Wearable productivity measurement IoT system development

Reevan Heppell[1] and *André* van der Merwe[2]

[1,2] Department of Industrial Engineering, University of Stellenbosch University, Country South Africa, e-mail address andrevdm@sun.ac.za

Abstract. This project develops a system that captures and stores acceleration and rotation data from an operator's hand and tests said data's compatibility with a machine-learning algorithm. Tracking the productivity of operations is a goal a follow-up project aims to achieve, and this system is a necessary part of a possible solution. An IoT system was developed and tested by presenting the data to a machine learning algorithm to ensure that an algorithm can identify certain movements from the data. The results are valid since the system itself was tested in a case study with 24 participants and performed as expected. the data that the system provides as output was presented to two machine learning algorithms and both were able to identify movements with more than 80% accuracy.

1 Introduction

1.1 Problem background and origin

Operator productivity has a substantial impact on an organisation's performance. Thus, one of the ways the performance of an organisation can be improved is by improving the productivity levels of the operators.

To be able to improve operator productivity, it is necessary to track said productivity. Tracking the number of units an operator produces in a shift can be an indication of their productivity. However, this cannot directly indicate the areas where productivity can be improved.

Methods such as motion and time studies can be used to identify areas in an operation that can be changed to improve the operator's productivity. Some of these methods include a time study, work sampling, or making use of predetermined time systems.

Each of the three methods mentioned above has unique advantages and disadvantages. Using a combination of these and other methods minimises or overcomes the disadvantages. Thus these methods are useful to improve the operator's productivity in the right scenario.

In addition to these methods, an analyst can also look at each movement of a step to determine if that movement was productive or not. The analyst can do that with the help of the Therbligs principles. The operator's productivity level can be increased by eliminating the ineffective Therbligs.

1.2 Problem statement

The three motion and time study methods mentioned in, Problem background and origin may be used to track the productivity of an operator and help identify specific steps in tasks where the productivity of the operator can be improved. These methods all have one downside, none of these methods is particularly well suited to an environment where the operators perform non-repetitive tasks.

These methods require an analyst to actively spend time to perform the analysis. This is not inherently a problem, but it does make it infeasible to continuously track the operators.

Not being able to continuously track the productivity of the operators in a financially viable approach has a few drawbacks. One of these drawbacks is not being able to incrementally change the environmental factors, such as lighting, and observe the effect on operator productivity.

Identifying the effective and effective Therbligs also requires time from the analyst. The analyst first has to observe the task and then identify each element. Only then can the analyst start classifying each element as effective or ineffective Therbligs. With a repetitive task, this is feasible, but once again it is not suitable for an environment with non-repetitive tasks. Therefore, the research problem is whether a system that tracks hand movements can assist in the identification of ineffective Therblig elements.

1.3 Problem objectives

The objective of this project is to start the development of an IoT system that can track the productivity of operators with little input from an analyst.

This project aims to perform a part of the research and work needed to achieve the overall objective. There have been projects that started this development and there will be future projects to continue the development.

Considering the role of this project in the overall development, the objectives of this specific project are as follows: Firstly, improve the wearable device responsible for tracking acceleration and rotation. Secondly, develop a stand-alone IoT system to transfer and store data captured by the wearable device. Lastly, test the validity of the data for use as input for a machine learning algorithm to identify movements.

2 Literature review

2.1 Productivity

Productivity is generally defined as the ratio that is calculated by dividing the output by the input. Ghoddousi & Hosseini (2010)

2.1.1 Current methods

There are various ways of measuring productivity, and the following three are some of the methods. These three methods are similar and require very little equipment.

Predetermined time systems is a method where an analyst observes a task and identifies individual steps. Within these steps, elements of each movement are identified. The analyst then uses a system such as Method-Time Measurement (MTM) or Maynard Operation Sequence Technique (MOST) to determine the time each element should take. All of these times can then be added up to determine the total time the average operator would need to complete that task. This is also known as the normal time. Thus, it cannot be used to track the individual steps of an operator. The total time a task should take an operator can be compared to the times that task took, but if the operator's time is slower than the normal time a different method has to be used to find which steps the operator is falling behind in if there are any specific steps. Niebel & Freivalds (2014)

Time study is a method where an analyst observes an operator performing a task. The analyst then identifies the steps an operator follows to complete the task. A stopwatch is then used to determine the duration of each step. The analyst has to observe a task being performed several times, depending on how long the task takes to complete. This method can also be used to determine the normal time, but unlike the predetermined time systems, this method tracks each step the operator performs. This can help identify the exact steps an operator can improve or perform differently to be more productive. Niebel & Freivalds (2014)

Work sampling is a method that uses statistics to help determine the standard time of a task and the steps involved. The standard time of a task is the normal time for that task plus a little extra time making allowance for certain working conditions. With this method, an analyst observes one or many operators performing the same task. Unlike time studies the analysis does not observe the tasks continuously and only takes note of what step or action the operator or operators are doing at random intervals. This data is then tallied up and can be used to determine the standard time for each step. Niebel & Freivalds (2014)

2.1.2 Measurement with sensors

Accelerometers have been used for movement detection in many different environments, including human movement. However, it is not just accelerometers that are being used for this purpose. GPS, heart rate monitors, thermostats, and video cameras are some of the other sensors being used for tracking human activity. Moore et al. (2021) The data gathered from these sensors can then be used to identify movements. Jurˇci´c & Magjarevi´c (2022) used only an accelerometer to gather data that was used to recognise forward falls, backward falls, and side falls among other activities such as sitting, walking, standing, standing up, sitting down, and lying down.

