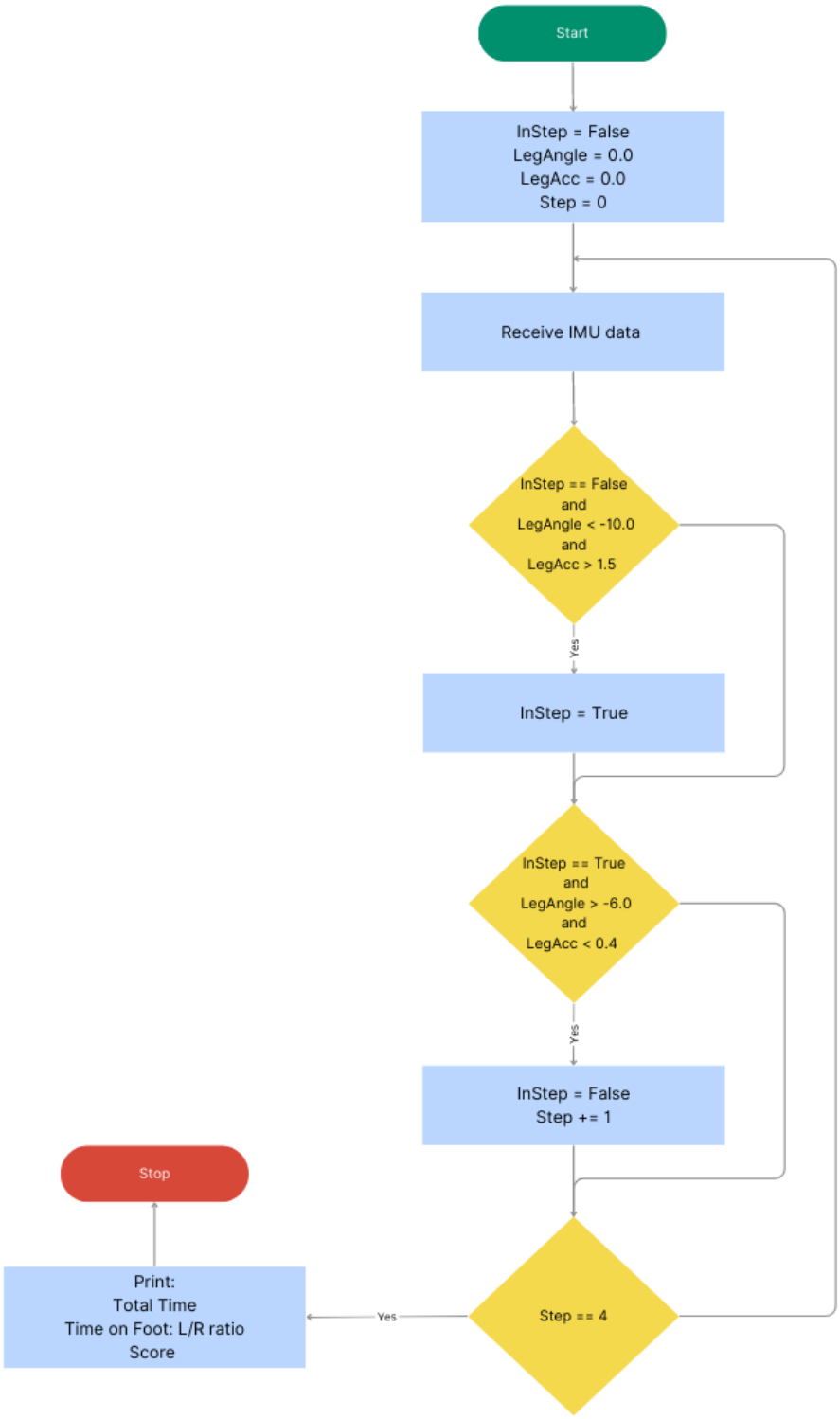

Figure 3.5.: The flowchart for the logic of sub task 2.

3.3.3. Turn 360 Degree

For this task Wearable 3 is placed around the chest. The wearable is mounted with the velcro strap so that the wearable switch faces downside and therefore faces the floor. The logic of the turn detection is described by figure 3.6. The patient has to turn around 360 degree and keep as stable as possible. This test involves attentiveness of the patient. Therefore the timer starts as the calibrate button in the graphical user interface of the test is pressed. A timescore is generated which can be seen in table 3.3. The balancescore is effected by the chest angles and the patient has to stand as straight as possible while turning around. Both chest angles, maximum leaning forward/backward angle $\alpha_{chestmax}$ and maximum leaning left/right angle $\beta_{chestmax}$, are summed up and the average is calculated. This is described by equation 3.18. The balancescore determination can be seen in table 3.4.

Table 3.3.: The criteria for the timescore of sub task 3.

t_{total} / s	$timescore$ / %
< 3.5	100
> 3.5 - 5.5	75
> 5.5 - 7.5	50
> 7.5	25
-	0

$$angle_{chest} = \frac{\alpha_{chestmax} + \beta_{chestmax}}{2} \tag{3.18}$$

Table 3.4.: The criteria for the balancescore of sub task 3.

$angle_{chest}$ / °	$balancescore$ / %
< 20	100
> 20 - 30	75
> 30 - 40	50
> 40	25

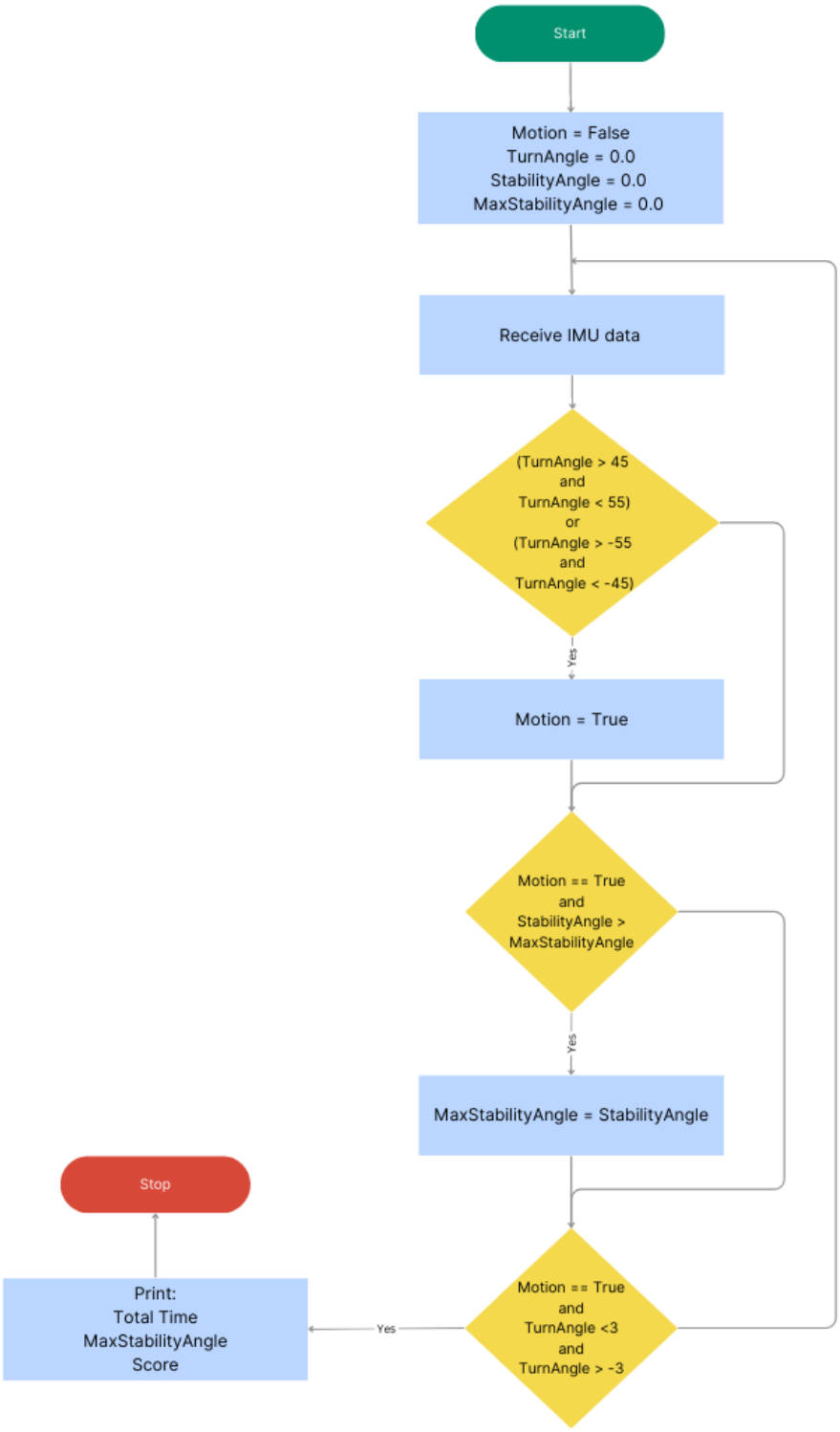

Figure 3.6.: The flowchart for the logic of sub task 3.

3.3.4. Chair Rise (5x) Without Arms

For this task wearable 1 is placed on the front of one upper leg and wearable 3 is placed around the chest. Wearable 1 is mounted with the velcro strap so that the wearable switch faces to the left side. Wearable 3 is mounted with the velcro strap so that the wearable switch faces downside and therefore faces the floor. The logic of the rise detection is described by figure 3.7. The patient has to go from sitting to standing position five times without the use of the arms. The height of the chair is $50.8\,\mathrm{cm}$. The patient should keep his upper body as stable as possible and do not lean to the left or right. The front/back chest angle is not used as a criteria for this test, as leaning forward while getting up is a natural human behavior. The timer starts as soon as the first stand up is recognized. The get up start moment can be detected if a leg angle $LegAngle$ bigger $40\,^\circ$ is measured. A timescore is generated which can be seen in table 3.5. The balancescore is effected by the left/right chest angle and the patient has to get up as straight as possible. The balancescore determination can be seen in table 3.6.

Table 3.5.: The criteria for the timescore of sub task 4.

t_{total} / s	$timescore$ / %
< 11	100
> 11 - 14	75
> 14 - 17	50
> 17	25
-	0

Table 3.6.: The criteria for the balancescore of sub task 4.

α_{chest} / °	$balancescore$ / %
< 20	100
> 20 - 30	75
> 30 - 40	50
> 40	25

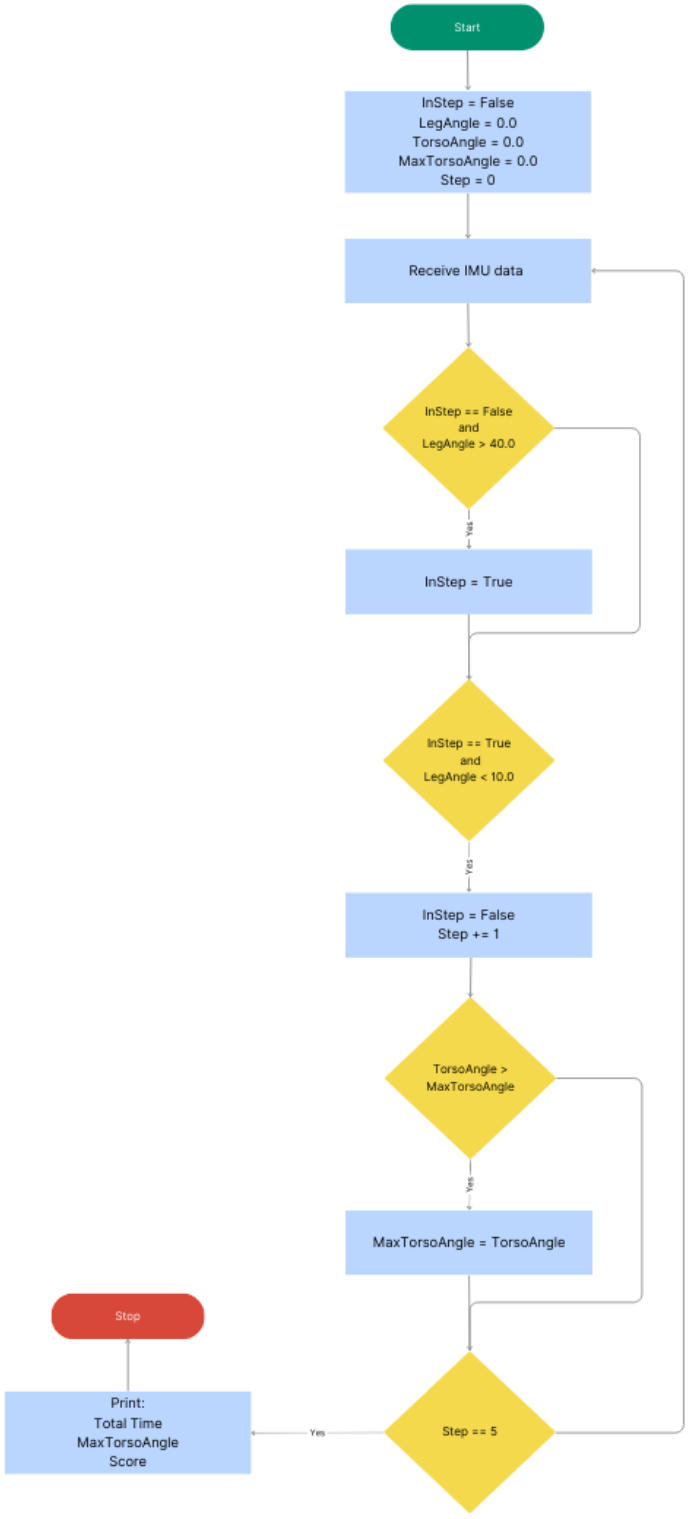

Figure 3.7.: The flowchart for the logic of sub task 4.

3.3.5. Pick Up Object While Sitting/Standing

The patient has to pick up a smart object of the floor and place it on a table while sitting and while standing. This results in two different sub tasks with different score criteria. The smart object (wearable 4) is identical to the wearable mounted to the body, but without a velcro strap. For the sitting part of the test, the height of the chair is $50.8\,\mathrm{cm}$. The table height is $71.76\,\mathrm{cm}$. The logic of the pick up detection is described by figure 3.8. The timer starts as soon as a movement of the object is detected. This movement of the object is detected, if either of the acceleration values x_{Acc}, or y_{Acc}, or z_{Acc} is bigger $0.5\,\mathrm{m/s^2}$. A timescore is generated which can be seen in table 3.7. The balancescore is effected by the hand movement of the patient while picking up the object, therefore the mean acceleration $MeanAcc$ is the measured quantity. In that way body tremor can be detected. The logic behind the mean acceleration of the smart object to determine the body tremor can be seen in figure 3.8. To calculate the $MeanAcc$, the $TotalAcc$ is calculated by summing up $xAcc$ and $yAcc$ in every code run. $zAcc$ is not summed up, as this is the main axis of movement, to put the object from the floor to the table. A $Count$ variable gets increased by two in every code cycle, as the two acceleration values are added every cycle. The balancescore determination can be seen in table 3.8. If the acceleration values x_{Acc}, and y_{Acc} are smaller $0.15\,\mathrm{m/s^2}$ and x_{Acc} is smaller $0.20\,\mathrm{m/s^2}$ at the same time, a stop of the smart object on the table is detected and the timer stops.

or z_{Acc} is bigger $0.5\,\mathrm{m/s^2}$

Table 3.7.: The criteria for the timescore of sub task 5 and sub task 6.

t_{total} / s	$timescore$ / %
< 2	100
> 2 - 4	75
> 4 - 6	50
> 6	25
-	0

Table 3.8.: The criteria for the balancescore of sub task 5 and sub task 6.

$MeanAcc$ / m/s^2	$balancescore$ / %
< 2	100
> 2 - 2.5	75
> 2.5 - 3	50
> 3	25

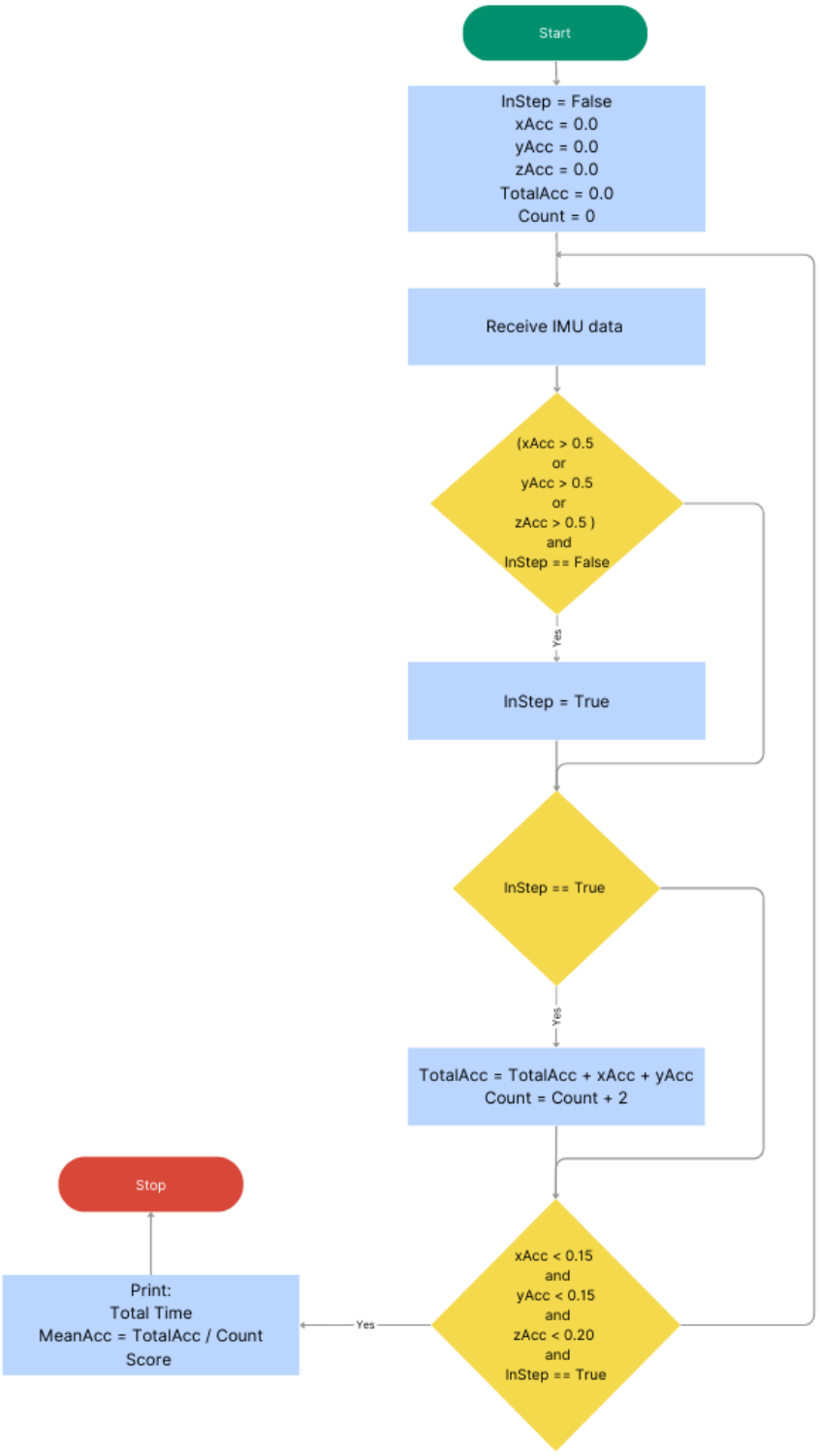

Figure 3.8.: The flowchart for the logic of sub task 5 and sub task 6.

3.3.6. Balance With Feet Together/While in Tandem Position

The patient has to stand on the balance board and stand still for ten seconds. This is done first with the feet together and afterwards in tandem position. The logic of the balance board and mean value calculation for left/right balance and front/back balance can be seen in figure 3.9. The values $S1$, $S2$, $S3$, and $S4$ stand for the four force sensor values. $S1$ is the value of the left foot front force sensor. $S2$ is the value of the left foot back force sensor. $S3$ is the value of the right foot front force sensor. $S4$ is the value of the right foot back force sensor. In order to calculate the mean left/right ratio $StabilityLR$, and the mean front/back ratio $StabilityFB$, the sensor ratios are summed up and divided by the count ov variables. This summed up sensor ratios are $TotalLR$ and $TotalFB$. The final balance score for balance with feet together is described by equation 3.19. The final balance score for balance while in tandem position is described by equation 3.20. If the balance score is under $25\,\%$ the patient gets a balance score of $25\,\%$, as the person is still able to perform the task.

$$balancescore = 100 - \left| \left(\frac{StabilityLR + StabilityFB}{2} - 1 \right) \cdot 100 \right| \tag{3.19}$$

$$balancescore = 100 - \left| \left(StabilityLR - 1 \right) \cdot 100 \right| \tag{3.20}$$

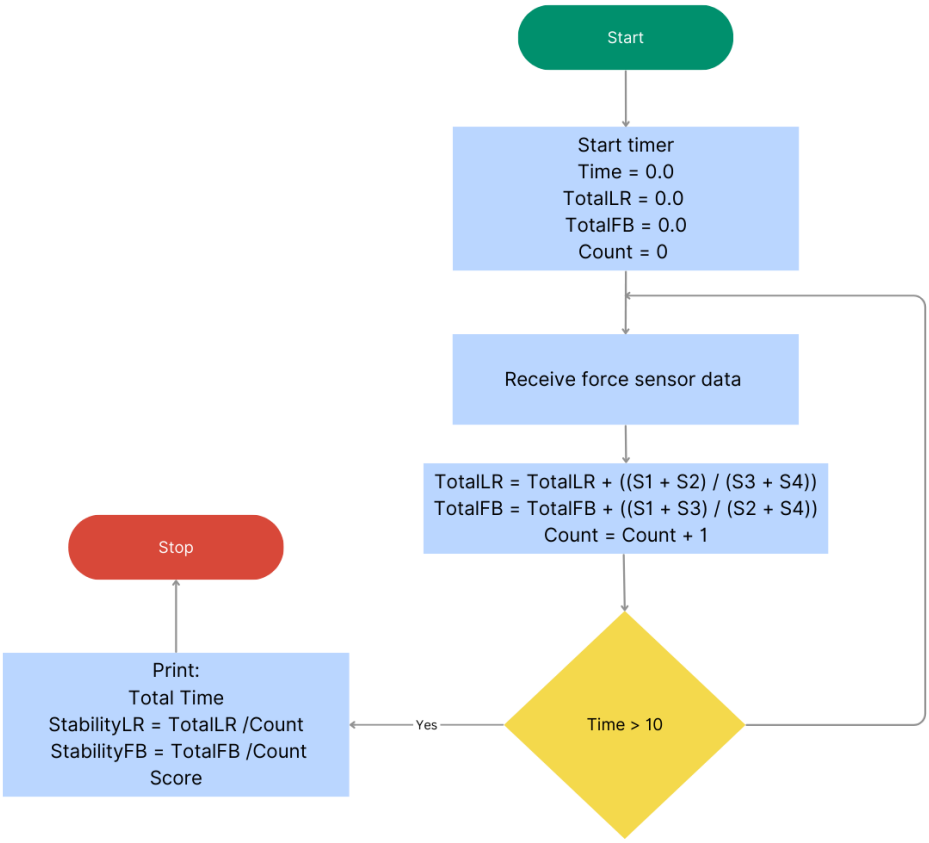

Figure 3.9.: The flowchart for the logic of sub task 7 and sub task 8.

