1 Introduction

1.1 Motivation

As per the project brief [1], the main motivation for this project is to create a solution to the inconvenience of playing drums. Real drums are loud, expensive and take space. Moreover, playing drums is not completely healthy. Several studies have shown that drumming contributes to hearing damage [2] and causes neurological and musculoskeletal injuries [3].

1.2 Proposed Solution

The proposed system utilises MEMS[1] technology to allow motion tracking of a drumstick in the air. This is achieved through the use of triple-axis angular rate sensors (gyroscopes), linear acceleration sensors (accelerometers) and hall-effect magnetic sensors (magnetometers[2]). When the data from these sensors is fused, an Earth-referenced orientation of the stick is produced. By monitoring the acceleration peaks, a signal processing algorithm is able to detect air strikes. These strikes wirelessly trigger drum sounds on a mobile device via Bluetooth. The system can also derive the loudness of the sound – which is proportional to the peak in angular velocity prior to a strike, and the type of drum that is being struck – which is dependent on the orientation of the drumstick at the time of striking.

Thanks to efficient time management, an extension to the originally planned work was made. An LSTM based neural network is utilised for more general motion detection. To train the network, a novel method in the form of a computer game is proposed that can efficiently label much data. Moreover, a relationship between the acceleration and the angular velocity is discovered that allows prediction of air strikes before they occur.

The resulting system is a compact PCB that has all the functionalities above and fits inside a drumstick. It is powered by a tiny $130mAh$ battery which can last weeks depending on the usage. The final design also includes an encapsulation of the product which is suitable for 3D printing or lathing. The testing results and feedback from drummers confirm that the system's performance is at a professional level.

[1] Microelectromechanical Systems

[2] Also referred to as a compass.

2 Background and Literature Review

The concept of air-drumming and virtual drumming is not new and there have been several studies conducted in this area; however, none of them have found a complete, high-quality and portable solution. This section reviews these studies as well as some novel techniques that can be adapted to improve on the existing work.

2.1 Current Developments

Okada et al. have exploited a camera-based motion sensing device and a projector to realise the idea of virtual drumming [4]. They project an image of circles - which correspond to drums - on a flat screen and hit them with a drumstick. The camera tracks the drummer's hands to detect a strike. However, their results show that the system is inaccurate, making it hard to produce a meaningful drum beat. Furthermore, it requires a flat area for projection, a camera and a computer, which defeats the idea of portability.

Kanke et al. manage to partially solve the portability issues with drumming by attaching accelerometers and gyroscopes to a drumstick [5]. Their idea is to substitute less frequently used drums, such as accent notes, with virtual ones. By looking at the acceleration and the angular velocity of the drumstick they manage to differentiate between a drum strike and a strike in the air. However, their system could not detect air strikes reliably and resulted in false positives and false negatives. It did not solve the portability issue completely either.

A more lightweight solution, that completely removes hitting surfaces is proposed in [6]. The suggested system requires a pair of motion sensors, attached to the drummer's arm. The acceleration data from the sensors is fed into a multilayer perceptron (MLP) [7], which decides whether the data corresponds to a strike. Although, this work proves the possibility of using a machine learning algorithm for air strike classification from acceleration data, the final product did not meet the standards required for professional music-making. Firstly, the author mentions that the velocity detection was not accurate and not always representative of the drummer's intentions. Furthermore, the response rate of the system was $124Hz$, which would result in a high latency jitter $\approx 8ms$; this was also noticed by evaluators of the product. In addition, the method for measuring latency was not quantitative, but rather objective. Another issue with the work is the method for

labelling data[1], which was highly inefficient, and prone to making mistakes. Lastly, the system was not wireless, which increased the inconvenience and decreased portability.

To summarise, the previous work provides some inspirations for this project. It also shows many gaps that can be improved.

2.2 Drumming Recognition

An air drumming system should take some motion data as an input, and output three things: a Boolean that tells whether the input corresponds to a strike; the velocity of that strike; the virtual drum this strike belongs to. Each of these outputs is a subproblem that needs to be solved.

2.2.1 Strike detection

In this paper the process of determining whether the motion data corresponds to a strike will be referred to as strike detection.

As seen in the developments section, there are two common ways of collecting motion data – with a camera or with inertial sensors.

Using a camera has several advantages. For example, it can directly give information about the position and orientation of an object within the field-of-view; velocity and acceleration can be easily derived with finite difference methods. However, image processing algorithms are relatively demanding, and more than a cheap microcontroller would be needed to execute them. What is more, the response rate would be inherently low due to the low frame rate of the camera. Lastly, cameras require extra setup which impacts portability.

The other option, and the one used in this project, is MEMS inertial sensors. They come in small, surface mount packages, and can fit inside a drum stick. Although the information they provide directly is more limited than a camera – only acceleration and angular velocity – there are algorithms that can derive orientation. Apart from portability, their biggest advantage over cameras is the response rate, which peaks at over $1000Hz$.