2.2 Artificial intelligence

Using artificial intelligence to identify movement has been approached in a number of different ways. One specific approach was to compare different machine learning algorithms to identify activities in daily living. The emphasis, however, was placed on recognizing when a human body falls and what type of fall occurred. Jurˇci´c & Magjarevi´c (2022) The machine learning algorithms compared by Jurˇci´c & Magjarevi´c (2022) include K-Nearest Neighbour, Random Forest, Support Vector Machine, Decision Tree, and Gaussian Naive Bayes. It was found that Random Forest classification has the potential to be an area for further research.

2.3 Therbligs

Therbligs refer to categories such as reach, grasp and more into which human movement can be classified. This is a time and motion study system that was developed by Frank and Lillian Bilberth in the 1920s. Everett (1993) The motions identified can be used as the target features for a machine learning algorithm. If all the motions are placed in these categories, the principle of effective vs ineffective Therbligs can provide an indication of productivity.

3 Methodology

The overall approach that will be used is the systems engineering V-model. Within the v-model, rapid prototyping will be used at different stages during the development of the hardware and software as well as training the machine learning algorithms.

3.1 V-model

The V-Model has two phases, project definition and project test and integration. These phases are broken down into steps, which are followed in this project.

The v-model is applied once to the IoT System. However, it can be seen as three subsections within each step of the verification phase and the unit testing step in the validation phase. These three subsections are the SDS Wearable, the server and lastly the IOT System as a whole.

3.2 Rapid prototyping

For the server, rapid prototyping is used during the coding, unit testing and to a lesser extent, the integrated testing steps of the V-model.
With the IoT system as a whole, rapid prototyping is not used, whereas, in the development of the SDS Wearable, rapid prototyping is employed during all the steps of the V-model except the acceptance test. All the other phases are repeated numerous times for the development of the SDS Wearable. The coding, unit testing, and integrated testing are the three steps in the SDS Wearable development that follow the rapid prototyping approach in the closest manner.

4 Hardware development

4.1 Hardware overview

For hardware development, there are several different components. These components include a microcontroller, accelerometer, battery and case.

4.1.1 Microcontroller

The ESP32-E is the microcontroller that was selected to use in the wearable device. With a dual-core, WiFi capability, PH2.0 lithium battery port and General-Purpose Input Output (GPIO) pins to connect to the accelerometer all requirements were met.

4.1.2 Accelerometer

The accelerometer is an MPU6050. The MPU6050 consists of a three-axis accelerometer, a three-axis gyroscope and a temperature sensor. The GPIO pins on the MPU6050 and ESP32-E are connected via wires soldered to each component.

Inter-Integrated Circuit I^2C is the protocol with which the ESP32-E and MPU6050 communicate and transfer data.

4.1.3 Battery

A LiPo 1000mAh 3.7V battery provides power to the MPU6050 and ESP32-E during the measurement. The ESP32-E has an integrated charging secret and can be charged through the USB-C port on the ESP32-E.

While in use the combined load of the ESP32-E and the MPU6050 on the battery is 4V and 0.022Ah. This results in an expected battery life of 42 hours of use.

4.1.4 Layout

Minimising the size of the case was the main concern for the layout. Thus, a configuration was chosen where the MPU6050 is attached to the underside of the ESP32-E so that the ground, 3-volt, Serial Data Line (SDA) and Serial Clock Line (SCL) pins are lined up and connected by pine soldered to both boards. In addition, a Wire connects the interrupt pin on the MPU6050 to GPIO pin 15 on the ESP32-E which does not line up as seen in ***Fig. 1*** and ***Fig. 2***.