3.4. Wearable Software

The wearable software is used in the IMU version of the wearable device and the balance board. The main three parts of the software are the following: MQTT sender, Receive sensor data, and IMU Calibration.

The first two sections are present in the IMU version of the wearable device and the balance board version. The IMU Calibration including the LED blink is only present in the IMU version of the wearable.
The MQTT sender part of the code uses the WIFI capabilities of the microcontroller(ESP32, Espressif Systems) to connect to a WIFI network and communicate via an MQTT broker, e.g. a mosquitto server. The WIFI name, WIFI password and server ID have to be defined. A local server was used in the beginning. Later a web server was used, so that the program can also run on PCs of clinical workers, without installing additional software such as Eclipse Mosquitto on their PC. Five topics are defined, one topic for each one of the microcontrollers. The ESP32 tries to connect to the server. If a connection is established, the board begins to read the sensor values. The received values are stored in a string and published to the MQTT server. The string consists of

values, each separated by a comma. The values of the IMU version of the wearable are the following: Quaternion w component, Quaternion x component, Quaternion y component, Quaternion z component, Acceleration x direction, Acceleration y direction, and Acceleration z direction.

The string values of the balance board software version are the following: Force Sensor 1, Force Sensor 2, Force Sensor 3, and Force Sensor 4.

The main quantity measured for the IMU are the quaternions and the linear acceleration for each axis, which can be obtained using the $Adafruit\_Sensor.h$ and $Adafruit\_BNO055.h$ libraries provided by Adafruit. The $getQuat()$ command provides quaternions for the rotation axis. The use of the Adafruit command $Adafruit_BNO055 :: VECTOR_LINEARACCEL$ is of big importance, as this delivers the acceleration and removes the influence of the gravitation.
The force sensor values in the balance board adaptation are read by a reading of four analog values.

The final part of the code in the IMU adaptation is important to check the calibration status of the accelerometer, gyroscope, and magnetometer to ensure accurate measurements [17]. The Adafruit function $getCalibration$ delivers the calibration status of the overall system, accelerometer, gyroscope, and magnetometer. The ESP32 features an internal LED, which is used to display the calibration status. If full calibration is achieved, the internal LED starts to blink. Before achieving full calibration status, the LED is switched off.

3.5. Balance Game

The balance game acts as a training mode of the project, which helps to train the lower body muscles and balance. The patient is motivated while exercising and practicing in a game. The patient is able to control a character in a virtual environment, that is programmed with the help of the UNITY game engine. This virtual environment can be seen in figure 3.10.

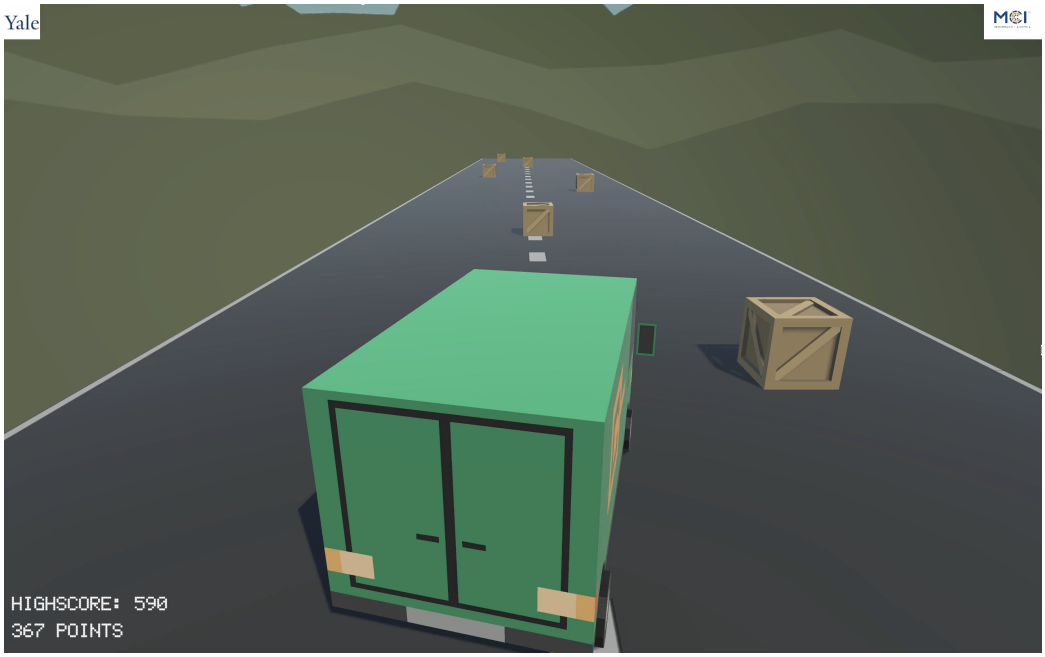

Figure 3.10.: The interface of the balance game.

In order to not feel like traditional, repetitive recovery exercise, the game offers a game character and the input of the character is provided by the balance of the user. The game features objects, which the user has to avoid. The character moves forward in an automatic way and the user balance input controls move the character to the left or right side of the screen. The input device features the same software and hardware as for the balance assessment of the digital MPPT part of the project. The UNITY software consists of two main parts. There is an MQTT handler, which receives messages send by the balance board hardware. The second part of the software is the game itself, which handles the score calculation and the control of the character. The score is determined by the time spent without touching an object. One frame in the game without touching an object equals in one high score point. The left/right input is determined by the L/R feet balance ratio. If the ratio is greater than 1, the character moves to the left. If the ratio is smaller than 1, the character moves to the right. The character movement behaves proportional to the L/R feet balance ratio. The high score is visible in the game screen. The lighting and collision detection is provided by the UNITY game engine and the integrated object handler. The game also features a game over screen and a start menu. The game over screen appears as the user leaves the road.

3.6. Wearable Design

The wearable, which is used to track the limb and body movement, consists of the following main components: microcontroller (ESP32, Espressif Systems), IMU (BNO055, BOSCH), battery (LP502030 $250\,\mathrm{mAh}$, EEMB), electrical switch (KCD1-101, DaierTek), wires, thermoplastic polyurethane (TPU) case, velcro strap, screws and nuts.

The exact version of the ESP32 used is the Adafruit HUZZAH32 - ESP32 Feather. This version of the ESP32 offers a JST connector, which is used to connect LiPoly batteries. The connection of ESP32 to BNO055 IMU can be seen in figure 3.11. The used battery offers a capacity of $250\,\mathrm{mAh}$, $3.7\,\mathrm{V}$, and $0.9\,\mathrm{Wh}$.
For the case of the wearable, 3D print technique is used. The chosen material is TPU. The TPU offers a stable design, which is also able to absorb light impacts. This is especially helpful if a patient collides with an object while wearing the wearable. A lot testing was done to find the ideal infill pattern and infill percentage for the 3D print. The ideal infill pattern was found to be "Cross" with an infill percentage of $20\,\%$.

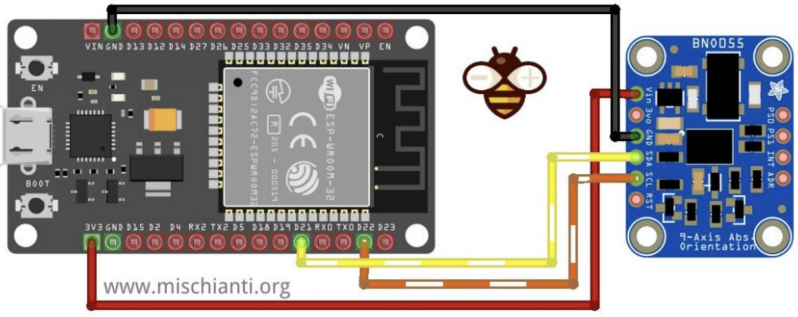

Figure 3.11.: Connection of the ESP32 to the BNO055.[18]

The infill pattern and infill percentage are used for the wearable design. The design can be seen in figure 3.12. The design features a screw on lid and holes in the base plate for the installation of ESP32 and BNO055 via screws and bolts. There are slots for the installation of a velcro strap in horizontal and vertical direction and cutouts for the charging port and electrical switch.

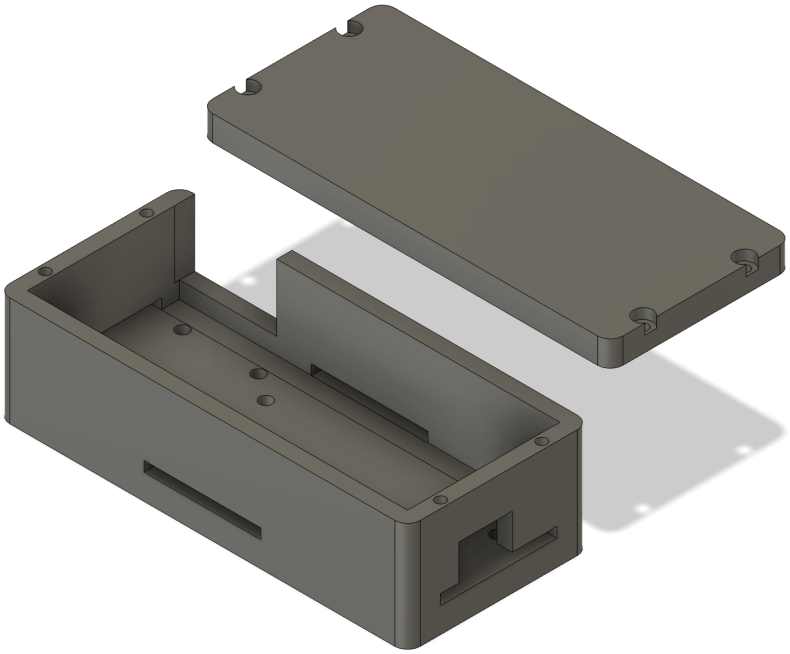

Figure 3.12.: The design of the IMU Wearable. The dimensions are $81\,\mathrm{mm} \cdot 36\,\mathrm{mm} \cdot 26.5\,\mathrm{mm}$.

The installation of the components in the wearable case can be seen in figure 3.13.

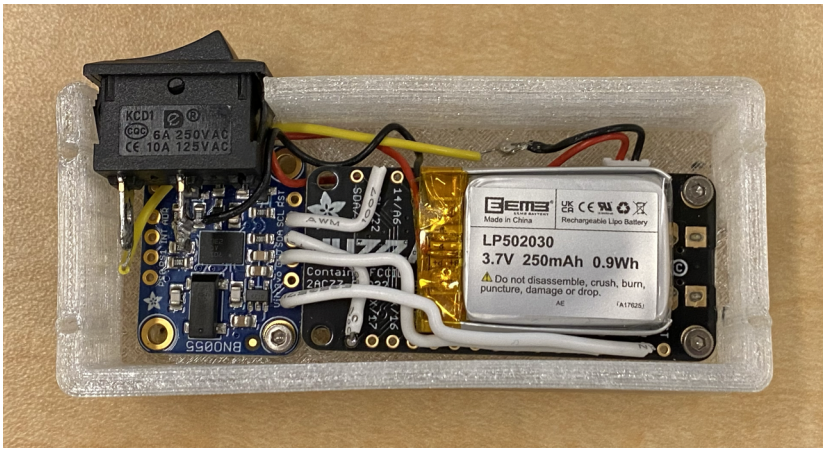

Figure 3.13.: The wiring of the wearable inside the case. The electrical switch, IMU, ESP32, and battery can be seen from top left to bottom right.

3.7. Balance Board Design

The balance board adaption features a shoe sole design for each individual foot. The board is used to measure patient balance and consists of the following main components: microcontroller (ESP32, Espressif Systems), force sensors (FSR 406, Interlink Electronics), resistors ($100\,\Omega$), external battery ($10\,000\,\mathrm{mAh}$, EasyAcc), wires, TPU case, TPU soles, screws and nuts.

In order to power the balance board the external battery is connected via the micro USB port of the microcontroller. The wire connection of ESP32 to the four force sensors and resistors can be seen in figure 3.14.
The sole design can be seen in figure 3.15. The sole design features eight individual pieces. Four pieces for the right sole and four pieces for the left sole. The reason for the four pieces per side is the sandwich design of the TPU sole. Two sensors are placed in the bottom parts of each side and the top parts are placed on them and fixed via plug connectors. The used infill pattern and infill percentage is identical to section 3.6. The CAD design features groves for the cables and sensor connectors, four groves for placement of the force sensors, a split/puzzle design, and plugs for clipping the sensor halves together. The sensors are fixed inside the TPU material by an adhesive film on the backside of the sensors. Balance tracking is possible by the placement of two sensors per foot in the sole. One sensor is placed at the front part of the sole, the other sensor is placed in the rear part of the sole. Wires are attached to the sensor connectors via soldering. The puzzle design of the sole is needed, in order to fit the sole on the bed of the used 3D printer. The Ender 3 printer features a square printing surface of $220\,\mathrm{mm}$. The length of the sole is $300\,\mathrm{mm}$. Therefore a print with the puzzle design approach is possible on the Ender 3. After the sensors are placed in the bottom sides of the sole, the top part can be fixed via the plug connectors.

The soles are placed under the shoes of the patients and the soles have to be centered under the shoes. This is done while the patient is sitting. The reasons to put the soles under the shoes and not in them, are because of time, hygienic reasons, and because the patient can also do the test if the soles are too large. It would not be possible to put too large soles inside the shoe, but it is possible to put them under the shoes and center them.
The rest of the electrical hardware is soldered onto a Electrocookie Solderable PCB Board and placed in a box with a lid. The design for the box can be seen in figure 3.16.

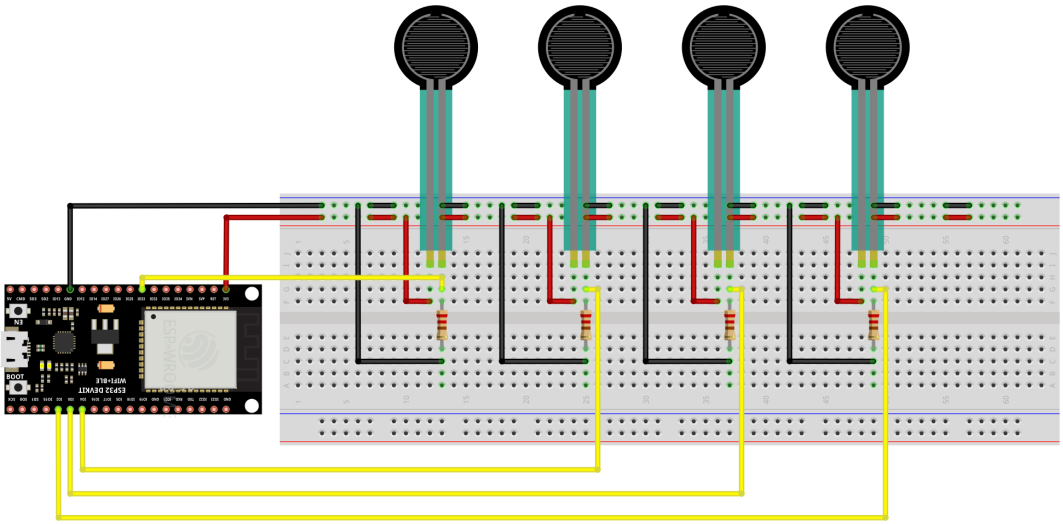

Figure 3.14.: Connection of the ESP32 to the force sensors and resistors.

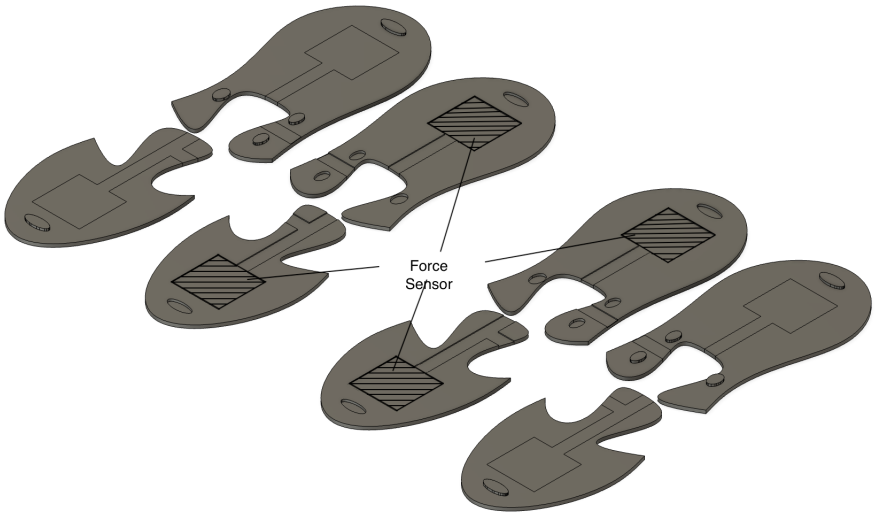

Figure 3.15.: The CAD design of the sole used to measure balance.

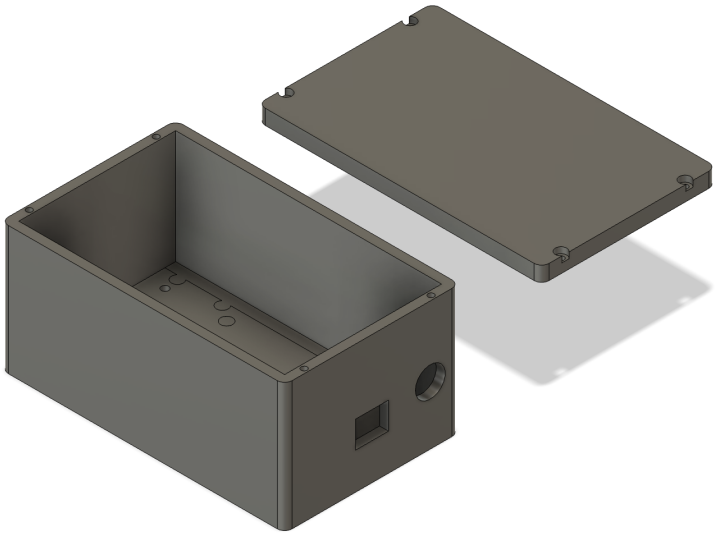

Figure 3.16.: The CAD design of the sole box used to store the electrical components. The electrical components are a microcontroller and resistors, connected by wires on a PCB Board.

4. Evaluation

4.1. System Evaluation

In order to evaluate the system different tasks are carried out and plots of the measured quantities are created. Those plots contain the measured values of the wearables and the balance board.

4.1.1. Setup

The setup for the evaluation of the force sensors consists of different weights placed on the force sensor, while the sensor is inside the TPU sole. The weights are stacked on the sensor surface by using a CAD created object, which can be seen in figure 4.1. In order to place the 3D printed object precisely on the sensor surface area, an angle tool is used, which is also created via 3D print out of Polylactic Acid (PLA).

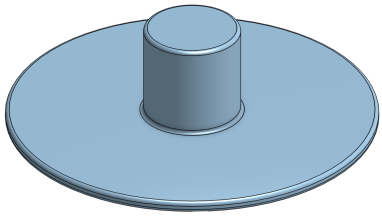

Figure 4.1.: The weight mount for the test of the force sensor. Weights are placed around the cylinder on top. The bottom features a $40\,\mathrm{mm} \cdot 40\,\mathrm{mm}$ square contact surface to place the FSR underneath.

In order to evaluate the angle values of the BNO055, the value of the IMU is compared to an encoder. The experimental setup for the comparison can be seen in figure 4.2. The wearable is attached around the outside of the forearm. The encoder shaft is attached to the outside of the forearm by using a velcro strap and a 3D printed plate attached to the shaft. A rotation around the encoder shaft is performed while the encoder is attached to a stable surface. The type of optical rotary encoder is "LPD3806-600BM-G5-24C" and it offers 600 values per resolution. The encoder is connected to two digital inputs of the ESP32 and the BNO055 is also connected to the same ESP32.