Current developments have tracked different parts of the arm as well as the drumstick to detect strikes, but none of them has given a reasoning for their choice. The motion of the upper arm, lower arm, wrist and drumstick have been captured

[1] Data labelling is a process, required for all supervised machine learning algorithms. More information is given in section 2.3.

in [8]. The results show that the most information is conveyed by the motion of the drumstick, since all other parts are almost stationary (Figure 2.1).

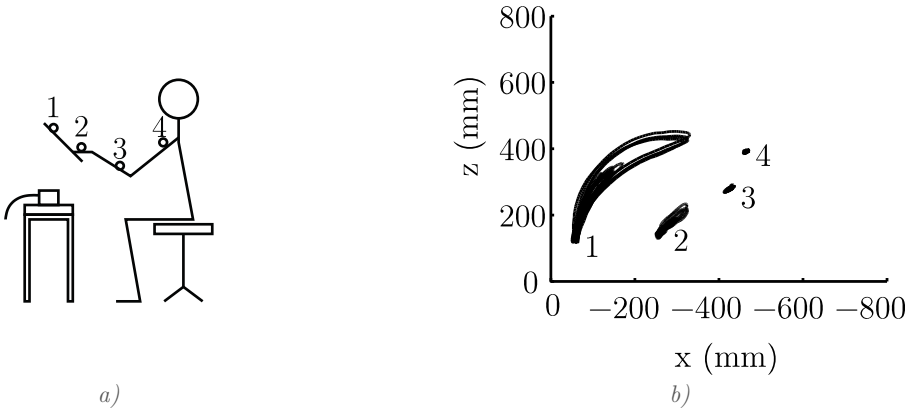


Figure 2.1. a) the locations of four markers that were placed on a drummer. b) the trajectories of these markers while playing. (Adapted from [8])

The definition of an air strike needs to be defined first, before creating an algorithm that can detect it. In this work, an air strike is the event at which the tip of the drumstick sharply changes direction from going down to going up. By applying trivial logic, this also means that the vertical velocity of the drumstick will change sign (cross the zero axis), and there will be a peak in the vertical acceleration of the drumstick. Although this definition was created through a thought experiment, there is also a paper [9] that leads to the same conclusion. Therefore, a strike detection algorithm would be looking for peaks in acceleration.

2.2.2 Velocity detection

A high-performance air drumming device should be able to produce a wide dynamic range of sound levels: that is, the user should have the ability to finely control the loudness of the generated sound. In real drums this is achieved by striking the drum with more force.

A study by S. Dahl et al. compares measurements of the striking velocity, and the produced sound level when hitting a drum [10]. The results show a linear relationship between the dynamic level and the striking velocity (Figure 2.2). This suggests that the peak of the angular velocity of the stick, when a strike occurs, can be used to infer the dynamic level.

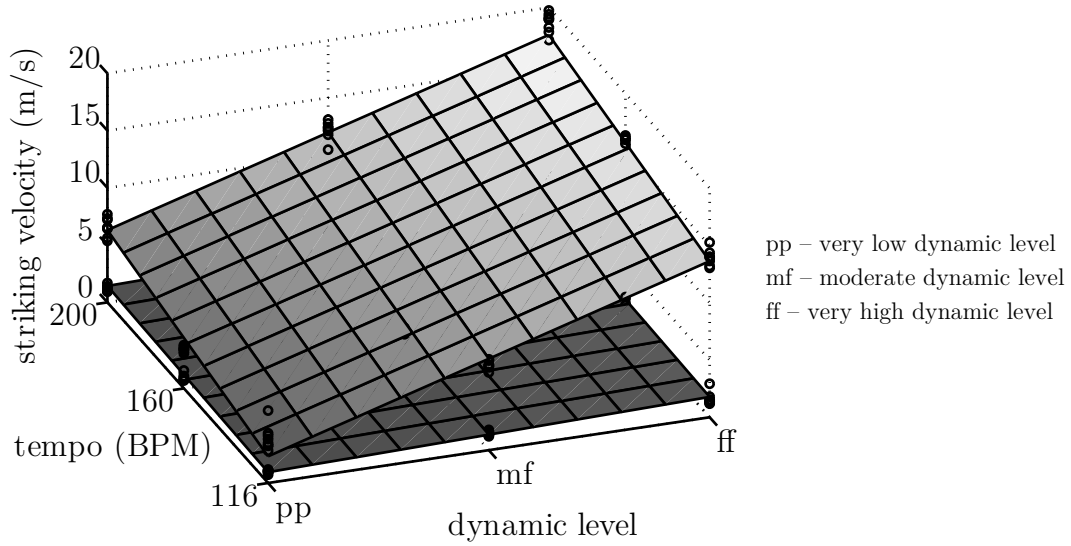


Figure 2.2. Linear relationship between the striking velocity and the produced dynamic level of a drum at different tempi. (Adapted from [10])

2.2.3 Drum detection

In the context of this work, drum detection means determining which drum does a strike belong to; which drum the player intended to hit (e.g. hi-hat, snare, etc.). The obvious solution is to track the location of the tip of the drumstick. Theoretically, this can be achieved by doing a double integration of the linear acceleration from the MEMS sensor. However, as shown in [11], this results in a significant drift over time (up to several meters per $10s$).

