# ECSE 444 Final Project Report

## Group 13 | December 5th, 2023

Chenyi Xu
260948311
McGill University
chenyi.xu@mcgill.ca

Fengqi Zhang
260963858
McGill University
fengqi.zhang@mail.mcgill.ca

Alex Wei
260981800
McGill University
yihui.wei@mail.mcgill.ca

Zhanyue Zhang
260944809
McGill University
zhanyue.zhang@mcgill.ca

## I. Design Problem

Smart watches are increasingly popular nowadays. People wear their watches for various purposes, including tracking sleep patterns and monitoring workouts. The team concentrates on achieving a CNN-based real-time Human Motion Detection (HMR) system with STM32 B-L475E-IOT01Ax kits. Such a system is capable of classifying basic human motions.

## II. Design Workflow

### A. General Solution with Block diagram

The team decided to engineer an advanced human classification program utilizing USB, STM32AI, UART, and I2C functionalities with low power operation as optimization. This application is capable of classifying motions like stationery. walking, running, jumping, clapping, and swimming. Datasets for stationery, walking and running were downloaded from STM32 Cube AI GitHub Repository [1]. Data for jumping, clapping, and swimming were collected at a rate of 26Hz using I2C feature and saved into the csv file using USB feature. Sleep mode will be entered when data is being processed into buffer before saving into USB to save power. The deep learning model is trained off-board utilizing PyTorch and TensorFlow's Keras API, showcased in PyCharm. The board will be tied onto the right hand of the user. I2C accelerometer will record the motion of the user. The result will then be classified and printed on the terminal using UART. The block diagram of the overall system is shown below.

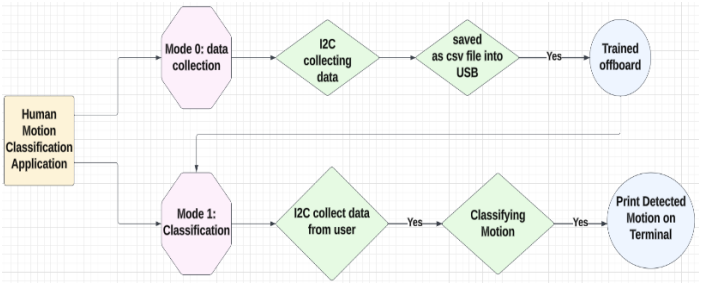


*Fig. 1 Block Diagram of the Application*

### B. Component detail, why implementing each component

There are three major components for this application which are data collection, training, and classification.

#### A) Collection

The data is collected by reading I2C accelerometer at 26 Hz sampling rate. For each motion type, twenty separate data sets will be recorded. The USB interface will then be responsible for automatically saving the data into a csv file onto the USB disk. This requires us to config the board as a USB host and utilize the middleware FATFS. The data will be automatically truncated to fit the input for our NN. Without this, collecting data can be painful and manually copy-paste from UART can be a huge hurdle for us to extend the range of the classification and improve the accuracy of our NN.

Lower power operation is implemented as the optimization method in this part. The system will enter sleep mode before the data is saved into the buffer. In addition, the system will exit sleep mode when the interruption is called. *Figure 2* shows the code required to enter sleep mode. To initiate Sleep mode, the systick interrupt is first disabled, this process achieved using HAL_SuspendTick(); Subsequently, the system executes the 'wait for interrupt' instruction. This action places the system into sleep mode after acquiring acceleration data via I2C and before storing the data in the buffer.

```
if (mode==0){
  for(int i=0;i<1000;i++) {

      LSM6DSL_Axes_t acc_axe;
      LSM6DSL_ACC_GetAxes(&MotionSensor, &acc_axe);
      HAL_SuspendTick();
      HAL_PWR_EnterSLEEPMode(PWR_MAINREGULATOR_ON, PWR_SLEEPENTRY_WFI);
//    for(int i=0;i<190000;i++);
      HAL_Delay(38);
      buffx[i]=acc_axe.x;
      buffy[i]=acc_axe.y;
      buffz[i]=acc_axe.z;
  }
  }
```

*Fig. 2 Sleep mode implementation*

#### B) Training Procedure

Key steps in our CNN training structure involves several are as follows. It starts with data preparation, where accelerometer data is imported from CSV files, standardized to US-ASCII encoding, and cleaned to remove invalid entries. This ensures high-quality, compatible data for training.