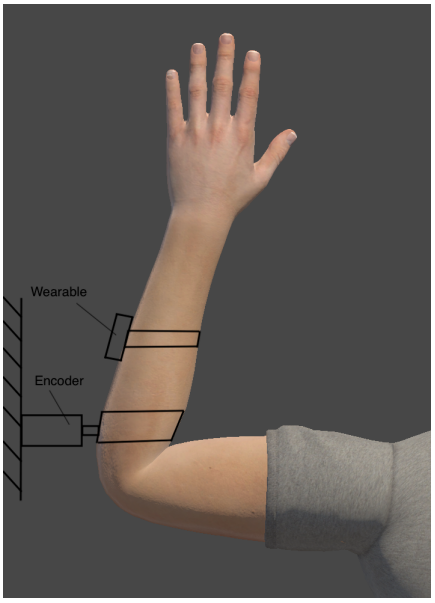

Figure 4.2.: The experimental setup for the comparison between IMU and encoder angle values.

The evaluation of the measured IMU quantities is done while participating in the digital MPPT in a test environment at Yale Center for Engineering. The values are logged into a text file while performed and plotted afterwards for evaluation purposes.

4.1.2. Results

The sensor weight distribution on the sensor surface of $40\,\mathrm{mm} \cdot 40\,\mathrm{mm}$ can be seen in figure 4.3. The plot features twelve calibration points. It shows a nearly linear behavior from $1\,\mathrm{kg}$ until $9\,\mathrm{kg}$. The R square value of 0.97 is high and indicates an excellent fit of the regression model to the data. The weight present in the measurement plot is concentrated on the sensor area. Tests with a patient with full weight of $85\,\mathrm{kg}$ balanced over only one sensor have shown an analog value of 1700 and therefore equals $11\,\mathrm{kg}$ concentrated weight. Tests with the same twelve calibration points on a contact surface of $60\,\mathrm{mm} \cdot 60\,\mathrm{mm}$ have shown no change of the analog value. This shows the weight distribution over the whole TPU sole.

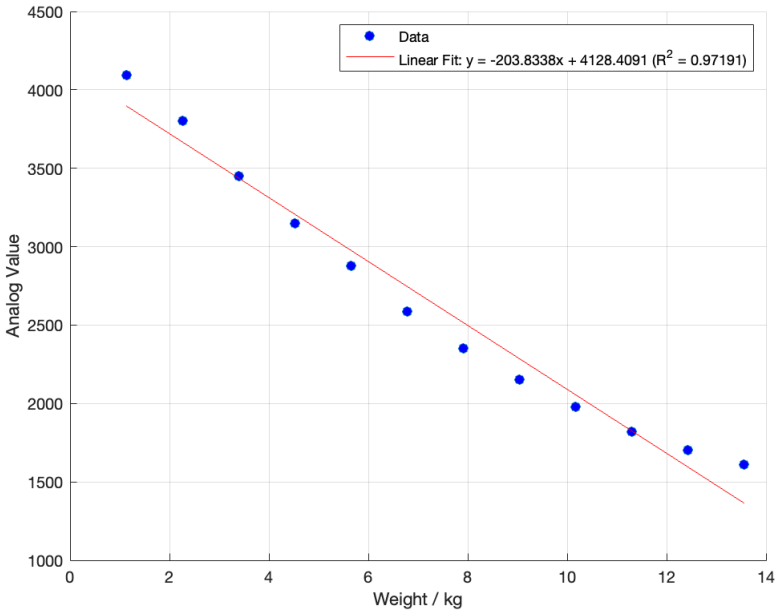

Figure 4.3.: The distribution of weight on the force sensor compared to the resulting analog value, including the linear regression.

The comparison between angle value of IMU and encoder can be seen in figure 4.4. No major lag can be detected. However, there are angle differences when the angles reach their peak values. The absolute mean error is $3.478\,\mathrm{degree}$.

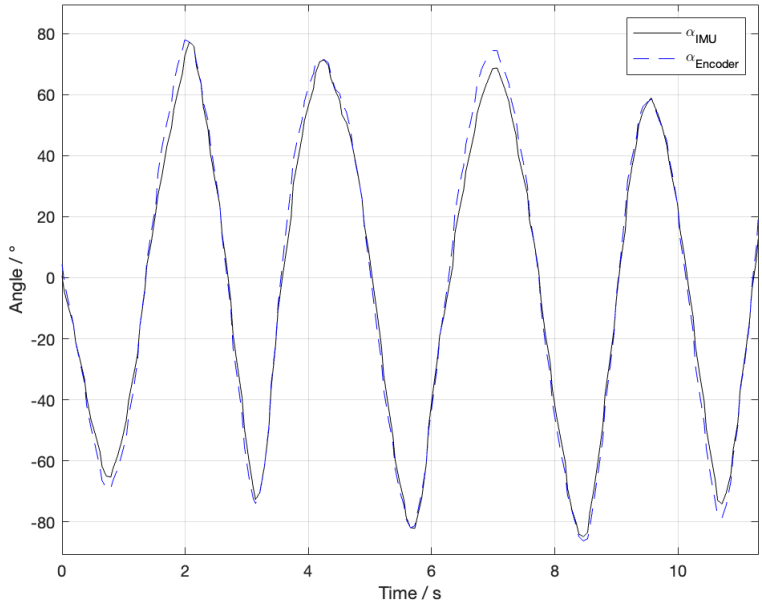

Figure 4.4.: The comparison between IMU and encoder angle values.

Figure 4.5 shows the measured data of the walking sub task. Steps are indicated by the peaks of leg forward rotation angle β and leg forward acceleration a_x.

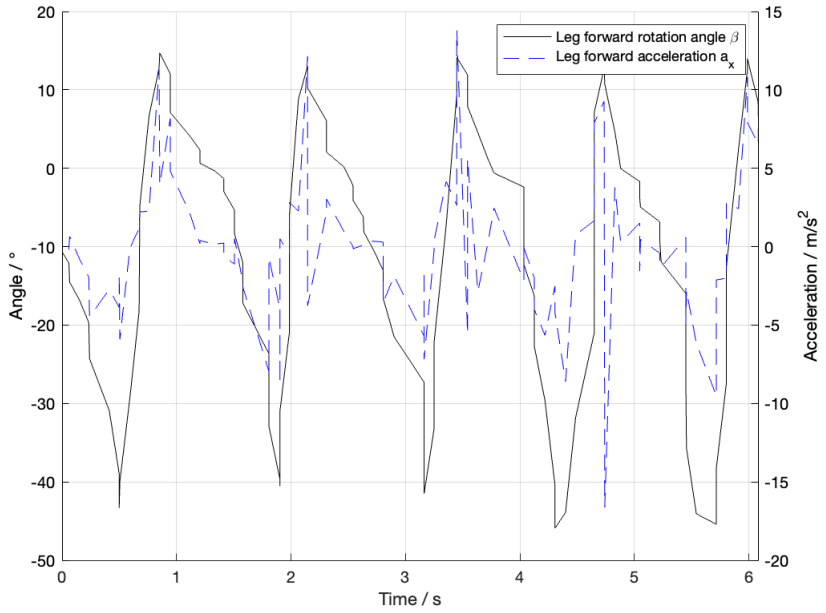

Figure 4.5.: The measured values of sub task 1.

Figure 4.6 shows the measured data of the stair walking sub task. Steps over stairs are indicated by the peaks of leg forward rotation angle β and leg upward acceleration a_y.

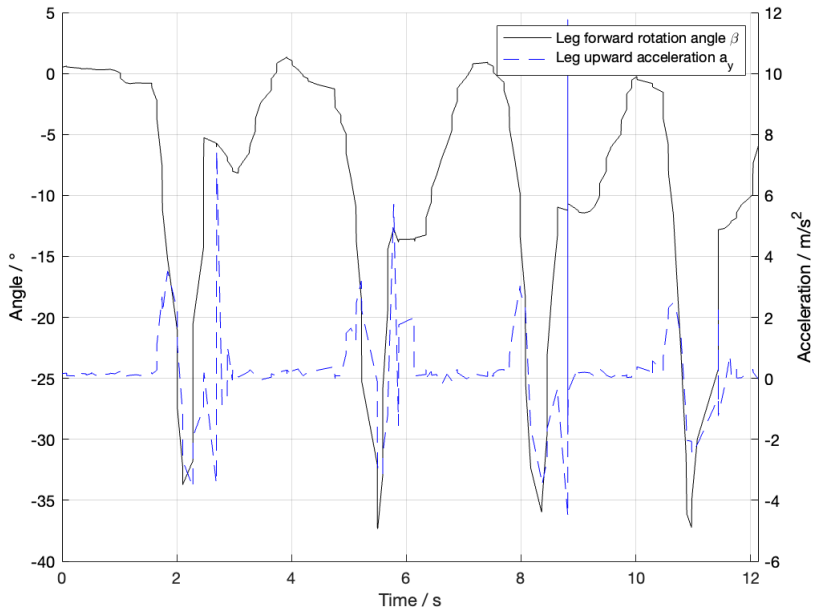

Figure 4.6.: The measured values of sub task 2.

Figure 4.7 shows the measured data of the turning 360 degree sub task. The turning is indicated by a ramp of γ from $0\,°$ to $360\,°$. Angles α and β are small, as they visualize that the upper body is level.

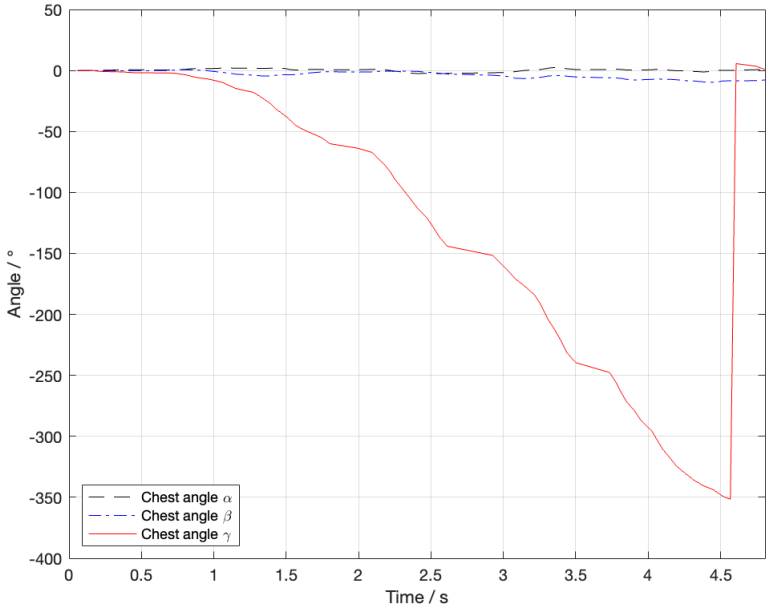

Figure 4.7.: The measured values of sub task 3.

Figure 4.8 shows the measured data of the sit to stand sub task. Upper leg angle α indicates the transition from sit to stand with the five peaks. Chest angle β is small, as it visualize that a left tilt or right tilt of the upper body is not existent.

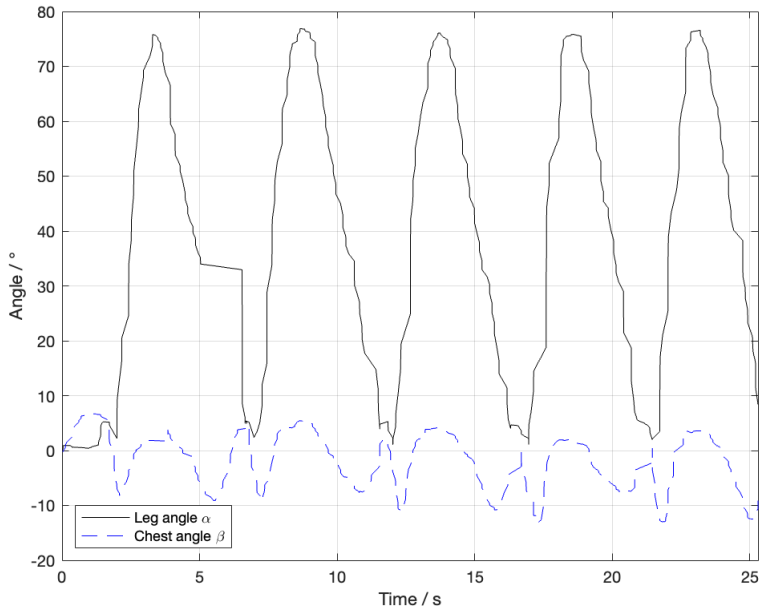

Figure 4.8.: The measured values of sub task 4.

Figure 4.9 shows the measured data of the picking up an object sub task. The data is measured by the IMU in the smart object. The acceleration in z direction a_z is signifi-

cantly higher than a_x and a_y, as the object is lifted from the floor to a table in z direction. The pick up of the object is detected after $0.55\,\mathrm{s}$.

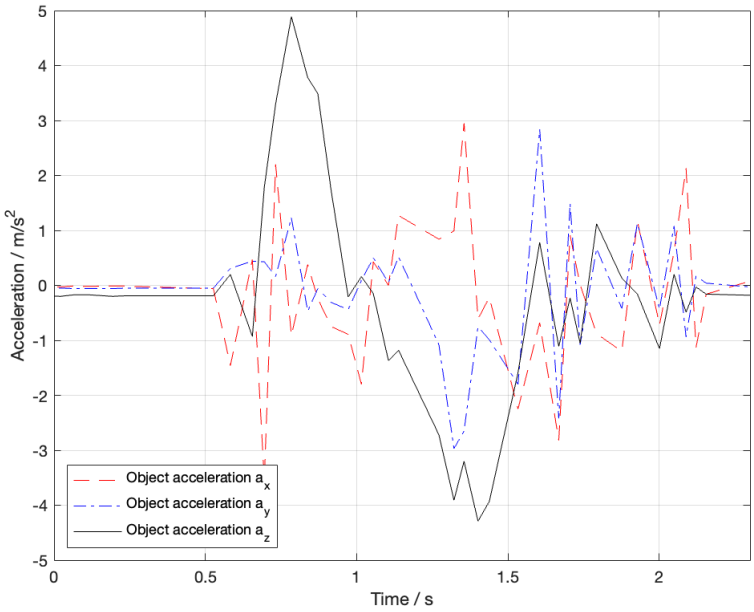

Figure 4.9.: The measured values of sub task 5 and sub task 6.

Figure 4.10 shows the measured data of the balance sub task. The data shows a balanced behavior of the patient in both cases, as the ratio is almost 1.

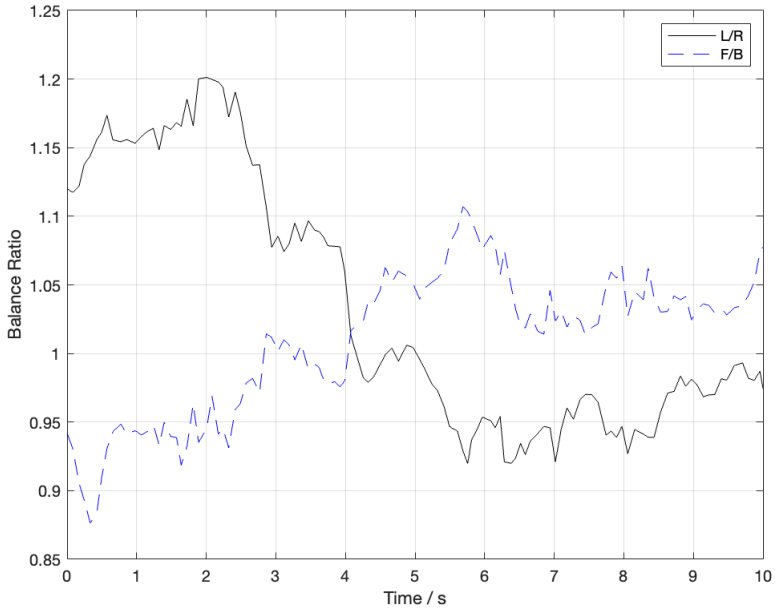

Figure 4.10.: The measured values of sub task 6 and sub task 7. The actual balance of the patient is very good and corresponds with the plot.

4.2. Clinical Evaluation

The project is used and tested in the clinical environment. Patients participate in the digitized MPPT and the result is compared to the traditional MPPT.

4.2.1. Institutional Review Board

In order to let participants take part in the project, a detailed description of the project, the methods used, and the used hardware has to be send to the Institutional Review Board. They have to decide if the university adheres to federal regulations. This is important to make sure that the human participants in research are protected. Without this approval of the Institutional Review Board (IRB) no patient can be recruited to participate in the trial and no data is able to be collected.

The project gained full approval of the Yale University IRB. With this consent ten patients of the Yale rehabilitation unit in Milford hospital were able to participate in the clinical trial.

4.2.2. Study Setup

First wearable 1 and wearable 2 are mounted to both lower legs. Then the computer software is started. After the first sub-task the patients goes to the bottom of the stairs. The patient has to stand in neutral position and the calibration button in the graphical user interface on the PC is pressed. After walking the stairs, the patient can take off wearable 1 and wearable 2. The patient has to put on wearable 3 around the chest. The rehabilitation worker has to press the calibration button in the graphical user interface on the PC and the patient turns 360 degree. After this sub-task, the patients has to sit on the chair and attach wearable 1 at the upper leg. Then the calibration button in the graphical user interface on the PC is pressed. After doing the sub-task, the wearables can be removed of the patient's body. Wearable 4 is placed on the floor and has to be picked up while sitting, after the calibration button is pressed. After that, the same is done while the patient is standing. For the last two tests, the patient has to stand on the soles to balance. The calibration button in the graphical user interface on the PC is pressed and the first balance sub-task starts. The same is done for the second balance test afterwards. The total score is calculated at the end. At the same time, the rehabilitation worker grades the patient with the traditional MPPT. Therefore the time is measured and the grading in the corresponding sub-tasks is done by the rehabilitation worker.

4.2.3. Result of Clinical Study

The test results of the ten patients are collected and compared to the traditional MPPT test. Table 4.1 shows the result comparison of patient 1 to patient 10.

Table 4.1.: The clinical trial patient data.

	Patient 1	Patient 2	Patient 3	Patient 4	Patient 5	Patient 6	Patient 7	Patient 8	Patient 9	Patient 10
Traditional MPPT Score / %	90.625	62.500	71.875	46.875	50.00	31.250	25.000	59.375	31.250	43,750
Digital MPPT Score / %	89.500	64.00	72.625	52.125	56.250	38.625	27.125	61.375	31.375	43.000
Task 1 Traditional Score / %	100	75	75	75	50	25	25	25	100	100
Task 1 Digital Score / %	97	78	78	79	61	41	41	40	96	97
Task 2 Traditional Score / %	100	50	75	75	50	75	25	25	75	75
Task 2 Digital Score / %	81	44	78	73	60	80	41	41	74	69
Task 3 Traditional Score / %	75	50	25	25	25	25	25	25	75	50
Task 3 Digital Score / %	81	63	44	44	44	44	38	44	81	63
Task 4 Traditional Score / %	50	25	25	25	25	25	25	25	0	25
Task 4 Digital Score / %	63	44	44	44	44	44	44	44	0	44
Task 5 Traditional Score / %	100	100	75	75	75	0	0	100	0	0
Task 5 Digital Score / %	100	100	81	81	81	0	0	100	0	0
Task 6 Traditional Score / %	100	100	100	0	75	0	0	75	0	0
Task 6 Digital Score / %	100	94	94	0	81	0	0	81	0	0
Task 7 Traditional Score / %	100	100	100	100	100	100	100	100	0	100
Task 7 Digital Score / %	98	89	69	96	79	100	53	95	0	71
Task 8 Traditional Score / %	100	0	100	0	0	0	0	100	0	0
Task 8 Digital Score / %	96	0	93	0	0	0	0	46	0	0

In order to see a possible correlation between the two scores, data analysis is used. Before checking the correlation the normality of the values has to analyzed in SPSS. The focus is on the Shapiro-Wilk Test, as this test is ideal for a sample size smaller than 50. The Shapiro-Wilk delivers a Sig. number greater than 0.05 Therefore the collected data is normally distributed.

As normality in the data is detected, the Pearson Correlation method can be used to analyze for correlation. By using SPSS the Pearson Correlation method shows a correlation r of 0.990. This indicates a strong correlation as the maximum correlation yields to a r of 1.

5. Discussion

The results indicate that the used wearable including the IMU offers a great precision to measure movements of the human body. The IMU angle measurement has an absolute mean error of $3.478\,^{\circ}$. The comparison with the ideal value of the encoder show almost no offset regarding the angle value. The data suggest that the body movement in the sub tasks is objectively measurable. Steps, turning, bending can be clearly seen in the data plots and behave as predicted. The study demonstrates a strong correlation between traditional MPPT and the digitized version with an r of 0.990.

The results met the expectations and even proofed almost perfect accuracy. This is expected as the IMU BNO055 is used in a lot of different projects and products and known for the high accuracy and reliability.

These results build on existing evidence of the great use cases for digitalization in patient assessment, training and rehabilitation. Those fields are of big importance and the use of digitalization, sensors, and computer software deliver a lot of benefits to the sectors.

It is beyond the scope of this study to focus on all available patient population groups. The main focus is on joint fracture and joint replacement patients in the rehabilitation center. Also the main focus of the MPPT are patients that are still able to follow simple instructions and do those simple sub tasks.

Avenues for future research include the development of a test for patients with conditions which prevent them from taking part in the MPPT. Such conditions could be if they are bound to a wheelchair, or are not able to follow the instructions for the test procedure. This would enable even more population groups to participate in patient assessment.

6. Conclusion

This research aimed on the development of hardware and software to offer better patient assessment. It is shown that the project offers a fast, standardized and accurate way of assessing the patient's physical performance. The patient results can be compared and progress can be tracked. The IMU angle measurement results are comparable to angle measurements of an encoder. The clinical study and the presented correlation between traditional patient assessment and digital patient assessment show further proof for this better patient assessment.

This digital approach of patient assessment is chosen as the traditional one has too many downsides. The patient movements are not measured, there are human errors involved and the procedure is no standardized. The digital version solves all those problems. The use of an IMU offers a small form factor and accurate results. The results of the digital MPPT match give a good overview over the physical abilities of a patient and match with the expected results.