However, it is possible to obtain the orientation of the drumstick by integrating the angular velocity. The integration drift can be compensated by regularly adjusting the orientation to known references such as the Earth's gravitational force direction, and the Magnetic North. These references can be obtained from an accelerometer and a magnetic sensor; the process of fixing the drift is called sensor fusion. Many algorithms [11, 12, 13] of high complexity have been developed to maximise performance. Most of them require dynamic calibration of the sensors and optimisations to run on embedded systems, which is beyond the scope of this project. Instead, a compiled library from InvenSense will be used, which has been optimised for their line-up of sensors.

Although the position of the drumstick is not available, it has been shown in [6] that different drums can be inferred from just the orientation.

2.3 Machine Learning

The drumming recognition problems introduced in Section 2.2 are typical problems in the field of signal processing. For instance, the strike detection problem can be solved using a peak finding algorithm. Although a signal processing approach can

produce a very efficient way of solving a problem, the solution would be very specific – that is, if a different type of motion was to be found, a completely new algorithm would have to be developed for it.

That is where machine learning comes into play. Machine learning is a fast-growing field based on linear algebra and the theory of probabilities. The two major problems in machine learning are regression and classification: regression is concerned with fitting a representative line (or a hyperplane in problems of higher dimensions) to some data; classification, on the other hand, tries to find a curve/surface that separates some number of classes of data [14] (Figure 2.3). The strike detection problem in this project is a classification problem in which the two classes are "strike" and "no strike", and the input to the problem is the motion data.

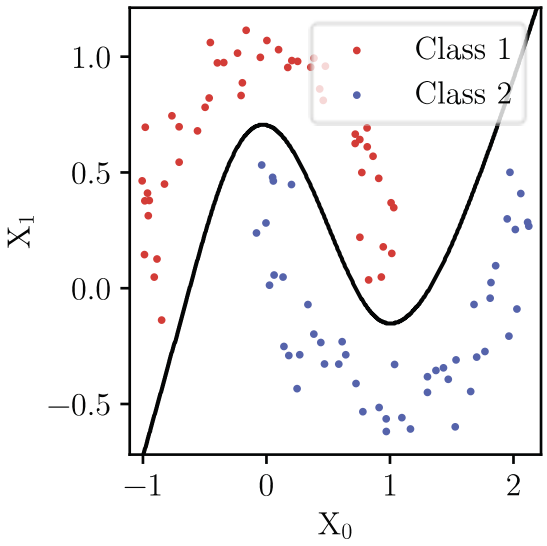


Figure 2.3. Dummy data that is classified by a neural network. The black line is called the decision boundary. A new point will be assigned to: class 1 if it is above the boundary; class 2 otherwise.

2.3.1 Neural Networks

The most basic building block of a neural network is the perceptron (Figure 2.4). It takes some number of inputs and outputs their weighted sum [15]. Often, the weighted sum is processed by a non-linear function $f(\cdot)$ called the activation function. A single perceptron is usually not useful – that is why many perceptrons are stacked in many layers to create a network called a Multilayer Perceptron (MLP). The activation functions in an MLP allows non-linear decision boundaries like the one in Figure 2.3 to be created.

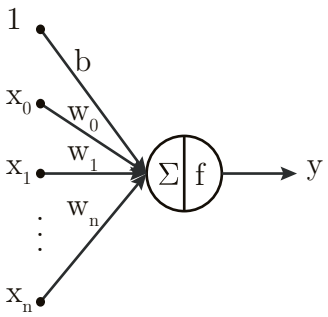


Figure 2.4. A single perceptron. The weights w_i allow the perceptron to prioritise certain inputs. The bias term b allows the perceptron to shift the decision boundary up and down. The bias term is usually not included in diagrams.

The mathematical expression of a perceptron is:

$$y = f\left(\left(\sum_{i}^{n} x_i w_i\right) + b\right) \tag{2.1}$$

The term learning in machine learning means finding the weights that create a decision boundary to best separate the classes.

A more advanced building block in neural networks is the Long Short-Term Memory (LSTM) cell (Figure 2.5). Unlike the perceptron which can only do inference based on the current inputs, the LSTM cell has memory and its output is also based on the previous state of the network.