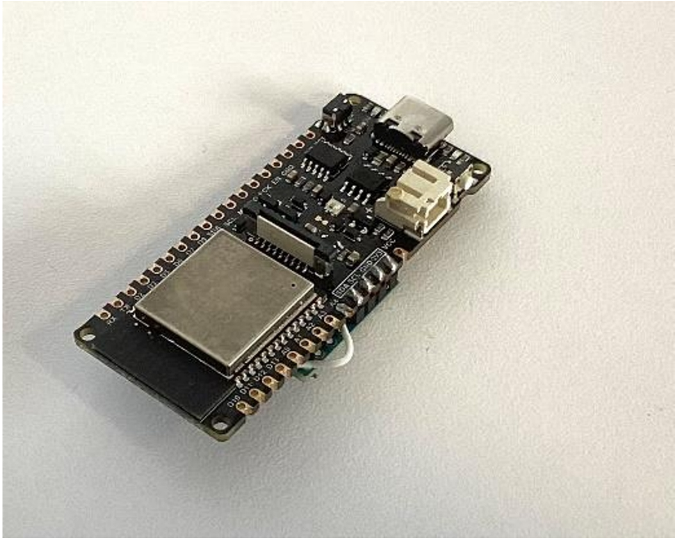

Fig. 1: *ESP32-E*

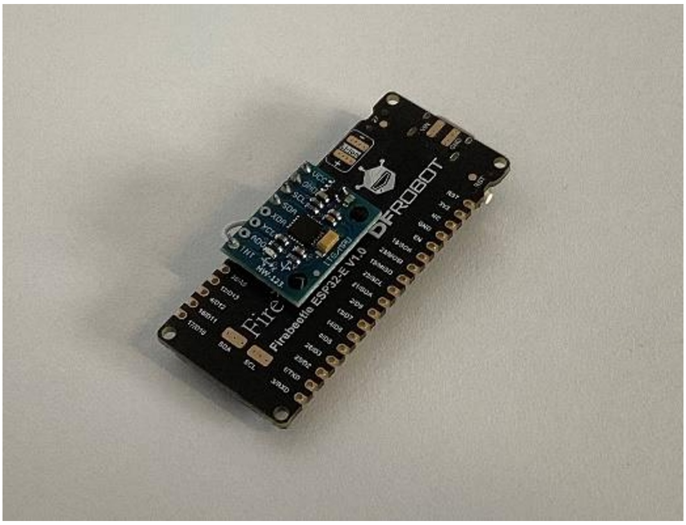

Fig. 2: ESP32-E and MPU6050

The battery connects to the PH2.0 pin on the ESP32-E and has a 100mm cable. Thus, the battery is connected to the ESP32-E, but not attached in a fixed position as the MPU6050 or ESP32-E. With this flexibility, it is possible to mount the battery to the lid as shown in ***Fig. 3***.

Having the battery mounted to the lid allows the removal of the lid and battery, without disconnecting the battery while providing access to the ESP32-E as seen in ***Fig. 3***. In addition to easier access to the ESP32-E, this configuration would allow for a clear view of the RGB-LED on the ESP32-E from the side. The clearance between the battery and the ESP32-E in this configuration also prevents the ESP32-E from overheating.

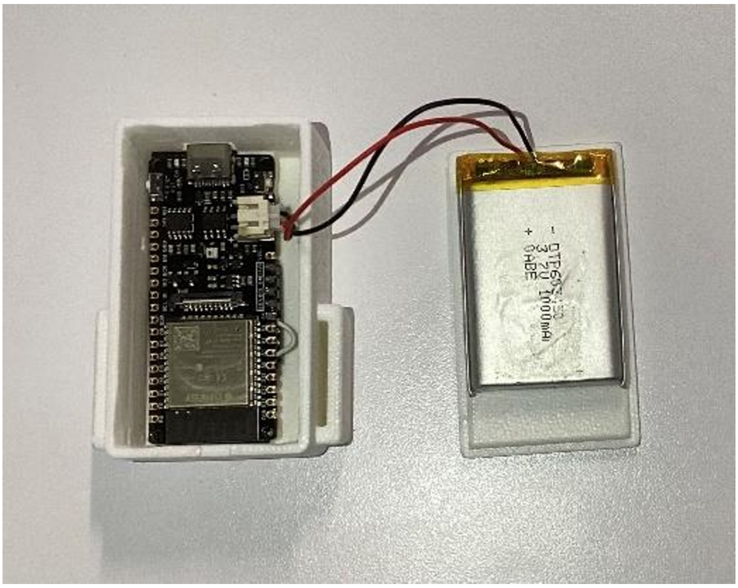

Fig. 3: Open Case

4.2 Case

The case had the overall requirement of having to provide a manner of attaching the electronic components to an operator's arm without hindering the operator's movement. Secondly, the case had to protect the electronic equipment. Thirdly, the case was required to prevent the ESP32-E from overheating. Lastly, the RGB-LED had to be visible to the operator.

To attach the case to the operator's arm, two slots were included in the design of the case. One on each side as can be seen in ***Fig. 4***. The slots are 22mm by 15mm to allow a 20mm velcro strip to be fed through without letting the velcro slide around in the slot.

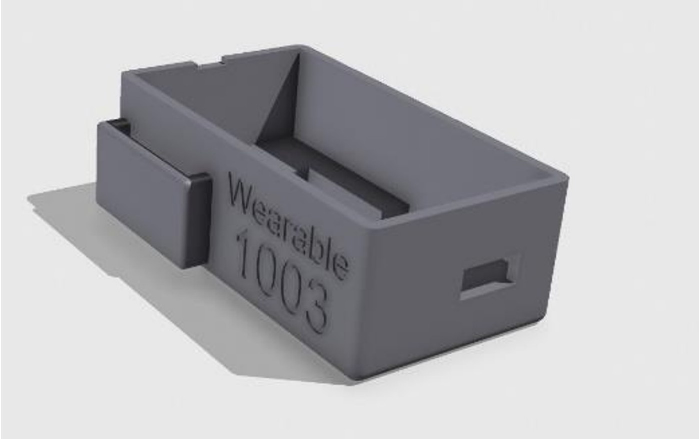

Fig. 4: Case Model in CAD

The case was designed to be worn in a similar position as a watch since this position would not restrict the operator's hand or wrist motions. Thus the double-sided velcro strap passes over the top of the case, through the slots and around the operator`s wrist as illustrated in ***Fig. 5***.