The clinical study was carried out mainly with joint fracture and joint replacement patients in the rehabilitation center. One option for further studies would be the test on participants with Parkinson's disease or stroke patients. They are also struggling in daily life and the test would suit them perfectly. Another possibility would be the use of the wearable for other approaches such as different medical assessment tests or the use of sensors in combination with a VR headset for rehabilitation purposes.

The project shows that digitization in patient assessment brings many benefits and can be used for a big variety of different patient population groups. The advantages for the patient and rehabilitation worker are immense. The patient feedback is positive and they feel encouraged by the feedback of the computer test in combination with the sensors.

Bibliography

[1] K. Berg, Ed., *The Balance Scale: Responding to clinically meaningful changes*. CAN: Canadian Journal of Rehabilitation, 1997.

[2] E. M. Roos and L. S. Lohmander, *The Knee injury and Osteoarthritis Outcome Score (KOOS): from joint injury to osteoarthritis*. Health and quality of life outcomes 1, 2003, November 03. [Online]. Available: https://doi.org/10.1186/1477-7525-1-64 [Accessed: 2023, Aug 08]

[3] AbilityLab, *Modified Physical Performance Test*. Chicago, IL, 2013. [Online]. Available: https://www.sralab.org/rehabilitation-measures/modified-physical-performance-test [Accessed: 2022, Aug 03]

[4] H. S. C. of Tompkins County, *Modified Physical Performance Scoring Sheet*. Ithaca, NY, 2021. [Online]. Available: https://hsctc.org/wp-content/uploads/2017/07/Modified-Physical-Performance-Test.pdf [Accessed: 2022, Aug 03]

[5] V. Vasco *et al.*, *HR1 Robot: An Assistant for Healthcare Applications*. Frontiers in robotics and AI, 2022. [Online]. Available: https://doi.org/10.3389/frobt.2022.813843 [Accessed: 2022, Mar 07]

[6] H. Uustal and E. Baerga, Eds., *Physical Medicine and Rehabilitation Board Review*. NY, USA: Demos MedicalPublishing: New York, 2004.

[7] L. Ojeda *et al.*, *Estimating Stair Running Performance Using Inertial Sensors*. Basel, Switzerland, 2017. [Online]. Available: https://doi.org/10.3390/s17112647 [Accessed: 2022, Sep 03]

[8] J. D. Sui and T. S. Chang, *Deep Gait Tracking With Inertial Measurement Unit*. Institute of Electronics, National Chiao Tung University, 2019. [Online]. Available: https://arxiv.org/pdf/2205.04666.pdf [Accessed: 2022, Aug 03]

[9] Y. Zhang *et al.*, "Can wearable devices and machine learning techniques be used for recognizing and segmenting modified physical performance test items?" *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 30, pp. 1776–1785, 2022.

[10] J. van den Helder *et al.*, "A digitally supported home-based exercise training program and dietary protein intervention for community dwelling older adults: protocol of the cluster randomised controlled vitamin trial." *BMC Geriatrics*, vol. 18, no. 183, 2018.

[11] C. Harpham *et al.*, “Co-creating a feasible, acceptable and safe home-based high-intensity interval training programme for people with parkinson& rsquo;s: The hiit-home4parkinson& rsquo;s study,” *International Journal of Environmental Research and Public Health*, vol. 20, no. 9, 2023.

[12] van den Helder J. *et al.*, “Bio-electrical impedance analysis: A valid assessment tool for diagnosis of low appendicular lean mass in older adults?” *Front. Nutr.*, vol. 9, 2022.

[13] Y. University, *About Bulldog RepBox*. Bulldog RepBox, 2018. [Online]. Available: https://bulldogrepbox.com [Accessed: 2023, Mar 07]

[14] I. A. Faisal *et al.*, “A review of accelerometer sensor and gyroscope sensor in imu sensors on motion capture,” *J. Eng. Appl. Sci*, vol. 15, no. 3, pp. 826–829, 2019.

[15] A. Becker. (2023) Introduction to kalman filter. [Online]. Available: https://www.kalmanfilter.net [Accessed: 2023, May 14]

[16] B. Sensortec, *Datasheet BNO055*. Bosch, 2023. [Online]. Available: https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bno055-ds000.pdf [Accessed: 2023, Mar 07]

[17] K. Townsend, *Adafruit BNO055 Absolute Orientation Sensor*. Adafruit Industries, 2020. [Online]. Available: https://www.sigmaelectronica.net/wp-content/uploads/2017/03/adafruit-bno055-absolute-orientation-sensor.pdf [Accessed: 2022, Mar 07]

[18] R. Mischianti, *BNO055 for esp32, esp8266, and Arduino: wiring and advanced Bosch library*. blog of digital electronics and programming, 2023. [Online]. Available: https://www.mischianti.org/2022/11/03/bno055-for-esp32-esp8266-and-arduino-wiring-and-advanced-bosch-library-2/ [Accessed: 2022, Mar 09]

List of Figures

List of Tables

List of Symbols

symbol	name	unit
q_{0prime}	init. first quat. comp.	$-$
q_{1prime}	init. second quat. comp.	$-$
q_{2prime}	init. third quat. comp.	$-$
q_{3prime}	init. fourth quat. comp.	$-$
q_0	mod. first quat. comp.	$-$
q_1	mod. second quat. comp.	$-$
q_2	mod. third quat. comp.	$-$
q_3	mod. fourth quat. comp.	$-$
α	angle Alpha	$^\circ$
β	angle Beta	$^\circ$
γ	angle Gamma	$^\circ$
$\alpha_{chestmax}$	max. alpha angle chest	$^\circ$
$\beta_{chestmax}$	max. beta angle chest	$^\circ$
b_{LRmean}	mean left right bal. ratio	$-$
b_{FBmean}	mean front back bal. ratio	$-$

Abbreviations

MPPT	Modified Physical Performance Test
KOOS	Knee Injury and Osteoarthritis Outcome Score
IMU	Inertial Measurement Unit
MQTT	Message Queuing Telemetry Transport
IoT	Internet of Things
CAD	Computer Aided Design
TPU	Thermoplastic Polyurethane
PLA	Polylactic Acid
IRB	Institutional Review Board

A. Code

A.1. Wearable IMU ESP32

```
#include <Wire.h>                       //Code used for IMU modules
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>
#include <WiFi.h>
#include <PubSubClient.h>
#define ONBOARD_LED  13

//###################### When Msoquitto is running on your laptop ########################
#define AIO_SERVER      "192.168.68.117"        // your IP address
#define AIO_SERVERPORT  1883                    // use 8883 for SSL
#define AIO_USERNAME    ""
#define AIO_KEY         ""
//###################### When Msoquitto is running on your laptop ########################

//////////////////////Definitions WIFI and MQTT/////////////////////////////////////////

const char* ssid = "yale_wireless";     //"yale wireless"; //"HSC_Guest"; //"Mesh :)";
const char* password =  ""; //"84LionSt";
const char* mqtt_Server = "broker.mqtt.cool";//or  own PC  AIO_SERVER; for own Mosquitto broker
const int mqttPort = 1883;

const char* publishTopic = "AAL_MCI_esp_Publish_Poti/WolfgangGrosek/1"; // make a unique topic for yourself -->
      Topic ".../1" used for IMU1, ".../2" for IMU2 etc.
const char* subscribeTopic = "AAL_MCI_esp_Subscribe/WolfgangGrosek"; // make a unique topic for yourself
const char* clientName = "AAL_MCI_esp_WolfgangGrosek_1";  // make a unique topic for yourself  --> Client Name ".. 
      _1" used for IMU1, ".._2" for IMU2 etc.

const char* mqttUser = "yourMQTTuser_MCI_AAL_";  // for secure connection
const char* mqttPassword = "yourMQTTpasswordpass"; // for secure connection

WiFiClient espClient;  // make the name unique
PubSubClient client(espClient);

//////////////////////Definitions BNO and Calculation/////////////////////////////////////

bool calib = false;    //calibration status
float w = 0.0, x = 0.0, y = 0.0, z = 0.0, offset_x = 0.0, offset_y = 0.0, offset_z = 0.0;
uint16_t BNO055_SAMPLERATE_DELAY_MS = 10; //how often to read data from the board
uint16_t PRINT_DELAY_MS = 500; // how often to print the data
uint16_t printCount = 0; //counter to avoid printing every 10MS sample

// Check I2C device address and correct line below (by default address is 0x29 or 0x28)
//                                   id, address
Adafruit_BNO055 bno = Adafruit_BNO055(55, 0x28);

imu::Vector<3> v;

String angle_str;

//////////////////Definitions for blink LED//////////////////////////////

// constants won't change. Used here to set a pin number:
const int ledPin = ONBOARD_LED;  // the number of the LED pin

// Variables will change:
int ledState = LOW;  // ledState used to set the LED

// Generally, you should use "unsigned long" for variables that hold time
// The value will quickly become too large for an int to store
unsigned long previousMillis = 0;  // will store last time LED was updated

// constants won't change:
const long interval = 100;  // interval at which to blink (milliseconds)
```

```
////////////////////////////////////////////////////////////////////////

void setup(void)
{
  Serial.begin(115200);
  setup_wifi();
  client.setServer(mqtt_Server, 1883);
  //client.setCallback(callback);

  while (!Serial) delay(10);  // wait for serial port to open!
  bno.begin(OPERATION_MODE_IMUPLUS); // set operation mode
  if (!bno.begin())
  {
    Serial.print("No␣BNO055␣detected");
    while (1);
  }
  bno.setMode(OPERATION_MODE_IMUPLUS);
  pinMode(ONBOARD_LED, OUTPUT);
  delay(1000);
}

void setup_wifi() {
  delay(10);
  // We start by connecting to a WiFi network
  Serial.println();
  Serial.print("Connecting␣to␣");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi␣connected");
  Serial.println("IP␣address:␣");
  Serial.println(WiFi.localIP()); // print your IP address
}

// constant mqtt connection
void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting␣MQTT␣connection...");
    // Attempt to connect
    if (client.connect(clientName)) {
      Serial.println("connected");
      // Once connected, publish an announcement...
      //client.publish(publishTopic, "Reconnected");
      // ... and resubscribe
      client.subscribe(subscribeTopic);
    } else {
      Serial.print("failed,␣rc=");
      Serial.print(client.state());
      Serial.println("␣try␣again␣in␣5␣seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}


void loop(void)
{
  if (!client.connected()) {
    reconnect();
  }
  client.loop();

  unsigned long tStart = micros(); //count time

  imu::Quaternion quat = bno.getQuat();   //get quaternions to describe orientation
  imu::Vector<3> accel = bno.getVector(Adafruit_BNO055::VECTOR_LINEARACCEL);

  if (printCount * BNO055_SAMPLERATE_DELAY_MS >= PRINT_DELAY_MS) {
    //enough iterations have passed that we can print the latest data
    displayCalStatus();
    printCount = 0;
```

```
  }
  else {
    printCount = printCount + 1;
  }

  while ((micros() - tStart) < (BNO055_SAMPLERATE_DELAY_MS * 1000))
  {
    //poll until the next sample is ready
  }

  angle_str = String(quat.w(), 4); //converting long to a string
  String angle_str1 = String(quat.x(), 4);
  String angle_str2 = String(quat.y(), 4);
  String angle_str3 = String(quat.z(), 4);
  String accel_str1 = String(accel.x());
  String accel_str2 = String(accel.y());
  String accel_str3 = String(accel.z());

  String string = angle_str + "," + angle_str1 + "," + angle_str2 + "," + angle_str3 + "," + accel_str1 + "," +
       accel_str2 + "," + accel_str3;

  char angle[string.length() + 1] = {0};
  string.toCharArray(angle, string.length() + 1); //packaging up the data to publish to mqtt whoa...
  //Serial.println(angle);
  //Serial.println(string);
  //Serial.println(angle);
  client.publish(publishTopic, angle, 4);
  angle[0] = 0;

  delay(50); //10ms ideal // 50ms because of clinic
}


/**************************************************************************/
/*
    Display sensor calibration status
*/
/**************************************************************************/
void displayCalStatus(void)
{
  /* Get the four calibration values (0..3) */
  /* Any sensor data reporting 0 should be ignored, */
  /* 3 means 'fully calibrated" */
  uint8_t system, gyro, accel, mag;
  system = gyro = accel = mag = 0;
  bno.getCalibration(&system, &gyro, &accel, &mag);

  if (system * gyro * accel * mag == 81) //used to determine full calibration --> calib == true --> LED blink
  {
    unsigned long currentMillis = millis();
    calib = true;

    if (currentMillis - previousMillis >= interval) {
      // save the last time you blinked the LED
      previousMillis = currentMillis;

      // if the LED is off turn it on and vice-versa:
      if (ledState == LOW) {
        ledState = HIGH;
      } else {
        ledState = LOW;
      }
      // set the LED with the ledState of the variable:
      digitalWrite(ledPin, ledState);
    }
  }
}
```

A.2. Force Sensor ESP32

```
#include <Wire.h>                       //Code used for balance module
#include <Adafruit_Sensor.h>
#include <WiFi.h>
#include <PubSubClient.h>

//###################### When Msoquitto is running on your laptop ########################
#define AIO_SERVER      "192.168.68.117"        // your IP address
#define AIO_SERVERPORT  1883                   // use 8883 for SSL
#define AIO_USERNAME    ""
```

```
#define AIO_KEY         ""
//###################### When Msoquitto is running on your laptop ########################

//////////////////////Definitions WIFI and MQTT/////////////////////////////////////////

const char* ssid = "yale wireless";    //"yale wireless";   //"HSC_Guest";
const char* password =  "";
const char* mqtt_Server = "broker.mqtt.cool";//or  own PC  AIO_SERVER; for own Mosquitto broker
const int mqttPort = 1883;

const char* publishTopic = "AAL_MCI_esp_Publish_Poti/WolfgangGrosek/5"; // make a unique topic for yourself -->
     Topic ".../5" used for sole sensors
const char* subscribeTopic = "AAL_MCI_esp_Subscribe/WolfgangGrosek"; // make a unique topic for yourself
const char* clientName = "AAL_MCI_esp_WolfgangGrosek_5";  // make a unique topic for yourself  --> Client Name ".."
     _5" used for sole sensors

const char* mqttUser = "yourMQTTuser_MCI_AAL_";  // for secure connection
const char* mqttPassword = "yourMQTTpasswordpass"; // for secure connection

WiFiClient espClient;  // make the name unique
PubSubClient client(espClient);

//////////////////////Definitions BNO and Calculation/////////////////////////////////////

bool calib = false;    //calibration status

float sensorValue1;
float sensorValue2;
float sensorValue3;
float sensorValue4;
uint16_t BNO055_SAMPLERATE_DELAY_MS = 10; //how often to read data from the board
uint16_t PRINT_DELAY_MS = 500; // how often to print the data
uint16_t printCount = 0; //counter to avoid printing every 10MS sample
uint16_t zahler = 0;    // used to count how often loop was executed after calib


String angle_str;


void setup(void)
{
  Serial.begin(115200);
  setup_wifi();
  client.setServer(mqtt_Server, 1883);
  //client.setCallback(callback);
}

void setup_wifi() {
  delay(10);
  // We start by connecting to a WiFi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP()); // print your IP address
}

// constant mqtt connection
void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Attempt to connect
    if (client.connect(clientName)) {
      Serial.println("connected");
      // Once connected, publish an announcement...
      //client.publish(publishTopic, "Reconnected");
      // ... and resubscribe
      client.subscribe(subscribeTopic);
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
```

```
      Serial.println("␣try␣again␣in␣5␣seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}

void loop(void) //used to measure sensor value 1 to 4 and send via MQTT
{
  //RIGHT FOOT
  sensorValue2 = analogRead(36);  // red wire
  sensorValue1 = analogRead(39); // yellow wire

  //RIGHT FOOT
  sensorValue4 = analogRead(34); // yellow wire
  sensorValue3 = analogRead(33); // red wire

  if (!client.connected()) {
    reconnect();
    }
  client.loop();

  angle_str = String(sensorValue1); //converting float to a string
  String angle_str1 = String(sensorValue2);
  String angle_str2 = String(sensorValue3);
  String angle_str3 = String(sensorValue4);

  String string = angle_str + "," + angle_str1 + "," + angle_str2 + "," + angle_str3;

  char angle[string.length() + 1] = {0};
  string.toCharArray(angle, string.length() + 1); //packaging up the data to publish to mqtt whoa...
  client.publish(publishTopic, angle, 4);
  angle[0] = 0; //empty char

  delay(50); //50ms time delay sufficent
}
```