The CNN model, developed using TensorFlow and Keras, comprises layers designed for time-series data: Conv1D layers, Batch Normalization, ReLU Activation layers, and Dropout

layers, concluding with a Dense layer with softmax activation for classifying different motions. This architecture is adept at processing temporal accelerometer data. Focal loss is employed as the loss function to address the imbalance in the dataset. Training involves running the model through several epochs using the RMSprop optimizer (Adam manifests similar performance), with a focus on computational efficiency.

```
model = Sequential([    # REFERENCE [1]
    layers.Conv1D(filters = 64, kernel_size = 5, input_shape = (26, 3)),
    layers.BatchNormalization(),
    layers.Activation('relu'),
    layers.Conv1D(filters = 32, kernel_size = 5),
    layers.BatchNormalization(),
    layers.Activation('relu'),
    layers.Dropout(0.25),
    layers.Flatten(),
    layers.Dense(64),
    layers.BatchNormalization(),
    layers.Activation('relu'),
    layers.Dense(NUM_STATE, activation = 'softmax')
])
```

*Figure 3. CNN core structure*

C) Classification & Printing

During this mode, the system constantly reads the I2C accelerometer at 26 Hz and feeds the normalized X, Y, Z value sequentially into a buffer. For each prediction, we collect 26 sets of values. The NN will give an array of confidence for each class as the output, the class with highest confidence will be selected as the prediction result. The data is transmitted to UART and will then be printed on the terminal.

## III. METRICS

### *A. Evaluation Methodology*

A) Collecting

The board acquires I2C accelerometer data using BSP functions to gather acceleration statistics of different motions, which are then trained and utilized to predict the movement. Subsequently, this acceleration data is transmitted to UART and printed out as results in the terminal. The frequency of reading and printing is set to 26 Hz, achieved by HAL_Delay(). The data attained will be evaluated. We will make sure the data sets are consistent. In addition, data elimination will be performed for the ones didn't capture correctly. For instance, in occasion only X coordinate is captured but empty Y and Z, due to some inevitable transmission error. The data can later be verified to see if the motion is correctly recognized.

B) USB Interface

The board utilizes a USB interface to gather 1,000 acceleration data points of various motions at a frequency of 26 Hz and records them into a CSV file on an external USB flash disk. To avoid confusion in each data collection and writing session, the CSV file is overwritten each time. There are two components to testing the USB interface. Firstly, data must be collected at a frequency of 26 Hz. Although the frequency of reading and writing is tested during the collection phase, it is not yet determined if the use of the USB interface affects this frequency. The testing procedure is as follows: a timer is used to measure the data collection duration. Since the program begins by collecting data, the timer is started immediately after the program runs. A message displayed in the terminal will indicate the end of the timing period. This test will be conducted 10 times, and the average duration should be approximately 38 seconds. Secondly, to verify if the right data is written in the csv file, the UART will print the acceleration acquired and then transfer 10 data into csv file via USB. Compare data in csv file and the data in terminal to verify transmission works well.

C) Training & Cube AI

The accuracy of movement prediction can be effectively evaluated using a confusion matrix. In this context, a total of 6 categories (stationary, walking, running, jumping, clapping, breaststroke) will be evaluated. Subsequently, corresponding accelerations will be generated by I2C and utilized as inputs for the model to predict the associated movements. The confusion matrix is used to visualize the performance of this model, providing details of correct and incorrect predictions in different movement categories.

D) Low power

Current at pin JP5 are measured after removing the jumper with a multimeter, which can be seen in figure 4. Values before applying lower power and after lower power are measured for comparison purposes. It is designed as the optimization feature for this project.

*Figure 4. Measuring Current*

E) Classification & printing

Users will perform the motion of stationery, walking, running, jumping, clapping and swimming 10 times respectively. The data will be collected from I2C and the output result will be

transmitted through UART and be printed on the terminal to see if the motion has been successfully classified by the system.