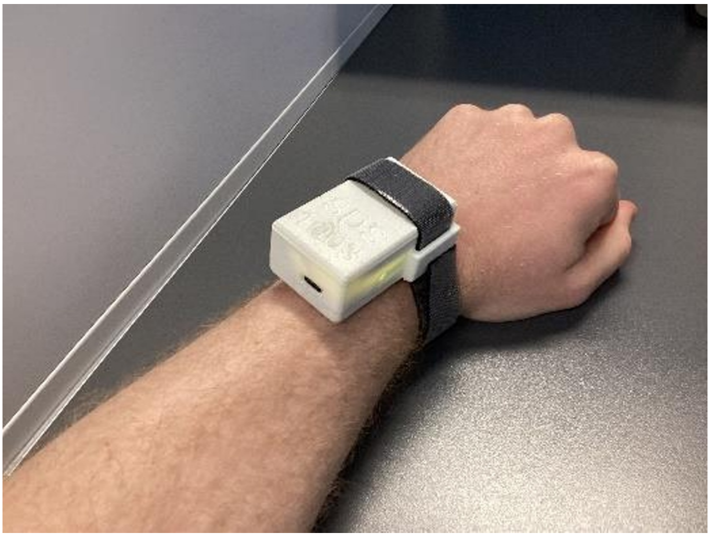

Fig. 5: Wearing the Case

To protect the electronic equipment the case had to be impact-resistant and have a small footprint. Since the case is on a continuously moving part of the operator's body, a smaller footprint would help prevent interference with the movement and reduce the likelihood of the device being damaged during use. With this in mind, the inside of the case is 34mm wide, 63mm in length and 22mm tall. The width and length dimensions were determined by the battery and ESP32-E respectively. The height was determined by the layout of the components.

Due to the placement of the MPU6050, the ESP32-E could not be mounted in the bottom of the case, but the two blue highlighted sections illustrated in ***Fig. 6*** were designed to be 5mm taller than the bottom of the case. This along with the double-sided tape used to mount the ESP32-E to the highlighted sections, there was roughly 1mm of clearance below the MPU6050.

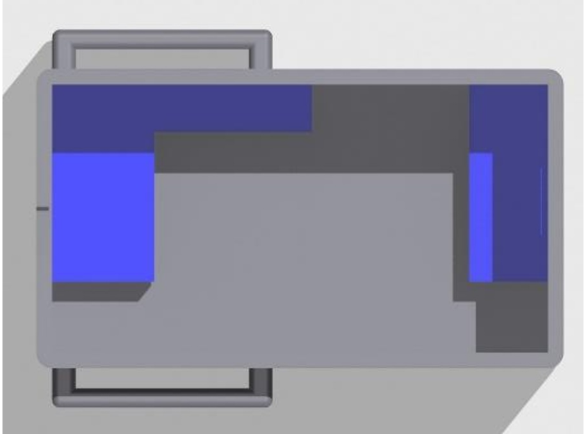

Fig. 6: Supports for MPU6050

The RGB-LED on the ESP32-E is used to convey information to the operator while the device is in use. THE RGB-LED shines through the sides of the case but is faint when using PLA material or in bright environments. Thus, a 4mm diameter hole is included on the side of the case. The hole and green light can be seen in ***Fig.*** *7*.

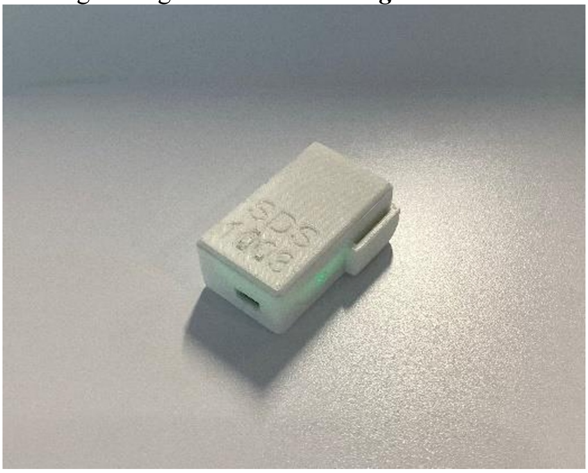

Fig. 7: Case

5 System development

The system's overall objective is to provide acceleration data from an operator's wrist to a machine learning algorithm. This data must be in a format that allows a machine-learning algorithm to identify the operator's movements. The system as seen in ***Fig. 8*** there are four components in the system. The Smart Digital System Wearable (SDS Wearable), server, processing device and WiFi.

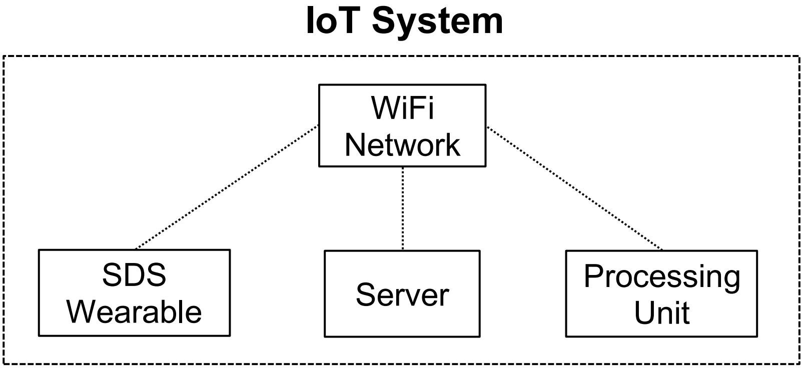

Fig. 8: Iot System

5.1 SDS Wearable

The Smart Digital Systems (SDS) Wearable is the part of the system responsible for reading the data and sending said data to the server. As mentioned in section 4.1 the MPU6050 measures the acceleration and rotation data while the ESP32-E records and transmits said data.