A.3. MPPT Software

```
import random  #import libraries
import time
import math
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import tkinter as tk

from matplotlib import style
from paho.mqtt import client as mqtt_client


broker = 'broker.mqtt.cool' #define broker and port for mqtt
port = 1883
topic =  "AAL_MCI_esp_Publish_Poti/WolfgangGrosek/1"    #topics for each wearable/ESP32
topic1 = "AAL_MCI_esp_Publish_Poti/WolfgangGrosek/2"
topic2 = "AAL_MCI_esp_Publish_Poti/WolfgangGrosek/3"
topic3 = "AAL_MCI_esp_Publish_Poti/WolfgangGrosek/4"
topic4 = "AAL_MCI_esp_Publish_Poti/WolfgangGrosek/5"

# Generate a Client ID with the subscribe prefix.
client_id = f'subscribe-{random.randint(0,␣100)}'
# username = 'emqx'
# password = 'public'

T1_IMU1AngleX=[]    #definition of list variables for each subtask T1-T8
T1_IMU1AngleY=[]
T1_IMU1AngleZ=[]
T1_IMU1AccX=[]
T1_IMU1AccY=[]
T1_IMU1AccZ=[]
T1_IMU1Time=[]

T1_IMU2AngleX=[]
T1_IMU2AngleY=[]
T1_IMU2AngleZ=[]
T1_IMU2AccX=[]
T1_IMU2AccY=[]
T1_IMU2AccZ=[]
T1_IMU2Time=[]
```

```
T2_IMU1AngleX = []
T2_IMU1AngleY = []
T2_IMU1AngleZ = []
T2_IMU1AccX = []
T2_IMU1AccY = []
T2_IMU1AccZ = []
T2_IMU1Time = []

T2_IMU2AngleX = []
T2_IMU2AngleY = []
T2_IMU2AngleZ = []
T2_IMU2AccX = []
T2_IMU2AccY = []
T2_IMU2AccZ = []
T2_IMU2Time = []

T3_IMU1AngleX = []
T3_IMU1AngleY = []
T3_IMU1AngleZ = []
T3_IMU1AccX = []
T3_IMU1AccY = []
T3_IMU1AccZ = []
T3_IMU1Time = []

T4_IMU1AngleX = []
T4_IMU1AngleY = []
T4_IMU1AngleZ = []
T4_IMU1AccX = []
T4_IMU1AccY = []
T4_IMU1AccZ = []
T4_IMU1Time = []

T4_IMU2AngleX = []
T4_IMU2AngleY = []
T4_IMU2AngleZ = []
T4_IMU2AccX = []
T4_IMU2AccY = []
T4_IMU2AccZ = []
T4_IMU2Time = []

T5_IMU1AngleX = []
T5_IMU1AngleY = []
T5_IMU1AngleZ = []
T5_IMU1AccX = []
T5_IMU1AccY = []
T5_IMU1AccZ = []
T5_IMU1Time = []

T6_IMU1AngleX = []
T6_IMU1AngleY = []
T6_IMU1AngleZ = []
T6_IMU1AccX = []
T6_IMU1AccY = []
T6_IMU1AccZ = []
T6_IMU1Time = []

T7_Sensor1 = []
T7_Sensor2 = []
T7_Sensor3 = []
T7_Sensor4 = []
T7_Time = []

T8_Sensor1 = []
T8_Sensor2 = []
T8_Sensor3 = []
T8_Sensor4 = []
T8_Time = []

_MESSAGE = {'TOPIC': "␣"}
_SUBTASK = {'TASK': "␣"}
_NEW = {'NEW': True}


#MPPT subtask 1:                    #definition of variables for each subtask T1-T8
_T1_TIME = {'STARTTIME': 0.0, 'ENDTIME': 0.0}
_T1_SCORE = {'BALANCE': 0.0, 'SCORE': 0.0}

_T1_LEFT_CALIB = {'CALIB': False, 'X': 0.0, 'Y': 0.0, 'Z': 0.0}
_T1_LEFT_STEP = {'INSTEP': False, 'STEP': 0}
_T1_LEFT_TIME = {'STEPTIME': 0.0, 'STARTSTEP': 0.0}
```

```
    _T1_RIGHT_CALIB = {'CALIB': False, 'X': 0.0, 'Y': 0.0, 'Z': 0.0}
    _T1_RIGHT_STEP = {'INSTEP': False, 'STEP': 0}
124 _T1_RIGHT_TIME = {'STEPTIME': 0.0, 'STARTSTEP': 0.0}

    #MPPT subtask 2:
    _T2_TIME = {'STARTTIME': 0.0, 'ENDTIME': 0.0}
    _T2_SCORE = {'BALANCE': 0.0, 'SCORE': 0.0}
129
    _T2_LEFT_CALIB = {'CALIB': False, 'X': 0.0, 'Y': 0.0, 'Z': 0.0}
    _T2_LEFT_STEP = {'INSTEP': False, 'STEP': 0}
    _T2_LEFT_TIME = {'STEPTIME': 0.0, 'STARTSTEP': 0.0}

134 _T2_RIGHT_CALIB = {'CALIB': False, 'X': 0.0, 'Y': 0.0, 'Z': 0.0}
    _T2_RIGHT_STEP = {'INSTEP': False, 'STEP': 0}
    _T2_RIGHT_TIME = {'STEPTIME': 0.0, 'STARTSTEP': 0.0}

    #MPPT subtask 3:
139 _T3_TIME = {'STARTTIME': 0.0, 'ENDTIME': 0.0}
    _T3_TURN = {'MOTION': False, 'X': 0.0, 'Y': 0.0}
    _T3_SCORE = {'BALANCE': 0.0, 'SCORE': 0.0}

    _T3_CHEST_CALIB = {'CALIB': False, 'X': 0.0, 'Y': 0.0, 'Z': 0.0}
144 _T3_CHEST_STEP = {'INSTEP': False, 'STEP': 0}
    _T3_CHEST_TIME = {'STEPTIME': 0.0, 'STARTSTEP': 0.0}

    #MPPT subtask 4:
    _T4_TIME = {'STARTTIME': 0.0, 'ENDTIME': 0.0}
149 _T4_SCORE = {'BALANCE': 0.0, 'SCORE': 0.0}

    _T4_TILT = {'X': 0.0, 'Y': 0.0}
    _T4_CHEST_CALIB = {'CALIB': False, 'X': 0.0, 'Y': 0.0, 'Z': 0.0}

154 _T4_RIGHT_CALIB = {'CALIB': False, 'X': 0.0, 'Y': 0.0, 'Z': 0.0}
    _T4_RIGHT_STEP = {'INSTEP': False, 'STEP': 0}
    _T4_RIGHT_TIME = {'STEPTIME': 0.0, 'STARTSTEP': 0.0}

    #MPPT subtask 5:
159 _T5_TIME = {'STARTTIME': 0.0, 'ENDTIME': 0.0}
    _T5_ACC = {'MEANACC': 0.0, 'COUNT': 0}
    _T5_SCORE = {'BALANCE': 0.0, 'SCORE': 0.0}

    _T5_OBJECT_CALIB = {'CALIB': False, 'X': 0.0, 'Y': 0.0, 'Z': 0.0}
164 _T5_OBJECT_STEP = {'INSTEP': False, 'STEP': 0}
    _T5_OBJECT_TIME = {'STEPTIME': 0.0, 'STARTSTEP': 0.0}

    #MPPT subtask 6:
    _T6_TIME = {'STARTTIME': 0.0, 'ENDTIME': 0.0}
169 _T6_ACC = {'MEANACC': 0.0, 'COUNT': 0}
    _T6_SCORE = {'BALANCE': 0.0, 'SCORE': 0.0}

    _T6_OBJECT_CALIB = {'CALIB': False, 'X': 0.0, 'Y': 0.0, 'Z': 0.0}
    _T6_OBJECT_STEP = {'INSTEP': False, 'STEP': 0}
174 _T6_OBJECT_TIME = {'STEPTIME': 0.0, 'STARTSTEP': 0.0}

    #MPPT subtask 7:
    _T7_TIME = {'STARTTIME': 0.0, 'ENDTIME': 0.0}
    _T7_BALANCE = {'LR': 0.0, 'FB': 0.0}
179 _T7_MEAN = {'LR': 0.0, 'FB': 0.0, 'COUNT': 0}
    _T7_CALIB = {'CALIB': False}
    _T7_SCORE = {'SCORE': 0.0}

    #MPPT subtask 8:
184 _T8_TIME = {'STARTTIME': 0.0, 'ENDTIME': 0.0}
    _T8_BALANCE = {'LR': 0.0}
    _T8_MEAN = {'LR': 0.0, 'FB': 0.0, 'COUNT': 0}
    _T8_CALIB = {'CALIB': False}
    _T8_SCORE = {'SCORE': 0.0}
189
    #MPPT Total:                #General var definitions for total test
    _SCORE = {'SCORE': 0.0}
    _TIME = {'STARTTIME': 0.0}

194


    def connect_mqtt() -> mqtt_client:          #connect to mqtt
        def on_connect(client, userdata, flags, rc):
            if rc == 0:
199             print("Connected␣to␣MQTT␣Broker!")
            else:
                print("Failed␣to␣connect,␣return␣code␣%d\n", rc)

        client = mqtt_client.Client(client_id)
```

```
    # client.username_pw_set(username, password)
    client.on_connect = on_connect
    client.connect(broker, port)
    return client

def subscribe(client: mqtt_client):          #subscribe to different clients and define
    def on_message(client, userdata, msg):  #what happens if message is received (subtask and wearable number)
        #print(f"Received `{msg.payload.decode()}` from `{msg.topic}` topic")
        _MESSAGE['TOPIC'] = {msg.topic}
        if _MESSAGE['TOPIC'] == {'AAL_MCI_esp_Publish_Poti/WolfgangGrosek/1'} and _SUBTASK['TASK'] == {'T1'}:
            T1Leftfoot(msg)
        if _MESSAGE['TOPIC'] == {'AAL_MCI_esp_Publish_Poti/WolfgangGrosek/2'} and _SUBTASK['TASK'] == {'T1'}:
            T1Rightfoot(msg)
        if _MESSAGE['TOPIC'] == {'AAL_MCI_esp_Publish_Poti/WolfgangGrosek/1'} and _SUBTASK['TASK'] == {'T2'}:
            T2Leftfoot(msg)
        if _MESSAGE['TOPIC'] == {'AAL_MCI_esp_Publish_Poti/WolfgangGrosek/2'} and _SUBTASK['TASK'] == {'T2'}:
            T2Rightfoot(msg)
        if _MESSAGE['TOPIC'] == {'AAL_MCI_esp_Publish_Poti/WolfgangGrosek/3'} and _SUBTASK['TASK'] == {'T3'}:
            T3Chest(msg)
        if _MESSAGE['TOPIC'] == {'AAL_MCI_esp_Publish_Poti/WolfgangGrosek/1'} and _SUBTASK['TASK'] == {'T4'}:
            T4Rightfoot(msg)
        if _MESSAGE['TOPIC'] == {'AAL_MCI_esp_Publish_Poti/WolfgangGrosek/3'} and _SUBTASK['TASK'] == {'T4'}:
            T4Chest(msg)
        if _MESSAGE['TOPIC'] == {'AAL_MCI_esp_Publish_Poti/WolfgangGrosek/4'} and _SUBTASK['TASK'] == {'T5'}:
            T5Object(msg)
        if _MESSAGE['TOPIC'] == {'AAL_MCI_esp_Publish_Poti/WolfgangGrosek/4'} and _SUBTASK['TASK'] == {'T6'}:
            T6Object(msg)
        if _MESSAGE['TOPIC'] == {'AAL_MCI_esp_Publish_Poti/WolfgangGrosek/5'} and _SUBTASK['TASK'] == {'T7'}:
            T7Balance(msg)
        if _MESSAGE['TOPIC'] == {'AAL_MCI_esp_Publish_Poti/WolfgangGrosek/5'} and _SUBTASK['TASK'] == {'T8'}:
            T8Balance(msg)

    client.subscribe(topic)     #subscribe to topics for each wearable
    client.subscribe(topic1)
    client.subscribe(topic2)
    client.subscribe(topic3)
    client.subscribe(topic4)
    client.on_message = on_message

def calibration():      #needed to define zero angle position/start position

    _T2_LEFT_CALIB['CALIB'] = True
    print("Calibrated")


def rotatequa(q0prime, q1prime, q2prime, q3prime, axis):    #needed if wearable is attached different as original
                                                           #coordinate system
    if axis == 0:
        q0 = (math.sqrt(2)/2) * q0prime - (math.sqrt(2)/2) * q1prime;#rotate about x
        q1 = (math.sqrt(2)/2) * q1prime + (math.sqrt(2)/2) * q0prime;
        q2 = (math.sqrt(2)/2) * q2prime + (math.sqrt(2)/2) * q3prime;
        q3 = (math.sqrt(2)/2) * q3prime - (math.sqrt(2)/2) * q2prime;

    if axis == 1:
        q0 = (math.sqrt(2)/2) * q0prime - (math.sqrt(2)/2) * q2prime;#rotate about y
        q1 = (math.sqrt(2)/2) * q1prime - (math.sqrt(2)/2) * q3prime;
        q2 = (math.sqrt(2)/2) * q2prime + (math.sqrt(2)/2) * q0prime;
        q3 = (math.sqrt(2)/2) * q3prime + (math.sqrt(2)/2) * q1prime;

    if axis == 2:
        q0 = (math.sqrt(2)/2) * q0prime - (math.sqrt(2)/2) * q3prime;#rotate about z
        q1 = (math.sqrt(2)/2) * q1prime + (math.sqrt(2)/2) * q2prime;
        q2 = (math.sqrt(2)/2) * q2prime - (math.sqrt(2)/2) * q1prime;
        q3 = (math.sqrt(2)/2) * q3prime + (math.sqrt(2)/2) * q0prime;

    phi = (360/(2*3.141592653))*math.atan2(2*(q0*q1+q2*q3),1-2*(q1*q1+q2*q2));  #quat to euler angle

    if (2*(q0*q2-q3*q1)) <-1: theta = (360/(2*3.141592653))*math.asin(-1)
    elif (2*(q0*q2-q3*q1)) >1: theta = (360/(2*3.141592653))*math.asin(1)
    else: theta = (360/(2*3.141592653))*math.asin(2*(q0*q2-q3*q1));

    chi = (360/(2*3.141592653))*math.atan2(2*(q0*q3+q1*q2),1-2*(q2*q2+q3*q3));

    return phi, theta, chi;


def T1Leftfoot(msg):        #Subtask1 wearable left foot
    if _T2_LEFT_CALIB['CALIB'] ==True:  #only start if wearable calibrated
        temp = list(map(float, msg.payload.decode().split(",")))    #split string to individual values
```

```
        if len(temp)==7:    #make sure number of elements is 7
            temp[0], temp[1], temp[2] = rotatequa(temp[0], temp[1], temp[2], temp[3], 0)    #rotation and quat to
                euler angles
            temp[0]=temp[0]-_T1_LEFT_CALIB['X'] #actual value is angle minus offset
            temp[1]=temp[1]-_T1_LEFT_CALIB['Y']
            temp[2]=temp[2]-_T1_LEFT_CALIB['Z']


            if _T1_LEFT_CALIB['X'] == 0.0:      #used to define offset with first run
                _T1_LEFT_CALIB['X'] = temp[0];
                _T1_LEFT_CALIB['Y'] = temp[1];
                _T1_LEFT_CALIB['Z'] = temp[2];
                _TIME['STARTTIME']=time.time()

            else:
                T1_IMU1AngleX.append(temp[0])   #starting from second run, no offset needed because already defined
                T1_IMU1AngleY.append(temp[1])
                T1_IMU1AngleZ.append(temp[2])

                T1_IMU1AccX.append(temp[4])
                T1_IMU1AccY.append(temp[5])
                T1_IMU1AccZ.append(temp[6])
                T1_IMU1Time.append(time.time()-_TIME['STARTTIME'])  #define current time stamp

            #print(f"Angle: {temp[1]}")

            if (temp[4] > 2.5) and (_T1_LEFT_STEP ['INSTEP'] == False) and (temp[1] > 10.0):    #in step movement
                _T1_LEFT_STEP ['INSTEP'] = True
                _T1_LEFT_TIME['STARTSTEP'] = time.time()
                if (_T1_LEFT_STEP ['STEP'] == 0) and ( _T1_TIME['STARTTIME'] == 0.0):   #starttime definition
                    _T1_TIME['STARTTIME'] = time.time()

            if (temp[4] > -0.4) and (_T1_LEFT_STEP ['INSTEP'] == True) and (temp[1] < 6.0): #step is over
                _T1_LEFT_STEP ['INSTEP'] = False
                _T1_LEFT_STEP ['STEP'] += 1     #increase step counter
                print(f"Left Steps: {_T1_LEFT_STEP ['STEP']}")
                _T1_LEFT_TIME['STEPTIME'] = _T1_LEFT_TIME['STEPTIME'] + (time.time() - _T1_LEFT_TIME['STARTSTEP'])
                    #calc time foot in air

            if _T1_LEFT_STEP ['STEP'] + _T1_RIGHT_STEP ['STEP'] == 10:  #if 10 steps detected finish calculations
                and save to file, go to next subtask
                _T1_TIME['ENDTIME'] = time.time() - _T1_TIME['STARTTIME']   #calc of total time for 10 steps

                print()
                print(f"TotalTime: {_T1_TIME['ENDTIME']}")
                #print(f"StepTime: {_T1_LEFT_TIME['STEPTIME']}")
                print(f"TimeonFoot Left: {_T1_TIME['ENDTIME'] - _T1_LEFT_TIME['STEPTIME']}")
                print(f"TimeonFoot Right: {_T1_TIME['ENDTIME'] - T1_RIGHT_TIME['STEPTIME']}")

                if _T1_TIME['ENDTIME'] <= 8.0:  #define score for time
                    _T1_SCORE['SCORE'] = 100
                elif _T1_TIME['ENDTIME'] > 8.0 and _T1_TIME['ENDTIME'] <= 10.5:
                    _T1_SCORE['SCORE'] = 75
                elif _T1_TIME['ENDTIME'] > 10.5 and _T1_TIME['ENDTIME'] <= 13:
                    _T1_SCORE['SCORE'] = 50
                elif _T1_TIME['ENDTIME'] > 13.0:
                    _T1_SCORE['SCORE'] = 25

                #define score for L/R balance:
                Balance = 100 - abs((((_T1_TIME['ENDTIME'] - _T1_LEFT_TIME['STEPTIME'])/(_T1_TIME['ENDTIME'] -
                    _T1_RIGHT_TIME['STEPTIME']))-1)*100)
                if Balance < 25:
                    Balance = 25

                _T1_SCORE['BALANCE'] = Balance
                _T1_SCORE['SCORE'] = (_T1_SCORE['SCORE'] * 0.75) + (Balance * 0.25) #total T1 score

                print(f"Balance: {Balance}")
                print(f"Score: {_T1_SCORE['SCORE']}")


                with open('MPPTResults.txt', 'a') as f:     #save to text file
                    f.write('\n New Patient!!!!\n')
                    f.write('Test1:')

                    f.write('\n T1_IMU1AngleX:\n')
                    for line in T1_IMU1AngleX:
                        f.write(str(line))
                        f.write(',')
                    f.write('\n T1_IMU1AngleY:\n')
                    for line in T1_IMU1AngleY:
                        f.write(str(line))
```

```
364                         f.write(',')
                        f.write('\n T1_IMU1AngleZ:\n')
                        for line in T1_IMU1AngleZ:
                            f.write(str(line))
                            f.write(',')
369                     f.write('\n T1_IMU1AccX:\n')
                        for line in T1_IMU1AccX:
                            f.write(str(line))
                            f.write(',')
                        f.write('\n T1_IMU1AccY:\n')
374                     for line in T1_IMU1AccY:
                            f.write(str(line))
                            f.write(',')
                        f.write('\n T1_IMU1AccZ:\n')
                        for line in T1_IMU1AccZ:
379                         f.write(str(line))
                            f.write(',')
                        f.write('\n T1_IMU1Time:\n')
                        for line in T1_IMU1Time:
                            f.write(str(line))
384                         f.write(',')
                        f.write('\n T1_IMU2AngleX:\n')
                        for line in T1_IMU2AngleX:
                            f.write(str(line))
                            f.write(',')
389                     f.write('\n T1_IMU2AngleY:\n')
                        for line in T1_IMU2AngleY:
                            f.write(str(line))
                            f.write(',')
                        f.write('\n T1_IMU2AngleZ:\n')
394                     for line in T1_IMU2AngleZ:
                            f.write(str(line))
                            f.write(',')
                        f.write('\n T1_IMU2AccX:\n')
                        for line in T1_IMU2AccX:
399                         f.write(str(line))
                            f.write(',')
                        f.write('\n T1_IMU2AccY:\n')
                        for line in T1_IMU2AccY:
                            f.write(str(line))
404                         f.write(',')
                        f.write('\n T1_IMU2AccZ:\n')
                        for line in T1_IMU2AccZ:
                            f.write(str(line))
                            f.write(',')
409                     f.write('\n T1_IMU2Time:\n')
                        for line in T1_IMU2Time:
                            f.write(str(line))
                            f.write(',')

414                     f.write("\n Total Time " + str(_T1_TIME['ENDTIME']))
                        f.write("\n Balance " + str(Balance))
                        f.write("\n Score " + str(_T1_SCORE['SCORE']))


419


                    print(f"Subtask T2: Walking Stairs")     #go to next task
                    _T2_LEFT_CALIB['CALIB'] = False
424                 _SUBTASK['TASK'] = {'T2'}
                    _NEW['NEW'] = {True}



429


    def T1Rightfoot(msg):         #Subtask1 wearable right foot
        if _T2_LEFT_CALIB['CALIB'] ==True:
            temp = list(map(float, msg.payload.decode().split(",")))
434         if len(temp)==7:
                temp[0], temp[1], temp[2] = rotatequa(temp[0], temp[1], temp[2], temp[3], 0)
                temp[0]=temp[0]-_T1_RIGHT_CALIB['X']
                temp[1]=temp[1]-_T1_RIGHT_CALIB['Y']
                temp[2]=temp[2]-_T1_RIGHT_CALIB['Z']
439


                if _T1_RIGHT_CALIB['X'] == 0.0:
                    _T1_RIGHT_CALIB['X'] = temp[0];
                    _T1_RIGHT_CALIB['Y'] = temp[1];
444                 _T1_RIGHT_CALIB['Z'] = temp[2];
                    _TIME['STARTTIME']=time.time()
```

```
            else:
                T1_IMU2AngleX.append(temp[0])
449             T1_IMU2AngleY.append(temp[1])
                T1_IMU2AngleZ.append(temp[2])

                T1_IMU2AccX.append(temp[4])
                T1_IMU2AccY.append(temp[5])
454             T1_IMU2AccZ.append(temp[6])
                T1_IMU2Time.append(time.time()-_TIME['STARTTIME'])

            #print(f"ACC: {temp[4]}")

459         if (temp[4] > 2.5) and (_T1_RIGHT_STEP ['INSTEP'] == False) and (temp[1] > 10.0):
                #print(f"UP")
                #print(f"Ang: {temp[1]}")
                #print(f"acc: {temp[4]}")
                _T1_RIGHT_STEP ['INSTEP'] = True
464             _T1_RIGHT_TIME['STARTSTEP'] = time.time()
                if (_T1_RIGHT_STEP ['STEP'] == 0) and ( _T1_TIME['STARTTIME'] == 0.0):
                    _T1_TIME['STARTTIME'] = time.time()

            if (temp[4] > -0.4) and (_T1_RIGHT_STEP ['INSTEP'] == True) and (temp[1] < 6.0):
469             #print(f"DOWN")
                _T1_RIGHT_STEP ['INSTEP'] = False
                _T1_RIGHT_STEP ['STEP'] += 1
                print(f"Right Steps: {_T1_RIGHT_STEP ['STEP']}")
                _T1_RIGHT_TIME['STEPTIME'] = _T1_RIGHT_TIME['STEPTIME'] + (time.time() - _T1_RIGHT_TIME['STARTSTEP'
                    ])
474
            if _T1_LEFT_STEP ['STEP'] + _T1_RIGHT_STEP ['STEP'] == 10:
                _T1_TIME['ENDTIME'] = time.time() - _T1_TIME['STARTTIME']

                print()
479             print(f"TotalTime: {_T1_TIME['ENDTIME']}")
                #print(f"StepTime: {_T1_LEFT_TIME['STEPTIME']}")
                print(f"TimeonFoot Left: {_T1_TIME['ENDTIME'] - T1_LEFT_TIME['STEPTIME']}")
                print(f"TimeonFoot Right: {_T1_TIME['ENDTIME'] - T1_RIGHT_TIME['STEPTIME']}")

484             if _T1_TIME['ENDTIME'] <= 8.0:
                    _T1_SCORE['SCORE'] = 100
                elif _T1_TIME['ENDTIME'] > 8.0 and _T1_TIME['ENDTIME'] <= 10.5:
                    _T1_SCORE['SCORE'] = 75
                elif _T1_TIME['ENDTIME'] > 10.5 and _T1_TIME['ENDTIME'] <= 13:
489                 _T1_SCORE['SCORE'] = 50
                elif _T1_TIME['ENDTIME'] > 13.0:
                    _T1_SCORE['SCORE'] = 25


494             Balance = 100 - abs((((_T1_TIME['ENDTIME'] - _T1_LEFT_TIME['STEPTIME'])/(_T1_TIME['ENDTIME'] -
                    _T1_RIGHT_TIME['STEPTIME']))-1)*100)
                if Balance < 25:
                    Balance = 25

                _T1_SCORE['BALANCE'] = Balance
499             _T1_SCORE['SCORE'] = (_T1_SCORE['SCORE'] * 0.75) + (Balance * 0.25)

                print(f"Balance: {Balance}")
                print(f"Score: {_T1_SCORE['SCORE']}")

504
                with open('MPPTResults.txt', 'a') as f:
                    f.write('\n New Patient!!!!\n')
                    f.write('Test1:')

509                 f.write('\n T1_IMU1AngleX:\n')
                    for line in T1_IMU1AngleX:
                        f.write(str(line))
                        f.write(',')
                    f.write('\n T1_IMU1AngleY:\n')
514                 for line in T1_IMU1AngleY:
                        f.write(str(line))
                        f.write(',')
                    f.write('\n T1_IMU1AngleZ:\n')
                    for line in T1_IMU1AngleZ:
519                     f.write(str(line))
                        f.write(',')
                    f.write('\n T1_IMU1AccX:\n')
                    for line in T1_IMU1AccX:
                        f.write(str(line))
524                     f.write(',')
                    f.write('\n T1_IMU1AccY:\n')
```

```
                    for line in T1_IMU1AccY:
                        f.write(str(line))
                        f.write(',')
                f.write('\n T1_IMU1AccZ:\n')
                for line in T1_IMU1AccZ:
                    f.write(str(line))
                    f.write(',')
                f.write('\n T1_IMU1Time:\n')
                for line in T1_IMU1Time:
                    f.write(str(line))
                    f.write(',')
                f.write('\n T1_IMU2AngleX:\n')
                for line in T1_IMU2AngleX:
                    f.write(str(line))
                    f.write(',')
                f.write('\n T1_IMU2AngleY:\n')
                for line in T1_IMU2AngleY:
                    f.write(str(line))
                    f.write(',')
                f.write('\n T1_IMU2AngleZ:\n')
                for line in T1_IMU2AngleZ:
                    f.write(str(line))
                    f.write(',')
                f.write('\n T1_IMU2AccX:\n')
                for line in T1_IMU2AccX:
                    f.write(str(line))
                    f.write(',')
                f.write('\n T1_IMU2AccY:\n')
                for line in T1_IMU2AccY:
                    f.write(str(line))
                    f.write(',')
                f.write('\n T1_IMU2AccZ:\n')
                for line in T1_IMU2AccZ:
                    f.write(str(line))
                    f.write(',')
                f.write('\n T1_IMU2Time:\n')
                for line in T1_IMU2Time:
                    f.write(str(line))
                    f.write(',')

                f.write("\n Total Time " + str(_T1_TIME['ENDTIME']))
                f.write("\n Balance " + str(Balance))
                f.write("\n Score " + str(_T1_SCORE['SCORE']))

            print(f"Subtask T2: Walking Stairs")
            _T2_LEFT_CALIB['CALIB'] = False
            _SUBTASK['TASK'] = {'T2'}
            _NEW['NEW'] = {True}



def T2Leftfoot(msg):    #Subtask2 wearable left foot, calculations similar to T1 with different values
    if _T2_LEFT_CALIB['CALIB'] ==True:
        temp = list(map(float, msg.payload.decode().split(",")))
        if len(temp)==7:
            temp[0], temp[1], temp[2] = rotatequa(temp[0], temp[1], temp[2], temp[3], 0)
            temp[0]=temp[0]-_T2_LEFT_CALIB['X']
            temp[1]=temp[1]-_T2_LEFT_CALIB['Y']
            temp[2]=temp[2]-_T2_LEFT_CALIB['Z']


            if _T2_LEFT_CALIB['X'] == 0.0:
                _T2_LEFT_CALIB['X'] = temp[0];
                _T2_LEFT_CALIB['Y'] = temp[1];
                _T2_LEFT_CALIB['Z'] = temp[2];
                _TIME['STARTTIME']=time.time()

            else:
                T2_IMU1AngleX.append(temp[0])
                T2_IMU1AngleY.append(temp[1])
                T2_IMU1AngleZ.append(temp[2])

                T2_IMU1AccX.append(temp[4])
                T2_IMU1AccY.append(temp[5])
                T2_IMU1AccZ.append(temp[6])
                T2_IMU1Time.append(time.time()-_TIME['STARTTIME'])

            #print(f"Angle: {temp[1]}")

            if (temp[5] > 1.5) and (_T2_LEFT_STEP ['INSTEP'] == False) and (temp[1] < -10.0): #if (temp[4] < -2.5)
```

```
            and (_T2_LEFT_STEP['INSTEP'] == False) and (temp[1] < -10.0):
        print(f"UP")
609     _T2_LEFT_STEP['INSTEP'] = True
        _T2_LEFT_TIME['STARTSTEP'] = time.time()
        if (_T2_LEFT_STEP['STEP'] == 0) and (_T2_TIME['STARTTIME'] == 0.0):
            _T2_TIME['STARTTIME'] = time.time()

614 if (temp[5] < 0.4) and (_T2_LEFT_STEP['INSTEP'] == True) and (temp[1] > -6.0): #if (temp[4] > -0.4)
            and (_T2_LEFT_STEP['INSTEP'] == True) and (temp[1] > -6.0):