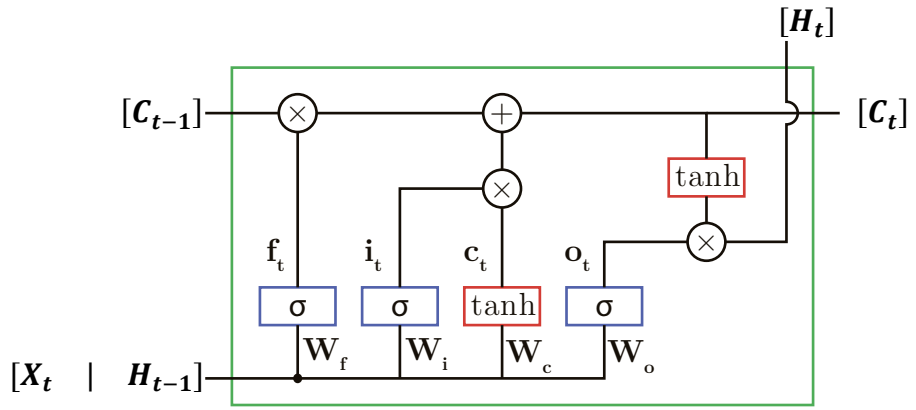


*Figure 2.5. LSTM cell structure. Note that the inputs and the outputs of the cell are matrices and not a single value. **X** is the output from the previous layer in the neural network. **H** is the output of the current layer of the network. **C** is the cell state. t and t-1 denote the timestep. (Adapted from [16])*

The main idea behind the LSTM cell is the cell state, which is represented as the horizontal line on the top of the diagram above [16]. Information is selectively added or removed in the cell state by the so called gates.

The first gate is called the forget gate and it is mathematically described by (2.2) (note that $\boldsymbol{b_f}$ is the bias term). It takes $\boldsymbol{X_t}$ and $\boldsymbol{H_{t-1}}$ as inputs and outputs a number between 0 and 1 – a 0 fully forgets the previous state, while a 1 fully passes it through.

$$\boldsymbol{f_t} = \sigma(\boldsymbol{W_f} \cdot [\boldsymbol{X_t} \quad | \quad \boldsymbol{H_{t-1}}] + \boldsymbol{b_f}) \tag{2.2}$$

$$\boldsymbol{i_t} = \sigma(\boldsymbol{W_i} \cdot [\boldsymbol{X_t} \quad | \quad \boldsymbol{H_{t-1}}] + \boldsymbol{b_i}) \tag{2.3}$$

$$\boldsymbol{c_t} = tanh(\boldsymbol{W_c} \cdot [\boldsymbol{X_t} \quad | \quad \boldsymbol{H_{t-1}}] + \boldsymbol{b_c}) \tag{2.4}$$

$$\boldsymbol{o_t} = \sigma(\boldsymbol{W_o} \cdot [\boldsymbol{X_t} \quad | \quad \boldsymbol{H_{t-1}}] + \boldsymbol{b_o}) \tag{2.5}$$

Next is the update gate that decides what new information to pass on to the next state of the network. It is described by (2.3), (2.4) and (2.6).

$$\boldsymbol{C_t} = \boldsymbol{f_t} \cdot \boldsymbol{C_{t-1}} + \boldsymbol{i_t} \cdot \boldsymbol{c_t} \tag{2.6}$$

$$\boldsymbol{H_t} = \boldsymbol{o_t} \cdot \tanh(\boldsymbol{C_t}) \tag{2.7}$$

Lastly, there is the output gate which decides what information the LSTM cell should output. The decision is based on the cell state and the inputs of the cell. The relevant equations are (2.5) and (2.7).

The reason that LSTMs are introduced here is because they have been outperforming all other models (including CNNs[1]) in numerous problems such as speech recognition [17] and event human activity recognition based on acceleration data [18]. Another advantage of LSTMs is that they do not require feature engineering; instead, they can directly take the raw data as an input. Lastly, they do not need windowing since the network keeps the previous information in the cell state. This means that there is no restriction in the length of the motions that can be learnt by the network. For more information on windowing and feature engineering see A.1 and A.2.

2.3.2 Training

As specified above, learning means adapting the value of the weights in the neural network so that it can correctly classify the input data. In the beginning, the network is initialised with random weights. The output of the network is then calculated, and the predicted output is compared to the expected output. If the prediction and the expectation do not match, the network is penalised using a loss function and the weights are updated using gradient descent to reduce the loss.