5.1.1 SDS Wearable general software

The SDS Wearable is the only part of the system that an operator will come into contact with. Thus, the SDS Wearable is tasked to do more than passively record the acceleration and rotation of the operator's movements. Therefore, the SDS Wearable is programmed to do the following, illustrated in ***Fig. 9***.

During the boot-up process, the SDS Wearable performs several operations. These operations include establishing a connection between the ESP32-E and the MPU6050 via I^2C, as well as preparing two separate folders on the internal storage of the ESP32- E. Thereafter, the secondary core, CORE0, is activated as well (Spiess, 2017).

With the main core, CORE1, and the secondary core active, the SDS Wearable can perform tasks in parallel. This allows the SDS Wearable to record incoming data with one core and transmit said data with the other core. The ability to perform these tasks in parallel is required for the SDS Wearable to record and transmit usable data over extended periods. Without this ability, the system would also not be able to work with data in real time. After the boot-up and activation processes, CORE0 is exclusively used for recording, refining, and storing the data from the MPU6050.

While the processes on CORE0 continue, CORE1 performs the following tasks sequentially: waiting for data that is ready to be transmitted, connecting to the WiFi network and MQTT broker. After establishing these connections, it transmits the data to the MQTT broker and ensures that storage space is made available for new data. Subsequently, CORE1 checks the stopping criteria. In this case, it monitors ten cycles of this process. If the stopping criteria have not been met, the process repeats; otherwise, if the criteria have been met, CORE1 finishes the last transmission.

After the last data has been transferred, CORE1 prepares the SDS Wearable for sleep mode. First, the MPU6050 is configured to detect sudden changes in acceleration and send a signal via the interrupt pin when the MPU6050 detects a tap. Secondly, the Wearable ceases all processes besides tap detection and enters a deep sleep mode to conserve energy and prevent the recording and transmitting of data during times when it is not required.

While in deep sleep, the SDS Wearable can be moved and attached to an operator's wrist without activating it. Once the SDS Wearable is required to record and transmit the data, the operator can give the SDS Wearable a firm tap. This action will wake up the SDS Wearable and restart the entire process by initiating the boot-up process.

For the duration of the process, the RGB-LED is used to communicate with the operator. During the boot-up procedure, the LED flashes blue a total of six times. While the data is being recorded, the LED alternates between green and yellow after each 10-second interval to indicate the different measurement cycles on CORE0. If the WiFi or MQTT broker is unavailable at the time of transmission, a red light is flashed once. After the stopping criteria have been met and the SDS Wearable enters sleep