### *B. Performance and Results*

We took the evaluation phase of the CNN model's seriously, which encompasses both visual and quantitative assessments. A key component of this evaluation is the use of a confusion matrix [1], which offers a detailed visualization of the model's performance across different classes. This matrix is instrumental in identifying both the strengths and weaknesses of the model, highlighting the specific motion classifications that are accurately identified and those prone to misclassification.

In addition to the visual assessment, the model's accuracy is rigorously quantified. The overall accuracy metric provides a holistic view of the model's performance, while the individual class accuracies offer a granular look at how the model fares in classifying each specific type of motion. These metrics are indispensable in gauging the effectiveness of the model and guiding further refinements. The results are sufficiently satisfying to reach accuracy and precision over 98% on average.

The next and last evaluation step would be practical validation using the board. During initial testing, we found the model can easily confuse running and jumping. After reviewing, we came to a conclusion that this is due to lack of dataset and collected a bit more data with different gesture angles. We also optimized the CNN implementation to address the imbalanced nature of our dataset library by changing to focal loss function. The outcomes of these modifications turned out to be effective and gave us a satisfactory final product.

#### A) Collecting

For each motion, 5 sets of data are collected. Data evaluations mentioned above are then performed. All data collected was consistent. The accelerometer is working as expected and will be used to collect more data for each motion.

#### B) USB Interface

During the I2C data collection, the average time is 39 seconds, which corresponds to the expected 26 Hz. Additionally, the data in the CSV file is consistent with the data printed in the terminal.

#### C) Training & Cube AI

The figure below shows the confusion matrix using the STM board to predict movement. The average accuracy is 97.5%. It is obvious that there is some misclassification for clapping and breaststroke. The reason is that these two movements are more complex than the others. Another reason is that the training data for these two movements is less than the others.

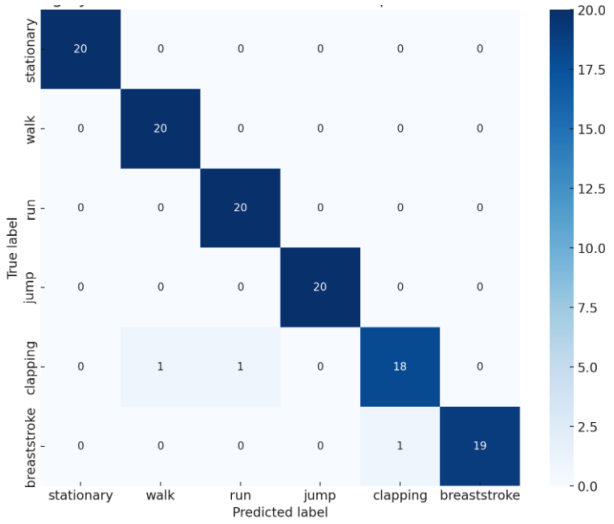


*Figure 5. prediction confusion matrix*

#### D) Low Power


*Figure 6. Current in normal mode (top) and sleep mode (bottom)*

The results demonstrating lower power are illustrated in the figures above. The current measured under normal operation condition is 828 μA. In contrast, sleep mode reads 821 μA.

#### E) Classification & Printing

The results are outputted on the terminal. After performing all motions, the application is proved to be functional of classifying all motions. The discrepancy can be caused by inaccurate placement of microcontrollers. The application is very sensitive to how the motion is performed and susceptible to the environment.

| **Motions** | **Still** | **Walk** | **Run** | **Jump** | **Swim** | **Clap** |
|---|---|---|---|---|---|---|
| **Accuracy** | 10/10 | 10/10 | 9/10 | 9/10 | 10/10 | 10/10 |

*Table 1 Application Performance*

### *C. Accomplished and Possible Optimizations*

Our training algorithm has been greatly optimized. The algorithm can now automatically unify encoding, delete void rows, update evaluations and generate models upon adding new dataset. A key upgrade is to eliminate the necessity of truncating all primitive data to the same length, which wastes a huge portion of the crucial data. Instead, we directly perform frame fragmentation to each individual csv collected. The memory efficiency has also been improved by 75% by forcing data type

to int16. The CNN-Network design is also redesigned to emphasize the imbalanced dataset among motion categories.