Many loss functions exist, but the one that is most commonly used in classification problems is the cross-entropy loss defined by [19]:

[1] Convolutional Neural Network

$$L(\boldsymbol{W}) = -\sum_{n=1}^{N}\{t_n \ln(\mathrm{y_n}) + (1-\mathrm{t_n})\ln(1-\mathrm{y_n})\} \tag{2.8}$$

Where t_n is the expected output of the network, y_n is the actual output and N is the number of classes. Note that the loss function is a function of the weights and the best weights aim to minimise it.

Training is always done on a portion of the dataset which is usually about 70% of the whole data. This is called the training data. The remaining 30% are called the validation data. The validation data is used to test the model on unseen data.

2.3.3 Activation Functions

Activation functions are non-linear functions that either squish an infinite space into a finite one (tanh and sigmoid) or rectify it (ReLU) (Figure 2.6). They are used in every neural network cell, which allows the network to classify data that is not linearly separable.

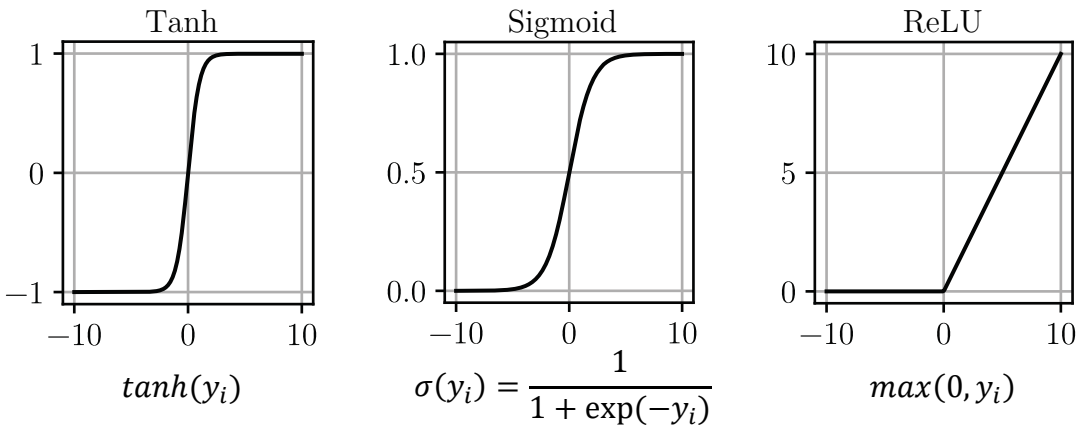


Figure 2.6. An example of the most common activation functions and their mathematical representations. (Equations reproduced from [14])

Another important activation is the softmax. The softmax is usually used on the output layer of a network. It converts the outputs of the network to a probability distribution [14]. The softmax is defined by:

$$softmax(y_i) = \frac{\exp(y_i)}{\sum_j \exp(y_j)} \tag{2.9}$$

Where i is the number of the output perceptron and the sum in the denominator iterates through all outputs of the network.

2.3.4 Overfitting, Underfitting and Regularisation

Complex networks with many layers and cells have the ability to create decision boundaries of a complicated shape by making the values of the weights big. Sometimes this can be a drawback since the network may learn things that are not useful for classification – for instance, noise in the data (Figure 2.7, right). A

network that is overfitting will usually have an extremely low loss with the training data, but a significantly higher loss with the validation data.

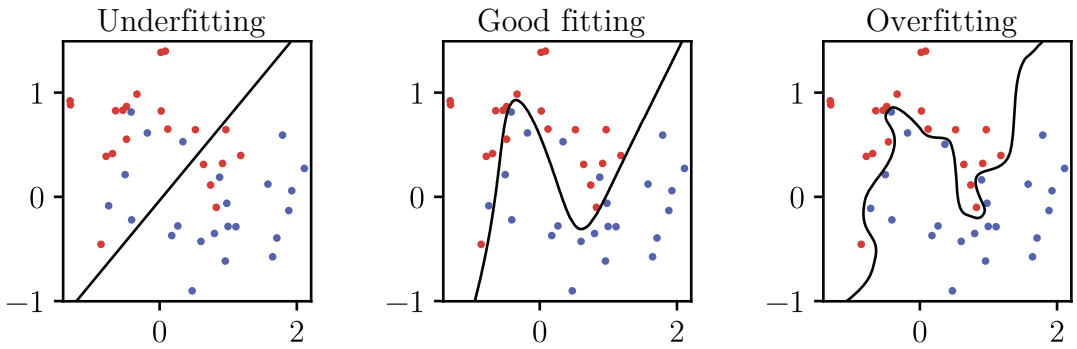


Figure 2.7. An example of underfitting and overfitting. Note how in the overfitting case, the decision boundary perfectly separates the scattered data. However, this scatter is due to noise and the decision boundary is not representative of the actual data that generated the distributions. On the good fitting example, the training algorithm has allowed some misclassification in an attempt to make the model generalise with unseen data. Too much generalisation resulted in underfitting (left) and the model could not learn the features of the distributions.