        print(f"DOWN")
        _T2_LEFT_STEP['INSTEP'] = False
        _T2_LEFT_STEP['STEP'] += 1
        print(f"Left Steps: {_T2_LEFT_STEP['STEP']}")
619     _T2_LEFT_TIME['STEPTIME'] = _T2_LEFT_TIME['STEPTIME'] + (time.time() - _T2_LEFT_TIME['STARTSTEP'])

    if _T2_LEFT_STEP['STEP'] + _T2_RIGHT_STEP['STEP'] == 4: #if _T2_LEFT_STEP['STEP'] + _T2_RIGHT_STEP
            ['STEP'] == 4: #if _T2_LEFT_STEP['STEP'] == 4:
        _T2_TIME['ENDTIME'] = time.time() - _T2_TIME['STARTTIME']

624     print()
        print(f"TotalTime: {_T2_TIME['ENDTIME']}")
        #print(f"StepTime: {_T2_LEFT_TIME['STEPTIME']}")
        print(f"TimeonFoot Left: {_T2_TIME['ENDTIME'] - T2_LEFT_TIME['STEPTIME']}")
        print(f"TimeonFoot Right: {_T2_TIME['ENDTIME'] - T2_RIGHT_TIME['STEPTIME']}")
629
        if _T2_TIME['ENDTIME'] <= 5.0:
            _T2_SCORE['SCORE'] = 100
        elif _T2_TIME['ENDTIME'] > 5.0 and _T2_TIME['ENDTIME'] <= 10:
            _T2_SCORE['SCORE'] = 75
634     elif _T2_TIME['ENDTIME'] > 10 and _T2_TIME['ENDTIME'] <= 15:
            _T2_SCORE['SCORE'] = 50
        elif _T2_TIME['ENDTIME'] > 15.0:
            _T2_SCORE['SCORE'] = 25

639
        Balance = 100 - abs((((_T2_TIME['ENDTIME'] - _T2_LEFT_TIME['STEPTIME'])/(_T2_TIME['ENDTIME'] -
            _T2_RIGHT_TIME['STEPTIME']))-1)*100)
        if Balance < 25:
            Balance = 25

644     _T2_SCORE['BALANCE'] = Balance
        _T2_SCORE['SCORE'] = (_T2_SCORE['SCORE'] * 0.75) + (Balance * 0.25)

        print(f"Balance: {Balance}")
        print(f"Score: {_T2_SCORE['SCORE']}")
649

        with open('MPPTResults.txt', 'a') as f:
            f.write('\n Test2:')

654         f.write('\n T2_IMU1AngleX:\n')
            for line in T2_IMU1AngleX:
                f.write(str(line))
                f.write(',')
            f.write('\n T2_IMU1AngleY:\n')
659         for line in T2_IMU1AngleY:
                f.write(str(line))
                f.write(',')
            f.write('\n T2_IMU1AngleZ:\n')
            for line in T2_IMU1AngleZ:
664             f.write(str(line))
                f.write(',')
            f.write('\n T2_IMU1AccX:\n')
            for line in T2_IMU1AccX:
                f.write(str(line))
669             f.write(',')
            f.write('\n T2_IMU1AccY:\n')
            for line in T2_IMU1AccY:
                f.write(str(line))
                f.write(',')
674         f.write('\n T2_IMU1AccZ:\n')
            for line in T2_IMU1AccZ:
                f.write(str(line))
                f.write(',')
            f.write('\n T2_IMU1Time:\n')
679         for line in T2_IMU1Time:
                f.write(str(line))
                f.write(',')
            f.write('\n T2_IMU2AngleX:\n')
            for line in T2_IMU2AngleX:
684             f.write(str(line))
                f.write(',')
```

```
                    f.write('\n T2_IMU2AngleY:\n')
                    for line in T2_IMU2AngleY:
                        f.write(str(line))
                        f.write(',')
                    f.write('\n T2_IMU2AngleZ:\n')
                    for line in T2_IMU2AngleZ:
                        f.write(str(line))
                        f.write(',')
                    f.write('\n T2_IMU2AccX:\n')
                    for line in T2_IMU2AccX:
                        f.write(str(line))
                        f.write(',')
                    f.write('\n T2_IMU2AccY:\n')
                    for line in T2_IMU2AccY:
                        f.write(str(line))
                        f.write(',')
                    f.write('\n T2_IMU2AccZ:\n')
                    for line in T2_IMU2AccZ:
                        f.write(str(line))
                        f.write(',')
                    f.write('\n T2_IMU2Time:\n')
                    for line in T2_IMU2Time:
                        f.write(str(line))
                        f.write(',')

                    f.write("\n Total Time " + str(_T2_TIME['ENDTIME']))
                    f.write("\n Balance " + str(Balance))
                    f.write("\n Score " + str(_T2_SCORE['SCORE']))


                print(f"Subtask T3: Turning 360 degree")
                _T2_LEFT_CALIB['CALIB'] = False
                _SUBTASK['TASK'] = {'T3'}
                _NEW['NEW'] = {True}


def T2Rightfoot(msg):        #Subtask2 wearable right foot, calculations similar to T1 with different values
    if _T2_LEFT_CALIB['CALIB'] ==True:
        temp = list(map(float, msg.payload.decode().split(",")))
        if len(temp)==7:
            temp[0], temp[1], temp[2] = rotatequa(temp[0], temp[1], temp[2], temp[3], 0)
            temp[0]=temp[0]-_T2_RIGHT_CALIB['X']
            temp[1]=temp[1]-_T2_RIGHT_CALIB['Y']
            temp[2]=temp[2]-_T2_RIGHT_CALIB['Z']
            #print(f"acc: {temp[4]}")


            if _T2_RIGHT_CALIB['X'] == 0.0:
                _T2_RIGHT_CALIB['X'] = temp[0];
                _T2_RIGHT_CALIB['Y'] = temp[1];
                _T2_RIGHT_CALIB['Z'] = temp[2];
                _TIME['STARTTIME']=time.time()

            else:
                T2_IMU2AngleX.append(temp[0])
                T2_IMU2AngleY.append(temp[1])
                T2_IMU2AngleZ.append(temp[2])

                T2_IMU2AccX.append(temp[4])
                T2_IMU2AccY.append(temp[5])
                T2_IMU2AccZ.append(temp[6])
                T2_IMU2Time.append(time.time()-_TIME['STARTTIME'])

            #print(f"ACC: {temp[4]}")

            if (temp[5] < -1.5) and (_T2_RIGHT_STEP ['INSTEP'] == False) and (temp[1] < -10.0): #if (temp[4] <
                -2.5) and (_T2_LEFT_STEP ['INSTEP'] == False) and (temp[1] < -10.0):
                print(f"UPr")
                #print(f"Ang: {temp[1]}")
                #print(f"acc: {temp[4]}")
                _T2_RIGHT_STEP ['INSTEP'] = True
                _T2_RIGHT_TIME['STARTSTEP'] = time.time()
                if (_T2_RIGHT_STEP ['STEP'] == 0) and ( _T2_TIME['STARTTIME'] == 0.0):
                    _T2_TIME['STARTTIME'] = time.time()

            if (temp[5] > -0.4) and (_T2_RIGHT_STEP ['INSTEP'] == True) and (temp[1] > -6.0): #if (temp[4] > -0.4)
                and (_T2_RIGHT_STEP ['INSTEP'] == True) and (temp[1] > -6.0):
                print(f"DOWN")
                _T2_RIGHT_STEP ['INSTEP'] = False
                _T2_RIGHT_STEP ['STEP'] += 1
                print(f"Right Steps: {_T2_RIGHT_STEP ['STEP']}")
                _T2_RIGHT_TIME['STEPTIME'] = _T2_RIGHT_TIME['STEPTIME'] + (time.time() - _T2_RIGHT_TIME['STARTSTEP'
```

```
            ])

        if _T2_LEFT_STEP['STEP'] + _T2_RIGHT_STEP['STEP'] == 4: #if _T2_LEFT_STEP['STEP'] + _T2_RIGHT_STEP
            ['STEP'] == 4: #if _T2_LEFT_STEP['STEP'] == 4:
            _T2_TIME['ENDTIME'] = time.time() - _T2_TIME['STARTTIME']

            print()
            print(f"TotalTime: {_T2_TIME['ENDTIME']}")
            #print(f"StepTime: {_T2_LEFT_TIME['STEPTIME']}")
            print(f"TimeonFoot Left: {_T2_TIME['ENDTIME'] - T2_LEFT_TIME['STEPTIME']}")
            print(f"TimeonFoot Right: {_T2_TIME['ENDTIME'] - T2_RIGHT_TIME['STEPTIME']}")

            if _T2_TIME['ENDTIME'] <= 5.0:
                _T2_SCORE['SCORE'] = 100
            elif _T2_TIME['ENDTIME'] > 5.0 and _T2_TIME['ENDTIME'] <= 10:
                _T2_SCORE['SCORE'] = 75
            elif _T2_TIME['ENDTIME'] > 10 and _T2_TIME['ENDTIME'] <= 15:
                _T2_SCORE['SCORE'] = 50
            elif _T2_TIME['ENDTIME'] > 15.0:
                _T2_SCORE['SCORE'] = 25

            Balance = 100 - abs((((_T2_TIME['ENDTIME'] - _T2_LEFT_TIME['STEPTIME'])/(_T2_TIME['ENDTIME'] -
                _T2_RIGHT_TIME['STEPTIME']))-1)*100)
            if Balance < 25:
                Balance = 25

            _T2_SCORE['BALANCE'] = Balance
            _T2_SCORE['SCORE'] = (_T2_SCORE['SCORE'] * 0.75) + (Balance * 0.25)

            print(f"Balance: {Balance}")
            print(f"Score: {_T2_SCORE['SCORE']}")

            with open('MPPTResults.txt', 'a') as f:
                f.write('\n Test2:')

                f.write('\n T2_IMU1AngleX:\n')
                for line in T2_IMU1AngleX:
                    f.write(str(line))
                    f.write(',')
                f.write('\n T2_IMU1AngleY:\n')
                for line in T2_IMU1AngleY:
                    f.write(str(line))
                    f.write(',')
                f.write('\n T2_IMU1AngleZ:\n')
                for line in T2_IMU1AngleZ:
                    f.write(str(line))
                    f.write(',')
                f.write('\n T2_IMU1AccX:\n')
                for line in T2_IMU1AccX:
                    f.write(str(line))
                    f.write(',')
                f.write('\n T2_IMU1AccY:\n')
                for line in T2_IMU1AccY:
                    f.write(str(line))
                    f.write(',')
                f.write('\n T2_IMU1AccZ:\n')
                for line in T2_IMU1AccZ:
                    f.write(str(line))
                    f.write(',')
                f.write('\n T2_IMU1Time:\n')
                for line in T2_IMU1Time:
                    f.write(str(line))
                    f.write(',')
                f.write('\n T2_IMU2AngleX:\n')
                for line in T2_IMU2AngleX:
                    f.write(str(line))
                    f.write(',')
                f.write('\n T2_IMU2AngleY:\n')
                for line in T2_IMU2AngleY:
                    f.write(str(line))
                    f.write(',')
                f.write('\n T2_IMU2AngleZ:\n')
                for line in T2_IMU2AngleZ:
                    f.write(str(line))
                    f.write(',')
                f.write('\n T2_IMU2AccX:\n')
                for line in T2_IMU2AccX:
                    f.write(str(line))
                    f.write(',')
                f.write('\n T2_IMU2AccY:\n')
                for line in T2_IMU2AccY:
```

```
                        f.write(str(line))
                        f.write(',')
                    f.write('\n T2_IMU2AccZ:\n')
                    for line in T2_IMU2AccZ:
                        f.write(str(line))
                        f.write(',')
                    f.write('\n T2_IMU2Time:\n')
                    for line in T2_IMU2Time:
                        f.write(str(line))
                        f.write(',')

                    f.write("\n Total Time " + str(_T2_TIME['ENDTIME']))
                    f.write("\n Balance " + str(Balance))
                    f.write("\n Score " + str(_T2_SCORE['SCORE']))


                print(f"Subtask T3: Turning 360 degree")
                _T2_LEFT_CALIB['CALIB'] = False
                _SUBTASK['TASK'] = {'T3'}
                _NEW['NEW'] = {True}

def T3Chest(msg):            #Subtask3 wearable chest
    if _T2_LEFT_CALIB['CALIB'] ==True:
        if _T3_TIME['STARTTIME'] == 0.0:
            _T3_TIME['STARTTIME'] = time.time()
            print(f"_____")

        temp = list(map(float, msg.payload.decode().split(",")))
        if len(temp)==7:
            temp[0], temp[1], temp[2] = rotatequa(temp[0], temp[1], temp[2], temp[3], 1)
            temp[0]=temp[0]-_T3_CHEST_CALIB['X']
            temp[1]=temp[1]-_T3_CHEST_CALIB['Y']
            temp[2]=temp[2]-_T3_CHEST_CALIB['Z']


            if _T3_CHEST_CALIB['X'] == 0.0:
                _T3_CHEST_CALIB['X'] = temp[0];
                _T3_CHEST_CALIB['Y'] = temp[1];
                _T3_CHEST_CALIB['Z'] = temp[2];
                print(f"CalibZ: {temp[2]}")
                _TIME['STARTTIME']=time.time()

            else:
                T3_IMU1AngleX.append(temp[0])
                T3_IMU1AngleY.append(temp[1])
                T3_IMU1AngleZ.append(temp[2])

                T3_IMU1AccX.append(temp[4])
                T3_IMU1AccY.append(temp[5])
                T3_IMU1AccZ.append(temp[6])
                T3_IMU1Time.append(time.time()-_TIME['STARTTIME'])



            print(f"Winkel: {temp[2]}")


            if (abs(temp[0]) > _T3_TURN['X'] and _T3_TURN['MOTION'] == True):    #detect max x angle
                #print(f"_____1")
                _T3_TURN['X']=abs(temp[0])


            if (abs(temp[1]) > _T3_TURN['Y'] and _T3_TURN['MOTION'] == True):    #detect max y angle
                #print(f"_____1")
                _T3_TURN['Y']=abs(temp[1])


            if ((temp[2] > -55) and (temp[2] < -45)) or ((temp[2] > 45) and (temp[2] < 55)):     #detect if turning
                print(f"TurnActive")
                _T3_TURN['MOTION'] = True


            if _T3_TURN['MOTION'] == True and (temp[2] < 3) and (temp[2] > -3):      #detect full turn and save to
                file and print
                print(f"SUCESS")
                _T3_TIME['ENDTIME'] = time.time() - _T3_TIME['STARTTIME']
                print()
                print(f"TotalTime: {_T3_TIME['ENDTIME']}")
                print(f"MaxXAngle: {_T3_TURN['X']}")
                print(f"MaxYAngle: {_T3_TURN['Y']}")
```

```
                if _T3_TIME['ENDTIME'] <= 3.5:
                    _T3_SCORE['SCORE'] = 100
                elif _T3_TIME['ENDTIME'] > 3.5 and _T3_TIME['ENDTIME'] <= 5.5:
929                 _T3_SCORE['SCORE'] = 75
                elif _T3_TIME['ENDTIME'] > 5.5 and _T3_TIME['ENDTIME'] <= 7.5:
                    _T3_SCORE['SCORE'] = 50
                elif _T3_TIME['ENDTIME'] > 7.5:
                    _T3_SCORE['SCORE'] = 25
934
                if ( _T3_TURN['X']+_T3_TURN['Y'])/2 <= 20.0:
                    Balance = 100
                elif ( _T3_TURN['X']+_T3_TURN['Y'])/2 > 20.0 and ( _T3_TURN['X']+_T3_TURN['Y'])/2 <= 30.0:
                    Balance = 75
939             elif ( _T3_TURN['X']+_T3_TURN['Y'])/2 > 30.0 and ( _T3_TURN['X']+_T3_TURN['Y'])/2 <= 40.0:
                    Balance = 50
                elif ( _T3_TURN['X']+_T3_TURN['Y'])/2 > 40.0:
                    Balance = 25

944             _T3_SCORE['BALANCE'] = Balance
                _T3_SCORE['SCORE'] = (_T3_SCORE['SCORE'] * 0.75) + (Balance * 0.25)

                print(f"Balance: {Balance}")
                print(f"Score: {_T3_SCORE['SCORE']}")
949

                with open('MPPTResults.txt', 'a') as f:
                    f.write('\n Test3:')

954                 f.write('\n T3_IMU1AngleX:\n')
                    for line in T3_IMU1AngleX:
                        f.write(str(line))
                        f.write(',')
                    f.write('\n T3_IMU1AngleY:\n')
959                 for line in T3_IMU1AngleY:
                        f.write(str(line))
                        f.write(',')
                    f.write('\n T3_IMU1AngleZ:\n')
                    for line in T3_IMU1AngleZ:
964                     f.write(str(line))
                        f.write(',')
                    f.write('\n T3_IMU1AccX:\n')
                    for line in T3_IMU1AccX:
                        f.write(str(line))
969                     f.write(',')
                    f.write('\n T3_IMU1AccY:\n')
                    for line in T3_IMU1AccY:
                        f.write(str(line))
                        f.write(',')
974                 f.write('\n T3_IMU1AccZ:\n')
                    for line in T3_IMU1AccZ:
                        f.write(str(line))
                        f.write(',')
                    f.write('\n T3_IMU1Time:\n')
979                 for line in T3_IMU1Time:
                        f.write(str(line))
                        f.write(',')


984                 f.write("\n Total Time: " + str(_T3_TIME['ENDTIME']))
                    f.write("\n Max Angle X: " + str(_T3_TURN['X']))
                    f.write("\n Max Angle Y: " + str(_T3_TURN['Y']))
                    f.write("\n Balance: " + str(Balance))
                    f.write("\n Score: " + str(_T3_SCORE['SCORE']))
989

                print(f"Subtask T4: Standing up")
                _T2_LEFT_CALIB['CALIB'] = False
                _SUBTASK['TASK'] = {'T4'}
994             _NEW['NEW'] = {True}


    def T4Rightfoot(msg):        #Subtask4 wearable right side
        if _T2_LEFT_CALIB['CALIB'] ==True:
999         temp = list(map(float, msg.payload.decode().split(",")))
            if len(temp)==7:
                temp[0], temp[1], temp[2] = rotatequa(temp[0], temp[1], temp[2], temp[3], 2)
                temp[0]=temp[0]-_T4_RIGHT_CALIB['X']
                temp[1]=temp[1]-_T4_RIGHT_CALIB['Y']
1004            temp[2]=temp[2]-_T4_RIGHT_CALIB['Z']


                if _T4_RIGHT_CALIB['X'] == 0.0:
```

```
                _T4_RIGHT_CALIB['X'] = temp[0];
1009            _T4_RIGHT_CALIB['Y'] = temp[1];
                _T4_RIGHT_CALIB['Z'] = temp[2];
                _TIME['STARTTIME']=time.time()

            else:
1014            T4_IMU1AngleX.append(temp[0])
                T4_IMU1AngleY.append(temp[1])
                T4_IMU1AngleZ.append(temp[2])

                T4_IMU1AccX.append(temp[4])
1019            T4_IMU1AccY.append(temp[5])
                T4_IMU1AccZ.append(temp[6])
                T4_IMU1Time.append(time.time()-_TIME['STARTTIME'])

            print(f"Angle: {temp[1]}")
1024
            if _T4_TIME['STARTTIME'] == 0.0 and temp[1] > 5:    #start timer
                _T4_TIME['STARTTIME'] = time.time()


1029        if (temp[1] > 40) and (_T4_RIGHT_STEP ['INSTEP'] == False): #detect if patient is fully standing
                print(f"UP")
                #print(f"Ang: {temp[1]}")
                #print(f"acc: {temp[4]}")
                _T4_RIGHT_STEP ['INSTEP'] = True
1034


            if (temp[1] < 10) and (_T4_RIGHT_STEP ['INSTEP'] == True):  #detect if patient is fully sitting
                print(f"DOWN")
1039            _T4_RIGHT_STEP ['INSTEP'] = False
                _T4_RIGHT_STEP ['STEP'] += 1


            if _T4_RIGHT_STEP ['STEP'] == 5:        #stop after five attempts and print/save to file
1044            _T4_TIME['ENDTIME'] = time.time() - _T4_TIME['STARTTIME']

                print()
                print(f"TotalTime: {_T4_TIME['ENDTIME']}")
                #print(f"StepTime: {_T4_LEFT_TIME['STEPTIME']}")
1049
                if _T4_TIME['ENDTIME'] <= 11.0:     #time score
                    _T4_SCORE['SCORE'] = 100
                elif _T4_TIME['ENDTIME'] > 11.0 and _T4_TIME['ENDTIME'] <= 14.0:
                     _T4_SCORE['SCORE'] = 75
1054            elif _T4_TIME['ENDTIME'] > 14.0 and _T4_TIME['ENDTIME'] <= 17.0:
                     _T4_SCORE['SCORE'] = 50
                elif _T4_TIME['ENDTIME'] > 17.0:
                     _T4_SCORE['SCORE'] = 25

1059            if _T4_TILT['X'] <= 20.0:       #balance score
                    Balance = 100
                elif _T4_TILT['X'] > 20.0 and _T4_TILT['X'] <= 30.0:
                    Balance = 75
                elif _T4_TILT['X'] > 30.0 and _T4_TILT['X'] <= 40.0:
1064                Balance = 50
                elif _T4_TILT['X'] > 40.0:
                    Balance = 25

                _T4_SCORE['BALANCE'] = Balance
1069            _T4_SCORE['SCORE'] = (_T4_SCORE['SCORE'] * 0.75) + (Balance * 0.25)

                print(f"MaxXAngle: {_T4_TILT['X']}")
                print(f"Balance: {Balance}")
                print(f"Score: {_T4_SCORE['SCORE']}")
1074

                with open('MPPTResults.txt', 'a') as f:
                    f.write('\n Test4:')

1079                f.write('\n T4_IMU1AngleX:\n')
                    for line in T4_IMU1AngleX:
                        f.write(str(line))
                        f.write(',')
                    f.write('\n T4_IMU1AngleY:\n')
1084                for line in T4_IMU1AngleY:
                        f.write(str(line))
                        f.write(',')
                    f.write('\n T4_IMU1AngleZ:\n')
                    for line in T4_IMU1AngleZ:
1089                    f.write(str(line))
```

```
                    f.write(',')
                f.write('\n T4_IMU1AccX:\n')
                for line in T4_IMU1AccX:
                    f.write(str(line))
                    f.write(',')
                f.write('\n T4_IMU1AccY:\n')
                for line in T4_IMU1AccY:
                    f.write(str(line))
                    f.write(',')
                f.write('\n T4_IMU1AccZ:\n')
                for line in T4_IMU1AccZ:
                    f.write(str(line))
                    f.write(',')
                f.write('\n T4_IMU1Time:\n')
                for line in T4_IMU1Time:
                    f.write(str(line))
                    f.write(',')
                f.write('\n T4_IMU2AngleX:\n')
                for line in T4_IMU2AngleX:
                    f.write(str(line))
                    f.write(',')
                f.write('\n T4_IMU2AngleY:\n')
                for line in T4_IMU2AngleY:
                    f.write(str(line))
                    f.write(',')
                f.write('\n T4_IMU2AngleZ:\n')
                for line in T4_IMU2AngleZ:
                    f.write(str(line))
                    f.write(',')
                f.write('\n T4_IMU2AccX:\n')
                for line in T4_IMU2AccX:
                    f.write(str(line))
                    f.write(',')
                f.write('\n T4_IMU2AccY:\n')
                for line in T4_IMU2AccY:
                    f.write(str(line))
                    f.write(',')
                f.write('\n T4_IMU2AccZ:\n')
                for line in T4_IMU2AccZ:
                    f.write(str(line))
                    f.write(',')
                f.write('\n T4_IMU2Time:\n')
                for line in T4_IMU2Time:
                    f.write(str(line))
                    f.write(',')


                f.write("\n Total Time: " + str(_T4_TIME['ENDTIME']))
                f.write("\n Max Angle X: " + str(_T4_TILT['X']))
                f.write("\n Balance: " + str(Balance))
                f.write("\n Score: " + str(_T4_SCORE['SCORE']))


            print(f"Subtask T5: Picking up object while sitting")
            _T2_LEFT_CALIB['CALIB'] = False
            _SUBTASK['TASK'] = {'T5'}
            _NEW['NEW'] = {True}

def T4Chest(msg):        #Subtask4 chest module
    if _T2_LEFT_CALIB['CALIB'] ==True:
        if _T4_TIME['STARTTIME'] == 0.0:
            _T4_TIME['STARTTIME'] = time.time()
            print(f"_____")

        temp = list(map(float, msg.payload.decode().split(",")))
        if len(temp)==7:
            temp[0], temp[1], temp[2] = rotatequa(temp[0], temp[1], temp[2], temp[3], 1)
            temp[0]=temp[0]-_T4_CHEST_CALIB['X']
            temp[1]=temp[1]-_T4_CHEST_CALIB['Y']
            temp[2]=temp[2]-_T4_CHEST_CALIB['Z']


            if _T4_CHEST_CALIB['X'] == 0.0:
                _T4_CHEST_CALIB['X'] = temp[0];
                _T4_CHEST_CALIB['Y'] = temp[1];
                _T4_CHEST_CALIB['Z'] = temp[2];
                print(f"CalibZ: {temp[2]}")
                _TIME['STARTTIME']=time.time()

            else:
                T4_IMU2AngleX.append(temp[0])
                T4_IMU2AngleY.append(temp[1])
```

```
                T4_IMU2AngleZ.append(temp[2])

1174            T4_IMU2AccX.append(temp[4])
                T4_IMU2AccY.append(temp[5])
                T4_IMU2AccZ.append(temp[6])
                T4_IMU2Time.append(time.time()-_TIME['STARTTIME'])

1179


            #print(f"Winkel: {temp[0]}")


1184        if (abs(temp[0]) > _T4_TILT['X']):      #detect max x angle chest
                #print(f"_____________________1")
                _T4_TILT['X']=abs(temp[0])


1189


def T5Object(msg):      #Subtask5 smart object
    if _T2_LEFT_CALIB['CALIB'] ==True:
1194        temp = list(map(float, msg.payload.decode().split(",")))
            if len(temp)==7:
                temp[0], temp[1], temp[2] = rotatequa(temp[0], temp[1], temp[2], temp[3], 2)
                temp[0]=temp[0]-_T5_OBJECT_CALIB['X']
                temp[1]=temp[1]-_T5_OBJECT_CALIB['Y']
1199            temp[2]=temp[2]-_T5_OBJECT_CALIB['Z']


                if _T5_OBJECT_CALIB['X'] == 0.0:
                    _T5_OBJECT_CALIB['X'] = temp[0];
1204                _T5_OBJECT_CALIB['Y'] = temp[1];
                    _T5_OBJECT_CALIB['Z'] = temp[2];
                    _TIME['STARTTIME']=time.time()

                else:
1209                T5_IMU1AngleX.append(temp[0])
                    T5_IMU1AngleY.append(temp[1])
                    T5_IMU1AngleZ.append(temp[2])

                    T5_IMU1AccX.append(temp[4])
1214                T5_IMU1AccY.append(temp[5])
                    T5_IMU1AccZ.append(temp[6])
                    T5_IMU1Time.append(time.time()-_TIME['STARTTIME'])