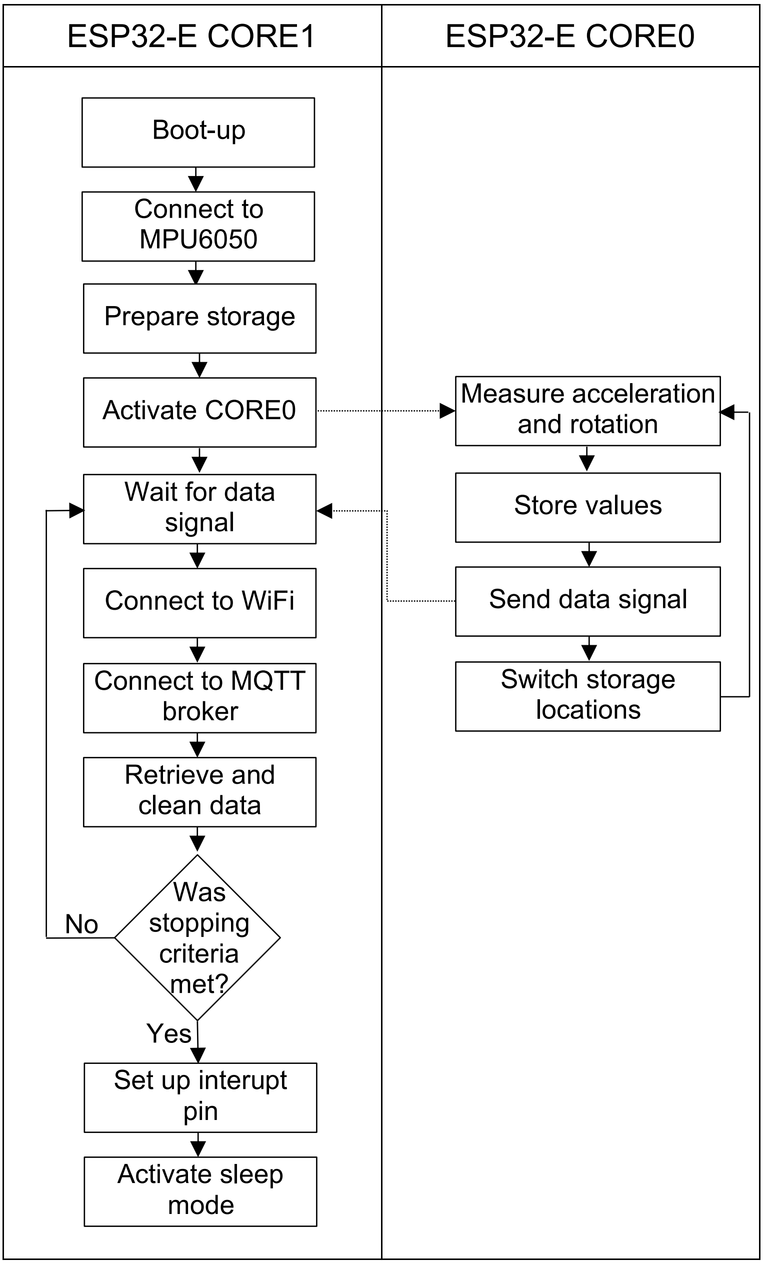

Fig. 9: SDS Wearable code diagram

5.1.2 SDS Wearable recording data

For the data to be useful, a high sampling frequency is required along with minimal interruptions. Thus, the ESP32-E is programmed to use one of its cores to record the acceleration and rotation data while the other core periodically sends said data to the server.

The ESP32-E continuously records data and calculates the averages of these measurements in 20 millisecond intervals. This results in seven average values, along with the time, being temporarily stored on the ESP32-E every 20 milliseconds (see photo below). This process is repeated 500 times over approximately 10 seconds. After 500 repetitions, the process restarts while the original values are transmitted via Message Queuing Telemetry Transport (MQTT) to the MQTT server as seen in ***Fig. 10***.

This approach allows the SDS Wearable to record data at 50Hz with brief interruptions of 80 milliseconds every 10 seconds. This makes battery life the only constraint on the operating time for the SDS Wearable. The program was also specifically designed to be able to increase the frequency of the measurements above 50Hz if required.

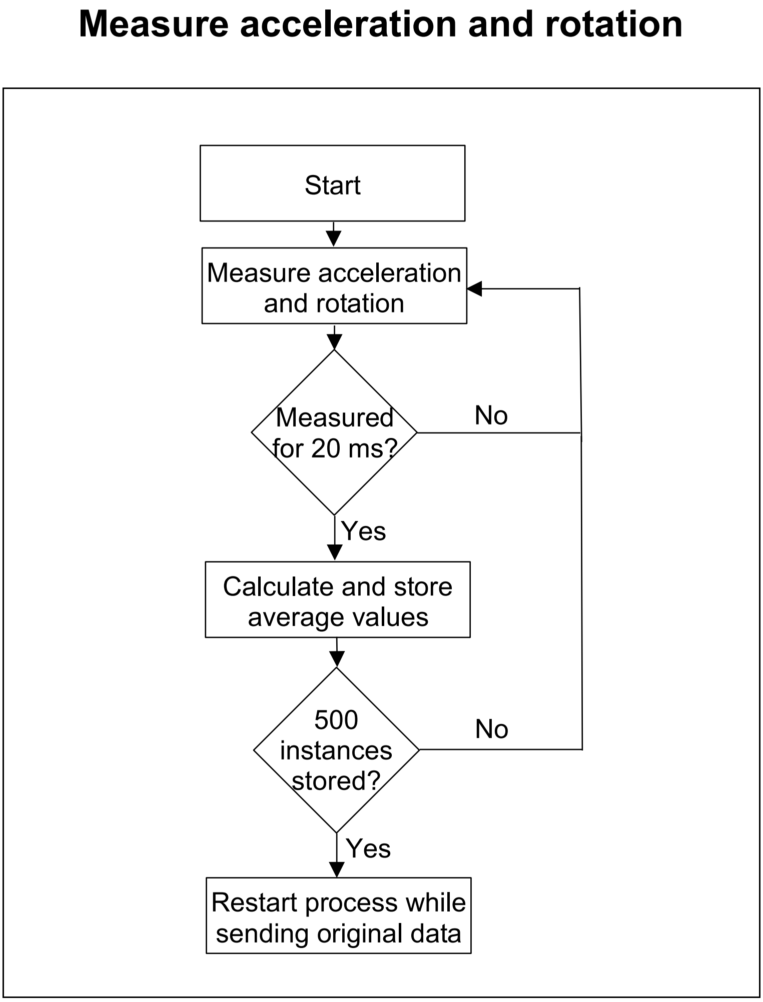
Measure acceleration and rotation
Start
Measure acceleration and rotation
Measured for 20 ms?
No
Yes
Calculate and store average values
500 instances stored?
No
Yes
Restart process while sending original data

Fig. 10: Measurement code diagram

5.2 Network

This system is required to transmit data wireless between the hardware devices, specifically between the SDS Wearable and the server. As stated earlier, the MQTT protocol was used, and for this protocol to work, the SDS Wearable and the server should be connected to the same network.

In addition, this network transmits the data from the MQTT broker to the computer responsible for analysing the data using the MQTT protocol. The next section discusses this in more detail.

5.3 Server

For the system to function, a device at least capable of receiving and storing data from the SDS Wearable is required. Within this system, a Raspberry Pi is used as the server.