One solution to overfitting is regularisation. Regularisation adds additional penalty to the loss function that prevents the weights from becoming big. A common regularisation is the $l2$ norm of the weights $\|\boldsymbol{W}\|_2^2$, also known as $l2$ regularisation. To regularise a loss function, the $l2$ norm is simply added to it:

$$L(\boldsymbol{W}) + \lambda\|\boldsymbol{W}\|_2^2 \tag{2.10}$$

Where λ is a user-controlled parameter that controls the strength of regularisation [20]. If the regularisation is too high, the network is called to underfit – that is, it cannot learn the features of the data well (Figure 2.7, left).

2.4 Music Data Transmission

The background so far has covered how to detect a strike, its velocity and the type of drum that has been struck. The next step would be to send a triggering signal to a program that will generate a sound when a strike is detected. This can be done via MIDI.

2.4.1 MIDI

The Musical Instrument Digital Interface (MIDI) is the most widespread communication protocol for efficient musical information transmission [21]. It provides a standardised, message-based protocol which describes how a piece of music should be played. The relevant MIDI messages for this project are: the note-on message (Figure 2.8), which instructs the software to play a sound; and the note-off message, which instructs the software to stop playing a sound.

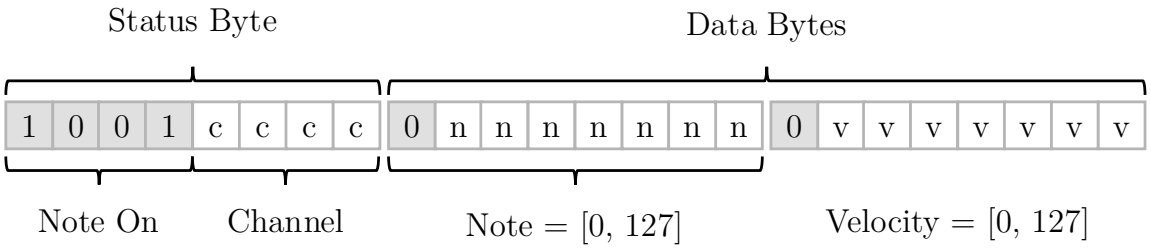


Figure 2.8. The MIDI note-on message structure. The user-modifiable values are in the white blocks. This type of message triggers a sound to start playing. The channel for percussion applications is 0x09. When using this channel, the notes are mapped to different drum sounds (see Table C.1). The velocity determines the loudness of the sound.

According to the MIDI specification [21], every note-on message should have a corresponding note-off message; this prevents the musical program to play a sound forever. Since drum sounds naturally decay over time, the note-off message does not affect how sound is played. Therefore, the note-off can be send together with the note-on for convenience (this is called a running status message [22]). A typical MIDI messages to trigger a drum sound would look like this:

byte 0 (Status Byte)	1	0	0	1	1	0	0	1
byte 1 (Data Byte)	0	n	n	n	n	n	n	n
byte 2 (Data Byte)	0	v	v	v	v	v	v	v
byte 3 (Data Byte)	0	n	n	n	n	n	n	n
byte 4 (Data Byte)	0	0	0	0	0	0	0	0

Figure 2.9. The format of a drum message. Note that the channel in the status byte has been set to 9 (the drum channel). The last 2 data bytes act as a note-off by setting the velocity to 0.

Although MIDI is perfect for musical data transmission, it is a wired protocol. To improve the portability of the air drumming system, a wireless protocol should be used.

2.4.2 MIDI Over BLE

Bluetooth Low Energy (BLE) is a version of Bluetooth that is optimised for low-power devices and it can reach speeds of $2Mb/s$ [23]. It achieves this by sending data in bursts, rather than continuously streaming.

BLE uses the Generic Attributes (GATT) profile to transmit data between devices. This profile describes how data is transferred using BLE. It contains services and characteristics (Figure 2.10). A service is simply a collection of information; its purpose is to logically separate one type of data from other types of data by their category. The service contains characteristics: the characteristic is simply a placeholder for some data (for instance, a sensor reading, or a MIDI message).

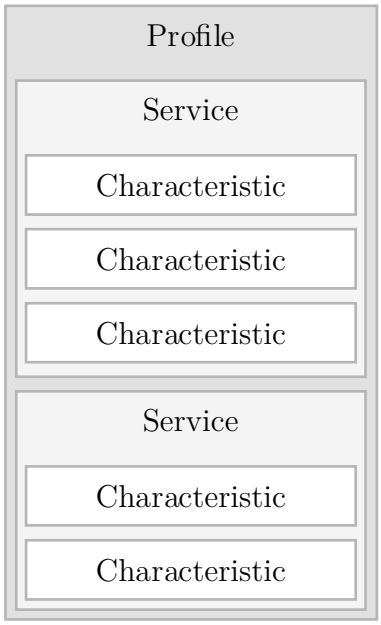


Figure 2.10. An example of a GATT profile. Each service and characteristic is identified by a UUID[1] (adapted from [24])

BLE-MIDI is simply a BLE service in which the characteristic is the MIDI message. The BLE-MIDI message is identical to the MIDI message, apart from a millisecond timestamp that is appended to the front of it (Figure 2.11). The timestamp is necessary since BLE buffers the MIDI messages and sends it in timed intervals. The timestamp tells the receiving device exactly when each data was created so that it can play the sounds with the right timing.