                print(f"ACC:␣{temp[4]},␣{temp[5]},␣{temp[6]}")
1219

                if (abs(temp[4])>0.5 or abs(temp[5])>0.5 or abs(temp[6])>0.5) and _T5_OBJECT_STEP['INSTEP'] == False:
                        #detect movement start object
                    print("Start")
                    _T5_TIME['STARTTIME'] =time.time()
1224                _T5_OBJECT_STEP['INSTEP'] = True

                if _T5_OBJECT_STEP['INSTEP'] == True:       #calc mean acceleration while move
                    _T5_ACC['MEANACC'] = _T5_ACC['MEANACC']+abs(temp[4])+abs(temp[5])
                    _T5_ACC['COUNT'] = _T5_ACC['COUNT'] + 2
1229

                if abs(temp[4])<0.15 and abs(temp[5])<0.15 and abs(temp[6])<0.20 and _T5_OBJECT_STEP['INSTEP'] == True:
                       #detect test ends, object stops
                    print("Stop")
                    #_T5_OBJECT_STEP ['INSTEP'] = False
1234                _T5_TIME['ENDTIME'] = time.time() - _T5_TIME['STARTTIME']

                    print()
                    print(f"TotalTime:␣{_T5_TIME['ENDTIME']}")
                    print(f"Mean␣Acceleration:␣{_T5_ACC['MEANACC']/_T5_ACC['COUNT']}")
1239

                    if _T5_TIME['ENDTIME'] <= 2.0:
                        _T5_SCORE['SCORE'] = 100
                    elif _T5_TIME['ENDTIME'] > 2.0 and _T5_TIME['ENDTIME'] <= 4.0:
1244                    _T5_SCORE['SCORE'] = 75
                    elif _T5_TIME['ENDTIME'] > 4.0 and _T5_TIME['ENDTIME'] <= 6.0:
                        _T5_SCORE['SCORE'] = 50
                    elif _T5_TIME['ENDTIME'] > 6.0:
                        _T5_SCORE['SCORE'] = 25
1249
                    if _T5_ACC['MEANACC']/_T5_ACC['COUNT'] <= 2.0:
                        Balance = 100
```

```
                elif _T5_ACC['MEANACC']/_T5_ACC['COUNT'] > 2.0 and _T5_ACC['MEANACC']/_T5_ACC['COUNT'] <= 2.5:
                    Balance = 75
                elif _T5_ACC['MEANACC']/_T5_ACC['COUNT'] > 2.5 and _T5_ACC['MEANACC']/_T5_ACC['COUNT'] <= 3.0:
                    Balance = 50
                elif _T5_ACC['MEANACC']/_T5_ACC['COUNT'] > 3.0:
                    Balance = 25

                _T5_SCORE['BALANCE'] = Balance
                _T5_SCORE['SCORE'] = (_T5_SCORE['SCORE'] * 0.75) + (Balance * 0.25)

                print(f"Balance: {Balance}")
                print(f"Score: {_T5_SCORE['SCORE']}")


                with open('MPPTResults.txt', 'a') as f:
                    f.write('\n Test5:')

                    f.write('\n T5_IMU1AngleX:\n')
                    for line in T5_IMU1AngleX:
                        f.write(str(line))
                        f.write(',')
                    f.write('\n T5_IMU1AngleY:\n')
                    for line in T5_IMU1AngleY:
                        f.write(str(line))
                        f.write(',')
                    f.write('\n T5_IMU1AngleZ:\n')
                    for line in T5_IMU1AngleZ:
                        f.write(str(line))
                        f.write(',')
                    f.write('\n T5_IMU1AccX:\n')
                    for line in T5_IMU1AccX:
                        f.write(str(line))
                        f.write(',')
                    f.write('\n T5_IMU1AccY:\n')
                    for line in T5_IMU1AccY:
                        f.write(str(line))
                        f.write(',')
                    f.write('\n T5_IMU1AccZ:\n')
                    for line in T5_IMU1AccZ:
                        f.write(str(line))
                        f.write(',')
                    f.write('\n T5_IMU1Time:\n')
                    for line in T5_IMU1Time:
                        f.write(str(line))
                        f.write(',')


                    f.write("\n Total Time: " + str(_T5_TIME['ENDTIME']))
                    f.write("\n Mean Acc: " + str(_T5_ACC['MEANACC']/_T5_ACC['COUNT']))
                    f.write("\n Balance: " + str(Balance))
                    f.write("\n Score: " + str(_T5_SCORE['SCORE']))


                print(f"Subtask T6: Picking up object while standing")
                _T2_LEFT_CALIB['CALIB'] = False
                _SUBTASK['TASK'] = {'T6'}
                _NEW['NEW'] = {True}


def T6Object(msg):      #Subtask6, same calculations as subtask5
    if _T2_LEFT_CALIB['CALIB'] ==True:
        temp = list(map(float, msg.payload.decode().split(",")))
        if len(temp)==7:
            temp[0], temp[1], temp[2] = rotatequa(temp[0], temp[1], temp[2], temp[3], 2)
            temp[0]=temp[0]-_T6_OBJECT_CALIB['X']
            temp[1]=temp[1]-_T6_OBJECT_CALIB['Y']
            temp[2]=temp[2]-_T6_OBJECT_CALIB['Z']


            if _T6_OBJECT_CALIB['X'] == 0.0:
                _T6_OBJECT_CALIB['X'] = temp[0];
                _T6_OBJECT_CALIB['Y'] = temp[1];
                _T6_OBJECT_CALIB['Z'] = temp[2];
                _TIME['STARTTIME']=time.time()

            else:
                T6_IMU1AngleX.append(temp[0])
                T6_IMU1AngleY.append(temp[1])
                T6_IMU1AngleZ.append(temp[2])

                T6_IMU1AccX.append(temp[4])
                T6_IMU1AccY.append(temp[5])
```

```
1334                T6_IMU1AccZ.append(temp[6])
                    T6_IMU1Time.append(time.time()-_TIME['STARTTIME'])

                print(f"ACC: {temp[4]}, {temp[5]}, {temp[6]}")

1339
                if (abs(temp[4])>0.5 or abs(temp[5])>0.5 or abs(temp[6])>0.5) and _T6_OBJECT_STEP['INSTEP'] == False:
                    print("Start")
                    _T6_TIME['STARTTIME'] =time.time()
                    _T6_OBJECT_STEP['INSTEP'] = True
1344
                if _T6_OBJECT_STEP['INSTEP'] == True:
                    _T6_ACC['MEANACC'] = _T6_ACC['MEANACC']+abs(temp[4])+abs(temp[5])
                    _T6_ACC['COUNT'] = _T6_ACC['COUNT'] + 2

1349
                if abs(temp[4])<0.15 and abs(temp[5])<0.15 and abs(temp[6])<0.20 and _T6_OBJECT_STEP['INSTEP'] == True:
                    print("Stop")
                    #_T6_OBJECT_STEP ['INSTEP'] = False
                    _T6_TIME['ENDTIME'] = time.time() - _T6_TIME['STARTTIME']
1354
                    print()
                    print(f"TotalTime: {_T6_TIME['ENDTIME']}")
                    print(f"Mean Acceleration: {_T6_ACC['MEANACC']/_T6_ACC['COUNT']}")

1359                if _T6_TIME['ENDTIME'] <= 2.0:
                        _T6_SCORE['SCORE'] = 100
                    elif _T6_TIME['ENDTIME'] > 2.0 and _T6_TIME['ENDTIME'] <= 4.0:
                         _T6_SCORE['SCORE'] = 75
                    elif _T6_TIME['ENDTIME'] > 4.0 and _T6_TIME['ENDTIME'] <= 6.0:
1364                     _T6_SCORE['SCORE'] = 50
                    elif _T6_TIME['ENDTIME'] > 6.0:
                         _T6_SCORE['SCORE'] = 25

                    if _T6_ACC['MEANACC']/_T6_ACC['COUNT'] <= 2.0:
1369                    Balance = 100
                    elif _T6_ACC['MEANACC']/_T6_ACC['COUNT'] > 2.0 and _T6_ACC['MEANACC']/_T6_ACC['COUNT'] <= 2.5:
                        Balance = 75
                    elif _T6_ACC['MEANACC']/_T6_ACC['COUNT'] > 2.5 and _T6_ACC['MEANACC']/_T6_ACC['COUNT'] <= 3.0:
                        Balance = 50
1374                elif _T6_ACC['MEANACC']/_T6_ACC['COUNT'] > 3.0:
                        Balance = 25

                    _T6_SCORE['BALANCE'] = Balance
                    _T6_SCORE['SCORE'] = (_T6_SCORE['SCORE'] * 0.75) + (Balance * 0.25)
1379
                    print(f"Balance: {Balance}")
                    print(f"Score: {_T6_SCORE['SCORE']}")


1384                with open('MPPTResults.txt', 'a') as f:
                        f.write('\n Test6:')

                        f.write('\n T6_IMU1AngleX:\n')
                        for line in T6_IMU1AngleX:
1389                        f.write(str(line))
                            f.write(',')
                        f.write('\n T6_IMU1AngleY:\n')
                        for line in T6_IMU1AngleY:
                            f.write(str(line))
1394                        f.write(',')
                        f.write('\n T6_IMU1AngleZ:\n')
                        for line in T6_IMU1AngleZ:
                            f.write(str(line))
                            f.write(',')
1399                    f.write('\n T6_IMU1AccX:\n')
                        for line in T6_IMU1AccX:
                            f.write(str(line))
                            f.write(',')
                        f.write('\n T6_IMU1AccY:\n')
1404                    for line in T6_IMU1AccY:
                            f.write(str(line))
                            f.write(',')
                        f.write('\n T6_IMU1AccZ:\n')
                        for line in T6_IMU1AccZ:
1409                        f.write(str(line))
                            f.write(',')
                        f.write('\n T6_IMU1Time:\n')
                        for line in T6_IMU1Time:
                            f.write(str(line))
1414                        f.write(',')
```

```
                    f.write("\n Total Time: " + str(_T6_TIME['ENDTIME']))
                    f.write("\n Mean Acc: " + str(_T6_ACC['MEANACC']/_T6_ACC['COUNT']))
                    f.write("\n Balance: " + str(Balance))
                    f.write("\n Score: " + str(_T6_SCORE['SCORE']))


                print(f"Subtask T7: Balance, feet together next to each other")
                _T2_LEFT_CALIB['CALIB'] = False
                _SUBTASK['TASK'] = {'T7'}
                _NEW['NEW'] = {True}


def T7Balance(msg):          #Subtask7 Balance Board
    if _T2_LEFT_CALIB['CALIB'] ==True:
        if _T7_TIME['STARTTIME'] == 0.0:
            _T7_TIME['STARTTIME'] =time.time()
        temp = list(map(float, msg.payload.decode().split(",")))
        if len(temp)==4:

            T7_Sensor1.append(temp[0])
            T7_Sensor2.append(temp[1])
            T7_Sensor3.append(temp[2])
            T7_Sensor4.append(temp[3])


            T7_Time.append(time.time()-_T7_TIME['STARTTIME'])

            _T7_TIME['ENDTIME'] = time.time() - _T7_TIME['STARTTIME']

            _T7_BALANCE['LR'] = (temp[0]+temp[1])/(temp[2]+temp[3])      #Balance ratios calc
            _T7_BALANCE['FB'] = (temp[0]+temp[2])/(temp[1]+temp[3])


            _T7_MEAN['LR'] = _T7_MEAN['LR']+((temp[0]+temp[1])/(temp[2]+temp[3]))    #mean value balance calc
            _T7_MEAN['FB'] = _T7_MEAN['FB']+((temp[0]+temp[2])/(temp[1]+temp[3]))

            _T7_MEAN['COUNT'] = _T7_MEAN['COUNT'] + 1

            if _T7_TIME['ENDTIME']>10.0:          #stop after 10 seconds
                print()
                print(f"Mean L/R Balance: {_T7_MEAN['LR']/_T7_MEAN['COUNT']}")
                print(f"Mean F/B Balance: {_T7_MEAN['FB']/_T7_MEAN['COUNT']}")


                Balance = ((_T7_MEAN['LR']/_T7_MEAN['COUNT']) + (_T7_MEAN['FB']/_T7_MEAN['COUNT']))/2


                _T7_SCORE['SCORE'] = 100 - abs((Balance-1)*100)
                if _T7_SCORE['SCORE'] < 25:
                    _T7_SCORE['SCORE'] = 25

                print(f"Balance: {Balance}")
                print(f"Score: {_T7_SCORE['SCORE']}")


                with open('MPPTResults.txt', 'a') as f:
                    f.write('\n Test7:')

                    f.write('\n T7_Sensor1:\n')
                    for line in T7_Sensor1:
                        f.write(str(line))
                        f.write(',')
                    f.write('\n T7_Sensor2:\n')
                    for line in T7_Sensor2:
                        f.write(str(line))
                        f.write(',')
                    f.write('\n T7_Sensor3:\n')
                    for line in T7_Sensor3:
                        f.write(str(line))
                        f.write(',')
                    f.write('\n T7_Sensor4:\n')
                    for line in T7_Sensor4:
                        f.write(str(line))
                        f.write(',')
                    f.write('\n T7_Time:\n')
                    for line in T7_Time:
                        f.write(str(line))
                        f.write(',')


                    f.write("\n Mean L/R Balance: " + str(_T7_MEAN['LR']/_T7_MEAN['COUNT']))
```

```
                    f.write("\n Mean F/B Balance: " + str(_T7_MEAN['FB']/_T7_MEAN['COUNT']))
                    f.write("\n Balance: " + str(Balance))
                    f.write("\n Score: " + str(_T7_SCORE['SCORE']))

                print(f"Subtask T8: Balance, tandem")
                _T2_LEFT_CALIB['CALIB'] = False
                _SUBTASK['TASK'] = {'T8'}
                _NEW['NEW'] = {True}


def T8Balance(msg):         #Subtask8 Balance Board, similar to subtask7
    if _T2_LEFT_CALIB['CALIB'] ==True:
        if _T8_TIME['STARTTIME'] == 0.0:
            _T8_TIME['STARTTIME'] =time.time()
        temp = list(map(float, msg.payload.decode().split(",")))
        if len(temp)==4:

            T8_Sensor1.append(temp[0])
            T8_Sensor2.append(temp[1])
            T8_Sensor3.append(temp[2])
            T8_Sensor4.append(temp[3])
            T8_Time.append(time.time()-_T8_TIME['STARTTIME'])

            _T8_TIME['ENDTIME'] = time.time() - _T8_TIME['STARTTIME']

            _T8_BALANCE['LR'] = (temp[0]+temp[1])/(temp[2]+temp[3])

            _T8_MEAN['LR'] = _T8_MEAN['LR']+((temp[0]+temp[1])/(temp[2]+temp[3]))

            _T8_MEAN['COUNT'] = _T8_MEAN['COUNT'] + 1

            if _T8_TIME['ENDTIME']>10.0:
                print()
                print(f"Mean L/R Balance: {_T8_MEAN['LR']/_T8_MEAN['COUNT']}")

                Balance = _T8_MEAN['LR']/_T8_MEAN['COUNT']


                _T8_SCORE['SCORE'] = 100 - abs((Balance-1)*100)
                if _T8_SCORE['SCORE'] < 25:
                    _T8_SCORE['SCORE'] = 25

                _SCORE['SCORE'] = _T1_SCORE['SCORE'] + _T2_SCORE['SCORE'] + _T3_SCORE['SCORE'] + _T5_SCORE['SCORE']
                       + _T5_SCORE['SCORE'] + _T6_SCORE['SCORE'] + ((_T7_SCORE['SCORE'] + _T8_SCORE['SCORE'])/2)
                _SCORE['SCORE'] = _SCORE['SCORE']/7

                print(f"Balance: {Balance}")
                print(f"Score: {_T8_SCORE['SCORE']}")
                print(f"TotalScore: {_SCORE['SCORE']}")



                with open('MPPTResults.txt', 'a') as f:
                    f.write('\n Test8:')

                    f.write('\n T8_Sensor1:\n')
                    for line in T8_Sensor1:
                        f.write(str(line))
                        f.write(',')
                    f.write('\n T8_Sensor2:\n')
                    for line in T8_Sensor2:
                        f.write(str(line))
                        f.write(',')
                    f.write('\n T8_Sensor3:\n')
                    for line in T8_Sensor3:
                        f.write(str(line))
                        f.write(',')
                    f.write('\n T8_Sensor4:\n')
                    for line in T8_Sensor4:
                        f.write(str(line))
                        f.write(',')
                    f.write('\n T8_Time:\n')
                    for line in T8_Time:
                        f.write(str(line))
                        f.write(',')

                    f.write("\n Mean L/R Balance: " + str(_T8_MEAN['LR']/_T8_MEAN['COUNT']))
                    f.write("\n Balance: " + str(Balance))
                    f.write("\n Score: " + str(_T8_SCORE['SCORE']))
                    f.write("\n\n Total MPPT Score: " + str(_SCORE['SCORE']))
```

```
1579
                input("End of Physical Performance Test")


1584 def run():       #main programm
        client = connect_mqtt()      #connect to mqtt
        print(f"Subtask T1: Walking Test")
        calibration()   #start with calibration
        subscribe(client)
1589    _SUBTASK['TASK'] = {'T1'}     #1walk, 2stair, 3turn, 4sit, 5pick1, 6pick2, 7balance1, 8balance2
        client.loop_start()


        window = tk.Tk()     #create GUI
1594    window.attributes('-fullscreen', True)  #open in fullscreen
        window.title("Digital MPPT")

        window.rowconfigure([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], minsize=50, weight=1)   #define grid for textboxes and
            labels
        window.columnconfigure([0, 1, 2, 3], minsize=50, weight=1)                  #define grid element size
1599

        label1 = tk.Label(master=window, text="test", bg="yellow")      #create elements in grid
        label1.grid(row=0, column=0)
        btn_decrease = tk.Button(master=window, text="Calibration", command=calibration)
1604    btn_decrease.grid(row=0, column=1, sticky="nsew")
        frame1 = tk.Frame(master=window, width=100, height=100, bg="blue")
        frame1.grid(row=0, column=2)
        label2 = tk.Label(master=window, text="test", bg="lightgrey")
        label2.grid(row=0, column=3)
1609
        lbl_t1_name = tk.Label(master=window, text="T1: Walk 10 steps", bg="white") #define text for subtasks in column
            0
        lbl_t1_name.grid(row=1, column=0)
        lbl_t2_name = tk.Label(master=window, text="T2: Walk 4 stair steps", bg="white")
        lbl_t2_name.grid(row=2, column=0)
1614    lbl_t3_name = tk.Label(master=window, text="T3: Turn 360 degree", bg="white")
        lbl_t3_name.grid(row=3, column=0)
        lbl_t4_name = tk.Label(master=window, text="T4: Sit to Stand 5 times", bg="white")
        lbl_t4_name.grid(row=4, column=0)
        lbl_t5_name = tk.Label(master=window, text="T5: Pick up object while sitting", bg="white")
1619    lbl_t5_name.grid(row=5, column=0)
        lbl_t6_name = tk.Label(master=window, text="T6: Pick up object while standing", bg="white")
        lbl_t6_name.grid(row=6, column=0)
        lbl_t7_name = tk.Label(master=window, text="T7: Balance 10 seconds feet together", bg="white")
        lbl_t7_name.grid(row=7, column=0)
1624    lbl_t8_name = tk.Label(master=window, text="T8: Balance 10 seconds tandem", bg="white")
        lbl_t8_name.grid(row=8, column=0)
        lbl_t_name = tk.Label(master=window, text="Total Test", bg="grey")
        lbl_t_name.grid(row=9, column=0)

1629
        lbl_T1_1 = tk.Label(master=window, text="\n Time: " + str(_T1_TIME['ENDTIME']))     #define time and score for
            each subtask in GUI
        lbl_T1_1.grid(row=1, column=1)
        lbl_T1_2 = tk.Label(master=window, text="\n Balance: " + str(_T1_SCORE['BALANCE']))
        lbl_T1_2.grid(row=1, column=2)
1634    lbl_T1_3 = tk.Label(master=window, text="\n Score: " + str(_T1_SCORE['SCORE']))
        lbl_T1_3.grid(row=1, column=3)

        lbl_T2_1 = tk.Label(master=window, text="\n Time: " + str(_T2_TIME['ENDTIME']))
        lbl_T2_1.grid(row=2, column=1)
1639    lbl_T2_2 = tk.Label(master=window, text="\n Balance: " + str(_T2_SCORE['BALANCE']))
        lbl_T2_2.grid(row=2, column=2)
        lbl_T2_3 = tk.Label(master=window, text="\n Score: " + str(_T2_SCORE['SCORE']))
        lbl_T2_3.grid(row=2, column=3)

1644    lbl_T3_1 = tk.Label(master=window, text="\n Time: " + str(_T3_TIME['ENDTIME']))
        lbl_T3_1.grid(row=3, column=1)
        lbl_T3_2 = tk.Label(master=window, text="\n Balance: " + str(_T3_SCORE['BALANCE']))
        lbl_T3_2.grid(row=3, column=2)
        lbl_T3_3 = tk.Label(master=window, text="\n Score: " + str(_T3_SCORE['SCORE']))
1649    lbl_T3_3.grid(row=3, column=3)

        lbl_T4_1 = tk.Label(master=window, text="\n Time: " + str(_T4_TIME['ENDTIME']))
        lbl_T4_1.grid(row=4, column=1)
        lbl_T4_2 = tk.Label(master=window, text="\n Balance: " + str(_T4_SCORE['BALANCE']))
1654    lbl_T4_2.grid(row=4, column=2)
        lbl_T4_3 = tk.Label(master=window, text="\n Score: " + str(_T4_SCORE['SCORE']))
        lbl_T4_3.grid(row=4, column=3)
```

```
        lbl_T5_1 = tk.Label(master=window, text="\n Time: " + str(_T5_TIME['ENDTIME']))
1659    lbl_T5_1.grid(row=5, column=1)
        lbl_T5_2 = tk.Label(master=window, text="\n Balance: " + str(_T5_SCORE['BALANCE']))
        lbl_T5_2.grid(row=5, column=2)
        lbl_T5_3 = tk.Label(master=window, text="\n Score: " + str(_T5_SCORE['SCORE']))
        lbl_T5_3.grid(row=5, column=3)
1664
        lbl_T6_1 = tk.Label(master=window, text="\n Time: " + str(_T6_TIME['ENDTIME']))
        lbl_T6_1.grid(row=6, column=1)
        lbl_T6_2 = tk.Label(master=window, text="\n Balance: " + str(_T6_SCORE['BALANCE']))
        lbl_T6_2.grid(row=6, column=2)
1669    lbl_T6_3 = tk.Label(master=window, text="\n Score: " + str(_T6_SCORE['SCORE']))
        lbl_T6_3.grid(row=6, column=3)

        lbl_T7_1 = tk.Label(master=window, text="\n L/R Balance: " + str(_T7_BALANCE['LR']))
        lbl_T7_1.grid(row=7, column=1)
1674    lbl_T7_2 = tk.Label(master=window, text="\n F/B Balance: " + str(_T7_BALANCE['FB']))
        lbl_T7_2.grid(row=7, column=2)
        lbl_T7_3 = tk.Label(master=window, text="\n Score: " + str(_T7_SCORE['SCORE']))
        lbl_T7_3.grid(row=7, column=3)

1679    lbl_T8_1 = tk.Label(master=window, text="\n L/R Balance: " + str(_T8_BALANCE['LR']))
        lbl_T8_1.grid(row=8, column=1)
        lbl_T8_2 = tk.Label(master=window, text="\n Score: " + str(_T8_SCORE['SCORE']))
        lbl_T8_2.grid(row=8, column=2)

1684    lbl_T_1 = tk.Label(master=window, text="\n Score: " + str(_SCORE['SCORE']))
        lbl_T_1.grid(row=9, column=1)

        while True:     #refresh while test running

1689        if _T2_LEFT_CALIB['CALIB'] == False:        #define color of calib element and current subtask label
                frame1.config(background='orange')
                label2.config(text="Not Calibrated")
            if _T2_LEFT_CALIB['CALIB'] == True:
                frame1.config(background='green')
1694            label2.config(text="Calibrated")
            if _SUBTASK['TASK'] == {'T1'}:
                label1["text"] = "Current Task: T1"
            if _SUBTASK['TASK'] == {'T2'}:
                label1["text"] = "Current Task: T2"
1699        if _SUBTASK['TASK'] == {'T3'}:
                label1["text"] = "Current Task: T3"
            if _SUBTASK['TASK'] == {'T4'}:
                label1["text"] = "Current Task: T4"
            if _SUBTASK['TASK'] == {'T5'}:
1704            label1["text"] = "Current Task: T5"
            if _SUBTASK['TASK'] == {'T6'}:
                label1["text"] = "Current Task: T6"
            if _SUBTASK['TASK'] == {'T7'}:
                label1["text"] = "Current Task: T7"
1709        if _SUBTASK['TASK'] == {'T8'}:
                label1["text"] = "Current Task: T8"

            lbl_T1_1["text"] = "Time: " + str(_T1_TIME['ENDTIME'])      #refresh the time and scores for each test
            lbl_T1_2["text"] = "Balance: " + str(_T1_SCORE['BALANCE'])
1714        lbl_T1_3["text"] = "Score: " + str(_T1_SCORE['SCORE'])

            lbl_T2_1["text"] = "Time: " + str(_T2_TIME['ENDTIME'])
            lbl_T2_2["text"] = "Balance: " + str(_T2_SCORE['BALANCE'])
            lbl_T2_3["text"] = "Score: " + str(_T2_SCORE['SCORE'])
1719
            lbl_T3_1["text"] = "Time: " + str(_T3_TIME['ENDTIME'])
            lbl_T3_2["text"] = "Balance: " + str(_T3_SCORE['BALANCE'])
            lbl_T3_3["text"] = "Score: " + str(_T3_SCORE['SCORE'])

1724        lbl_T4_1["text"] = "Time: " + str(_T4_TIME['ENDTIME'])
            lbl_T4_2["text"] = "Balance: " + str(_T4_SCORE['BALANCE'])
            lbl_T4_3["text"] = "Score: " + str(_T4_SCORE['SCORE'])

            lbl_T5_1["text"] = "Time: " + str(_T5_TIME['ENDTIME'])
1729        lbl_T5_2["text"] = "Balance: " + str(_T5_SCORE['BALANCE'])
            lbl_T5_3["text"] = "Score: " + str(_T5_SCORE['SCORE'])

            lbl_T6_1["text"] = "Time: " + str(_T6_TIME['ENDTIME'])
            lbl_T6_2["text"] = "Balance: " + str(_T6_SCORE['BALANCE'])
1734        lbl_T6_3["text"] = "Score: " + str(_T6_SCORE['SCORE'])

            lbl_T7_1["text"] = "L/R Balance: " + str(_T7_BALANCE['LR'])
            lbl_T7_2["text"] = "F/B Balance: " + str(_T7_BALANCE['FB'])
            lbl_T7_3["text"] = "Score: " + str(_T7_SCORE['SCORE'])
1739
```

```
            lbl_T8_1["text"] = "L/R_Balance:␣" + str(_T8_BALANCE['LR'])
            lbl_T8_2["text"] = "Score:␣" + str(_T8_SCORE['SCORE'])

            lbl_T_1["text"] = "Score:␣" + str(_SCORE['SCORE'])
1744
            window.update()

            if _NEW['NEW'] == {True}:       #do between each subtask, used to not get values in buffer for next subtask
                client.loop_stop()
1749            client.reinitialise()
                client = connect_mqtt()
                subscribe(client)
                client.loop_start()
                _NEW['NEW'] = {False}
1754

if __name__ == '__main__':
    run()
```