### *D. Challenges and Solutions*

The data was collected using I2C accelerometer and displayed on terminals using UART. Initially, for each trial, we gather approximately between 800 to 1000 points. Copy and paste data from the terminal to an excel sheet. These data are then automatically preprocessed by a python script aiming to cur void data points and to truncate them into a uniform length guaranteeing that a consistent dimension of input can be fed to the training algorithm. Note that the one or two files with the minimum number of points will be discarded for statistical significance. It is a lot of work to manually copy and paste everything into excel. To streamline the workload, the team implemented a more efficient file system where we configure the board as a USB host and utilize the middleware FATFS to facilitate easier data handling and integration into our training pipeline.

The team found that the current system is very robust. The motion can only be accurately sensed when the motion is performed in a specific way. It is very susceptible to the motion environment. The accuracy of the classification is not as good as expected, especially ones with subtle differences. More data sets need to be collected to resolve this issue.

### *E. Eliminations (Design Alternatives)*

Initially, we decided to manually copy and paste the data collected in UART terminal and to save them into a csv file. Configuring a USB interface will help us improve efficiency. During the USB interface design process, we find that mounting the USB disk and opening and closing the file will take up most of the running time. Also, writing 1000 pieces of information takes around 15 seconds. We made an improvement that writes 20 lines each time and write 50 times in total. By doing this, the time of execution has been reduced to around 5s, which is an optimal alternative.

To enhance the robustness of our HMR model, multiple improvements were made in refining the Deep Learning (DL) model. Initially, the team faced challenges with the model's sensitivity to specific motion patterns and its performance in diverse motion environments. The model also struggled with accurately classifying motions that had subtle differences. To address these issues, the team undertook a series of modifications to the DL model. This included adjusting the model architecture to better capture and differentiate nuanced motion patterns and implementing more sophisticated data preprocessing techniques to ensure that the model was trained on high-quality, representative data. Moreover, the model's ability to learn from a wider range of motion data without overfitting has been optimized. This involved fine-tuning the model parameters and potentially exploring different model architectures or learning algorithms. The aim was to create a model that is not only accurate in its current state but also scalable and adaptable to new motion types and environments.

Another improvement lies in the automation of preprocessing of primitive csv data. The necessity of truncation is also eliminated; instead, we directly extract frames from csv files, which utilizes all data to the maximum extent possible.

These modifications to the DL model significantly enhanced its robustness, making it more reliable across various motion types and less prone to inaccuracies due to environmental factors or subtle motion differences. This iterative approach in model development, aligned with the improvements in data handling, demonstrates the team's commitment to developing a highly efficient and accurate HMR system.

### *F. Recommendations Based on Experience*

For improved project execution, more comprehensive documentation of each code version is essential. The team, which previously relied on TEAMS for code and documentation sharing, experienced disorganization due to multiple members updating the code. To address this, the team would switch to using GitHub for better version control and collaboration. Moreover, the existing documentation lacked detail; going forward, the team would focus on meticulously documenting every stage of the development process, ensuring clear and useful references for future work.

If we were to do this project again, a key focus would be on enhancing the expandability of the project which would enable the integration of additional features into the application, building upon the existing framework.

## IV. CONCLUSION

In conclusion, the team successfully engineered an application capable of classifying fundamental motions such as stationary movement, walking, running, jumping, clapping, and swimming. This solution integrates four key features: USB, STM32AI, UART, and I2C, ensuring robust and efficient functionalities. To optimize the system and make it more effective, lower power operation was performed. Moving forward, the team is committed to continuous enhancement and integration of this application as a feature for our capstone project.

## V. REFERENCES

[1] STMicroelectronics. (n.d.). *GitHub - STMicroelectronics/stm32ai-wiki: git repo to illustrates and supports the STM32 AI wiki articles*. GitHub. https://github.com/STMicroelectronics/stm32ai-wiki/tree/master (accessed Nov. 8th, 2023).