The utilising of BLE-MIDI allows for a completely wireless system.

byte 0 (Header Byte)	1	0	timestamp_high					
byte 1 (Message Timestamp Byte)	0	timestamp_low						
byte 2 (Status Byte)	1	0	0	1	1	0	0	1
byte 3 (Data Byte)	0	n	n	n	n	n	n	n
byte 4 (Data Byte)	0	v	v	v	v	v	v	v
byte 5 (Data Byte)	0	n	n	n	n	n	n	n
byte 6 (Data Byte)	0	0	0	0	0	0	0	0

Figure 2.11. BLE-MIDI message structure. The first two bytes contain the 13-bit, millisecond resolution timestamp generated by the microcontroller. The rest of the message follows the typical legacy MIDI structure. (Adapted from [25])

[1] Universally Unique Identifier

2.5 Sensor Data Acquisition

The most common interfaces for data transmission are the Serial Peripheral Interface (SPI), Two Wire Interface (TWI) and Universal Asynchronous Receiver-Transmitter (UART). The main benefits of the last two are that they require less connections between the communicating systems. However, this significantly affects their speed [26]. That is why SPI will be used in this work.

An SPI network usually consists of a single master (the microcontroller) and multiple slaves (the sensors) as shown in Figure 2.12. When the master wants to read from or write to a slave, it changes the value of the Slave Select (SS) line of that slave to active. The master then generates a clock signal on the Signal Clock (SCK) line and transmission starts on the Master In Slave Out (MISO) and Master Out Slave In (MOSI) lines [27].

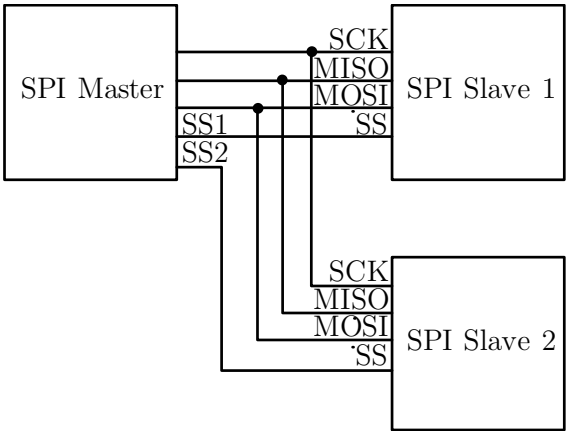


Figure 2.12. A typical SPI connection. (Adapted from [27])

The first byte (address byte) of an SPI transaction consists of a 7-bit register which is ORed with a read (1) or write (0) flag bit that determines the type of the transaction (Figure 2.13). The following bytes are the data. Due to the availability of two data lines, reading and writing can be performed simultaneously.

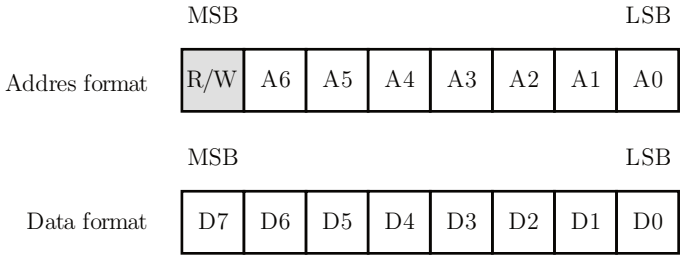


Figure 2.13. SPI bytes format. (Adapted from [27])

2.6 Real-Time Operating System

A few different tasks were covered so far: data acquisition, drumming recognition and music data transmission. For air drumming, all of these need to work concurrently. However, all of these tasks run at different rates so they cannot simply be implemented in a loop. FreeRTOS is a type of operating system that runs on microcontrollers. The main benefit of using it is that it allows different tasks to run independent from each other and at different rates. If multiple tasks have to run at the same time, the operating system chooses the one with the highest priority and runs it first [28].

Apart from allowing concurrency, using FreeRTOS would also make the code more modular, flexible and easier to maintain.

2.7 Performance Metrics

In order to evaluate and compare systems, there should be some common quantities that define their performance. None of the previous works have explicitly defined such measures. This paragraph proposes some performance metrics in the context of air-drumming.