A.4. Game FollowCar

```
using System.Collections;
using System.Collections.Generic;
3 using UnityEngine;

public class FollowCar : MonoBehaviour
{
    public GameObject player;
8

    // Start is called before the first frame update
    void Start()
    {
13
    }

    // Update is called once per frame
    void LateUpdate()
18  {
        transform.position = player.transform.position + new Vector3(0, 5, -7);
    }
}
```

A.5. Game GameOverScreen

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;
4 using UnityEngine.SceneManagement;

public class GameOverScreen : MonoBehaviour
{
    public Text pointsText;
9
    public  void Setup(int score)
    {
        gameObject.SetActive(true);
        //pointsText.text = score.ToString() + "POINTS";
14  }

    public void RestartButton ()
    {
        SceneManager.LoadScene("Game");
19  }

    public void ExitButton()
    {
        SceneManager.LoadScene("MainMenu");
24  }


}
```

A.6. Game MainMenu

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class MainMenu : MonoBehaviour
{
    public void ExitButton()
    {
        Application.Quit();
        Debug.Log("Game closed");
    }

    public void StartGame()
    {
        SceneManager.LoadScene("Game");
    }
}
```

A.7. Game mqttController

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Globalization;
using UnityEngine;
using UnityEngine.GridBrushBase;

public class mqttController : MonoBehaviour //Used for getting the IMU values from the MQTT receiver and forwarding
      them to the player controller
{
    [SerializeField]
    GameObject logicController;      //used to link receiver to controller

    public string nameController = "Controller 1";
    public string tagOfTheMQTTReceiver = "";
    public mqttReceiver _eventSender; //used to access variables in mqttReceiver

    public float xe = 4095;             //define sensor values
    public float ye = 4095;
    public float ze = 4095;
    public float zee = 4095;
    public float[] convertedItems;

    // Start is called before the first frame update
    void Start()
    {
        _eventSender = GameObject.FindGameObjectsWithTag(tagOfTheMQTTReceiver)[0].gameObject.GetComponent<
            mqttReceiver>();            //link to MQTTReceiver
        _eventSender.OnMessageArrived += OnMessageArrivedHandler;
                                                                                    //Link to event in mqttRceiver
            script
    }

    private void OnMessageArrivedHandler(string newMsg)
    {
        string[] tokens = _eventSender.messagenew.Split(',');                                                   //
            Split message into three sub values for force value
        convertedItems = Array.ConvertAll<string, float>(tokens, float.Parse);                                  //
            parse into a float

        if (_eventSender.topicname == "AAL_MCI_esp_Publish_Poti/WolfgangGrosek/5")                              //
            if topic == "AAL_MCI_esp_Publish_Poti/WolfgangGrosek/1"
        {                                                                                                       //
            use values for balance board
            xe = convertedItems[0] ;
            ye = convertedItems[1];
            ze = convertedItems[2];
            zee = convertedItems[3];
        }
    }
}
```

A.8. Game PlayController

```
using System.Collections;
using System.Collections.Generic;
using System.Threading;
using System.Threading.Tasks;
using Unity.VisualScripting;
using UnityEngine;
using UnityEngine.UI;
using System.Collections;

public class PlayController : MonoBehaviour
{
    private float speed = 5.0f;
    private float turnSpeed = 1.0f;
    public Text scoreText;
    public Text highscoreText;
    private GameObject image;
    public GameOverScreen GameOverScreen;


    int score = 0;
    int highscore = 0;
    int maxPlatform = 0;

    [SerializeField]                    //Set field in graphical editor
    GameObject logicController;         //Define field
    mqttController func;                //needed to link this script to logicController script

    public mqttController Joint2;       //Used to be able to have acces to degree variables from logicController
        script

    // Start is called before the first frame update
    void Start()
    {
        func = logicController.GetComponent<mqttController>();      //Used to be able to have acces to degree
            variables from logicController script
        scoreText.text = score.ToString() + " POINTS";
        highscoreText.text = "HIGHSCORE: " + highscore.ToString();
        image = (GameObject)Resources.Load("RawImage");
        AddImage();
    }

    private void OnCollisionEnter(Collision collision)
    {
        if(collision.gameObject.name == "Obstacle_1" || collision.gameObject.name == "Obstacle_2" || collision.
            gameObject.name == "Obstacle_3" || collision.gameObject.name == "Obstacle_4" || collision.gameObject.
            name == "Obstacle_5" || collision.gameObject.name == "Obstacle_6" || collision.gameObject.name == "
            Obstacle_7" || collision.gameObject.name == "Obstacle_8")
        {
            Debug.Log("POOOOW");
            ScoreManager.instance.RemovePoint();
        }
    }

    public void AddImage()
    {
        Vector3 spawnPosition = new Vector3(0, 2, 0);
        Quaternion spawnAngle = Quaternion.Euler(0, 0, 0);
        Instantiate(image, spawnPosition, spawnAngle);
    }

    // Update is called once per frame
    void Update()
    {
        //Move Vehicle
        turnSpeed = (Joint2.xe+ Joint2.ye)/(Joint2.ze + Joint2.zee);
        transform.Translate(Vector3.forward * Time.deltaTime * speed);
        if (turnSpeed != 0)
        {
            transform.Rotate(Vector3.up, (turnSpeed - 1) * 40 * Time.deltaTime);
        }
        else
        {
            transform.Rotate(Vector3.up, turnSpeed * 2*40 * Time.deltaTime);
        }
        Debug.Log(turnSpeed-1);

        if (this.gameObject.transform.position.y > -3)
        {
            ScoreManager.instance.AddPoint();
```

```
        }
77
        if (this.gameObject.transform.position.y < -3)
        {
            Debug.Log("Falling");
            GameOverScreen.Setup(maxPlatform);
82
        }

    }
}
```

A.9. Game ScoreManager

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
4 using UnityEngine.UI;

public class ScoreManager : MonoBehaviour
{
    public static ScoreManager instance;
9
    public Text scoreText;
    public Text highscoreText;


14  int score = 0;
    int highscore = 0;

    private void Awake()
    {
19      instance = this;
    }

    // Start is called before the first frame update
    void Start()
24  {
        highscore = PlayerPrefs.GetInt("highscore", 0);
        scoreText.text = score.ToString() + " POINTS";
        highscoreText.text = "HIGHSCORE: " + highscore.ToString();
    }
29
    // Update is called once per frame
    void Update()
    {
        //score = score + 1;
34      scoreText.text = score.ToString() + " POINTS";


    }

39  public void RemovePoint()
    {

        score -= 50;
        scoreText.text = score.ToString() + " POINTS";
44  }

    public void AddPoint()
    {

49      score += 1;
        scoreText.text = score.ToString() + " POINTS";
        if (highscore<score)
        {
            PlayerPrefs.SetInt("highscore", score);
54      }
    }

}
```

B. SPSS

Tests of Normality

	Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Statistic	df	Sig.	Statistic	df	Sig.
Traditional	.137	10	.200*	.956	10	.742
Digital	.108	10	.200*	.974	10	.928

*. This is a lower bound of the true significance.

a. Lilliefors Significance Correction

Figure B.1.: The normality analysis of the test scores in SPSS.

Correlations

		Traditional	Digital
Traditional	Pearson Correlation	1	.990**
	Sig. (2-tailed)		<.001
	N	10	10
Digital	Pearson Correlation	.990**	1
	Sig. (2-tailed)	<.001	
	N	10	10

**. Correlation is significant at the 0.01 level (2-tailed).

Figure B.2.: The Pearson Correlation analysis of the test scores in SPSS.