It is possible to process the data to identify the movements performed on the server. However, this is not the case in the current system. Within this system, the server receives the data, performs minimal refinement, and stores the data. The refinement process formats the data to be categorised, making it easier to store and test changes made to other parts of the system.

These processes require the server to run on Raspberry Pi OS (formerly Raspbian) and two other programs: MQTT Mosquitto and Node-RED. The next sections explain in more detail what each program is used for.

5.3.1 MQTT Mosquito

To use the MQTT protocol in the system, the system must contain an MQTT Broker. The server is thus set up as the MQTT Broker. MQTT Mosquito is the program installed on the server that allows the server to be set up as the MQTT Broker.

Thus, the SDS Wearable connects to the MQTT Broker and publishes the data through MQTT messages. Since the data is stored on the server the server subscribes to the MQTT broker under the same topic to which the SDS Wearable publishes the data. The Server subscribes to the MQTT Broke via Node-RED which is discussed in the next section.

5.3.2 Node-RED

Node-RED is installed on the server since the IoT system combines hardware and software for which Node-Red is well suited. Through Node-Red the server subscribes to the MQTT broker to receive the data as it comes in through the massages.

After the message is received, the data is extracted from the message. That data is then refined by using a function node. This function node formats the data to separate the acceleration and rotation into the respective axes as well as a vector containing the combination of the acceleration data. Thereafter the data is stored in a .csv file.

5.4 Processing device

The processing device in the current system is a laptop running on Windows 11. This device also uses Node-RED to fetch the data from the server. The data is then stored in the same format as on the server (.csv).

From the .csv files, the data is read into Python where the final cleaning and preparation of the data is performed. After the data has been cleaned, a separate Python script presents the data to a machine learning algorithm, which returns an output for each movement.

6 Machine learning

6.1 Machine learning overview

Aligning with the main aim of the project, which is to develop an IoT System that forms part of a larger development process, the larger development process aims to determine the productivity of operators by tracking movements. To be able to do this, acceleration and rotation data of an operator's movement must be gathered in such a manner that the data can be presented to a machine learning algorithm, which has to do two things: the identification of specific movements as well as determining the duration of each of these movements.

For this project to be successful, the IoT system should record and transmit data. This data must be at a sampling rate and quality high enough for the machine learning algorithm to be able to perform the required tasks. Thus, the aim of training the machine learning algorithms was to test the quality of the data collected and transferred by the IoT system.

6.2 Algorithms

For this project, two machine learning algorithms were trained and tested. These are not integrated into the IoT system. The data stored on the processing device is manually presented to the algorithms for training and testing.

The algorithms were trained to identify three types of motion of an operator's wrist. These three motions include identifying when the wrist is stationary, moving in a horizontal plane, or moving in a vertical plane. For this reason, two classification algorithms were used. These two classification algorithms are K-Nearest Neighbour (K-NN) and Random Forest (Krishnan et al., 2009).

6.2.1 Data collection

As previously discussed, the data can be viewed as having a sample rate of 50 Hz, due to storing the data every 20 ms. This sample rate is the same for all the data that is collected.

This data includes the acceleration in m/s2 in the x, y, and z directions, as well as the rotation in rad/s in the x, y, and z directions. The time is also stored alongside the other values. Lastly, the acceleration vector is calculated and stored at the same sampling rate. The acceleration vector is calculated using Equation 1.

$$Vector = x^2 + y^2 + z^2 \qquad (1)$$

6.2.2 Data cleaning

The data is stored and sent in groups of 500 instances. These are then used as individual movements where each movement is 10 seconds long since each of the 500 instances represents 20 milliseconds. This method also provides the data in a format that works well for classification algorithms as the data is already separated.

The data is further processed to identify the required features. For each of the movements the maximum, minimum, variance, mean and median are calculated. This is done for the acceleration and rotation in each of the three directions as well as the time and vector. this results in each movement having 40 fetchers and one target value. Upon training the two algorithms it was discovered that the removal of the time data increases the accuracy of the algorithms.

6.2.3 Algorithm training

To train both the K-NN and Random Forest algorithms, 480 movements were used. These movements were equally split between stationary, horizontal, and vertical movements.

The data was split into 70% training and 30% validation. With this, the K-NN algorithm achieved an accuracy of 87%. The Random Forest algorithm achieved an accuracy of 93%, which could indicate that the model was overfitting.

7 Case study

7.1 Case study overview

The case study was in the form of a project that was developed by Prof. van Der Merwe. This was done to test the second prototype of the SDS Wearable. Along with this the case study also aimed to provide exposure to an IoT System for the participants.

The case study was in the form of a project where 24 participants were provided with an SDS Wearable and asked to perform several tasks. The first task was to determine the maximum rotation range of a hand. Secondly, the acceleration data had to be used along with integration to determine linear movement distances. Lastly, the participants had to determine a distinct pattern for a repetitive motion from the data and any possible combinations of the data.

7.1.1 IoT system

The IoT system used in the case study differs slightly from what is described in Chapter 5. These differences were present in all aspects of the IoT system but varied to varying degrees in each section.

In terms of hardware, the 3D-printed case and the battery were different from the final design. The SDS Wearable 1.2 contains a 110mAh battery compared to the 100mAh battery in the earlier version. The footprint and overall size of the 3D-printed case were smaller for the SDS Wearable 1.2 due to not having to provide space for the larger battery.

The same microcontroller (ESP32-E) and accelerometer (MPU-6050) were used in the SDS Wearable 1.2. However, the connection differed slightly. The 3.3V, ground, SDA, and SCL pins were all connected in the same manner, but the interrupt pin on the MPU-6050 was connected to pin 39 on the ESP32-E.

The program that ran on the ESP32-E also differed between the two versions of the SDS Wearable. For SDS Wearable 1.2 the ESP32 was programmed to collect data in a similar manner as explained in Section 5.2.2. The main difference is that only one core was used. Thus all the measurements were done the data was compiled and sent through to the MQTT Broker.

7.1.2 Network

The case study was conducted with 24 participants over a month-long duration. Thus, the wearable could not be confined to one network. To overcome this, the ESP32-E was programmed to have the ability to connect to multiple networks.

The server also had to support data being transmitted from multiple networks. For this reason, an external server set up by Prof. Van Der Merwe was used. This server hosted the MQTT broker as well as the Node-RED software.

7.1.3 Processing device

During the case study, the data was processed by the participants and was not presented to a machine learning algorithm. Thus, a part of the processing was done on the server. However, the participants also used personal computers to process the data.

The participants managed this by accessing Node-RED on the server and storing the data on their personal devices. This was done in the laborious manner of copying and pasting data from Node-RED. From there, the students used the data to achieve the three objectives mentioned earlier.

7.2 Findings

From this case study, a number of observations were made. These observations include parts of the system that performed well and parts of the system that did not perform as well.

The part of the system that performed well was the interface of using the RGB-LED to provide feedback to the participant from the ESP32-E. The tap detection from the MPU-6050 worked well to allow communication from the user to the SDS Wearable 1.2. The design and positioning of the case also performed well. The sample rate of the SDS Wearable was also sufficient.

The use of Node-RED to integrate the hardware and software, along with minor processing, also worked as intended. The only note is that it is best to allow only one analyst to make changes to the programming or setup of Node-RED at a time. This should not prove to be a problem for future projects, since only one person will work on a Node-RED portal at a time, but it is worth investigating for future projects similar to the case study.

There were issues with parts of the system that did not perform well and required improvement during the next prototype and further development. These issues included a battery life of less than two hours of measurement time. This was solved by using a 1000mAh battery in the next prototype. Another concerning issue was the loss of data during transmissions from the SDS Wearable to the MQTT Broker. This was an irregular occurrence and was indirectly solved by two other changes, which are discussed later.

The measurement time was a maximum of 20 seconds, which is not practical for the final application of the IoT system. The solution required setting up the ESP32- E to use both cores as well as the flash memory. This setup allows the data to be sent in smaller packages with one core while using the other core to measure. The measurements were stored in the flash memory by one core and cleared by the other core after it had been transmitted. This setup allows the cores to work independently and not interfere with one another tasks. With this and the use of a local network, the problem of MQTT messages being lost was also solved.

8 Conclusion

This project successfully developed and tested an IoT system designed to measure, transmit, and store acceleration and rotation data. The system was built for future use in evaluating operator productivity by analyzing movement patterns.

A case study involving 24 participants demonstrated the system's ability to support multiple SDS Wearables simultaneously. Additionally, the study revealed that the system could detect unique signatures from the acceleration and rotation data for repetitive movements.

Movement data collected during the project was further analyzed using a machine learning algorithm, which correctly identified three types of movements 80% of the time. These results suggest that the developed IoT system will generate sufficient data for future research.

While the system met the project's initial requirements and objectives, there is still room for improvement. The most significant area for enhancement is the machine learning algorithm used to classify movements, as the current implementation was intended primarily to test the data.

References

1. Everett, J.G. (1993). Design-fabrication interface: Construction vs manufacturing. In G.H. Watson, R.L. Tucker & J.K. Walters, eds., Automation and robotics in construction X: proceedings of the 10th International Symposium on Automation and Robotics in Construction (ISARC), 391–397, International Association for Automation and Robotics in Construction (IAARC), Houston, TX, USA.
2. Ghoddousi, P. & Hosseini, M.R. (2010). A survey of the factors affecting the productivity of construction projects in Iran. Iran University of Science and Technology, received 28 September 2009; accepted 02 November 2010.
3. Jurci ˇ c, K. & Magjarevi ´ c, R. ´ (2022). Physical activity recognition based on machine learning. 37–41, Periodica Polytechnica Budapest University of Technology and Economics.

4. Krishnan, N.C., Juillard, C., Colbry, D. & Panchanathan, S. (2009). Recognition of hand movements using wearable accelerometers. Journal of Ambient Intelligence and Smart Environments, vol. 1, no. 2, pp. 143–155.
5. Moore, R., Archer, K.R. & Choi, L. (2021). Title: Statistical and machine learning models for classification of human wear and delivery days in accelerometry data.
6. Niebel, B.W. & Freivalds, A. (2014). Methods Standards & Work Design. WCP/McGraw-Hill, 13th edn.
7. Spiess, A. (2017, November 5). #168 ESP32 Dual Core on Arduino IDE including Data Passing and Task Synchronization [Video]. YouTube. https://www.youtube.com/watch?v=k_D_Qu0cgu8 (Accessed: 2024).