2.7.1 Latency

Latency is a measure of the time that passes between an air strike and the sound being played; it is a fundamental metric that defines the quality of a digital music instrument. Latency that is high or jittery will make the instrument feel clumsy and unnatural to play, and it can greatly reduce performance.

Jack et al. [29] have found that the subjective impression of a percussionist about the quality of an instrument degrades as the latency and latency jitter increase. A rule of thumb, created by Wessel [30], is that an unnoticeable latency will be no more than 10ms, and the maximum latency jitter should be 1ms.

2.7.2 Predictability

Predictability means that the instrument should behave according to the player's intentions: that is, the player should be able to predict what the output of the instrument will be before performing an action.

In the context of air-drumming three things can be measured for predictability: strike detection, velocity detection and drum detection. To measure each of them, we could ask a drummer to perform N number of hits (N should be large) and count the number of hits $N_{correct}$ that produced a sound that matched the drummer's

expectation. Then (2.1) is a probabilistic confidence level that measures the predictability of the instrument.

$$N_{correct}: N \tag{2.1}$$

3 System Architecture

The aim of this section is to provide a brief overview of the system that was built in this work and give some context to the background section. A system overview is presented in Figure 3.1. On the left, there are the three sensors for motion. They are connected to the microcontroller via SPI. The microcontroller acquires the sensor data and performs sensor fusion, strike detection, velocity detection and drum detection. Strike detection is performed with both a system processing and machine learning algorithm. Lastly, a musical message is wirelessly sent via BLE-MIDI if a strike is detected. All these tasks are handled by FreeRTOS.

On the receiving side there is a computer/smartphone connected to a speaker. If the computer receives a BLE-MIDI message, it plays the sound of the corresponding drum with the requested loudness.

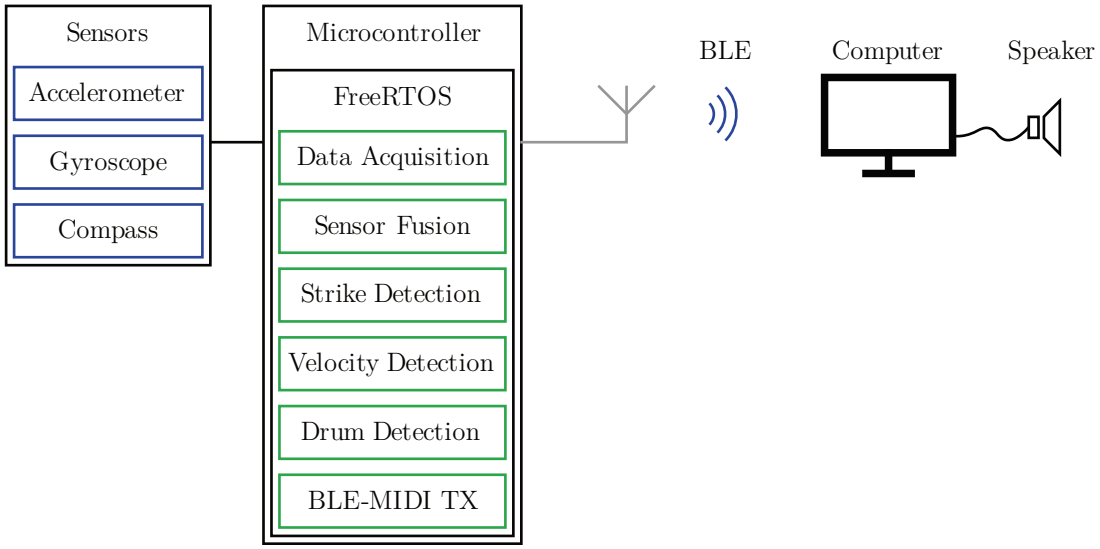


Figure 3.1. A simplified diagram of the proposed system.

4 Hardware Implementation

This section presents the hardware implementation. Two printed circuit boards (PCBs) were created for this project. The first one – the development board – was relatively big, with redundant hardware and pin headers for functionality extension. This board was used to develop a proof-of-concept, but it also allowed extra hardware to be connected to the microcontroller for testing, and data could be easily acquired through an UART-to-USB bridge. The second one – the pilot board – was a stripped-down version of the development board. It only contained the necessary features for an air-drumming device. It was made extremely compact so that it fits inside a drumstick.

4.1 The Development Board (Prototype 1)

The development board (Figure 4.1) is based around the nRF52832 by Nordic Semiconductor. This is a microcontroller with integrated BLE support. It is built around an ARM Cortex-M4 CPU and it has a floating point unit which is ideal for fast calculations. The system clock is $64MHz$. In addition, the chip comes with an example-rich Software Development Kit (SDK) and a large community, which makes software development fast and easy.

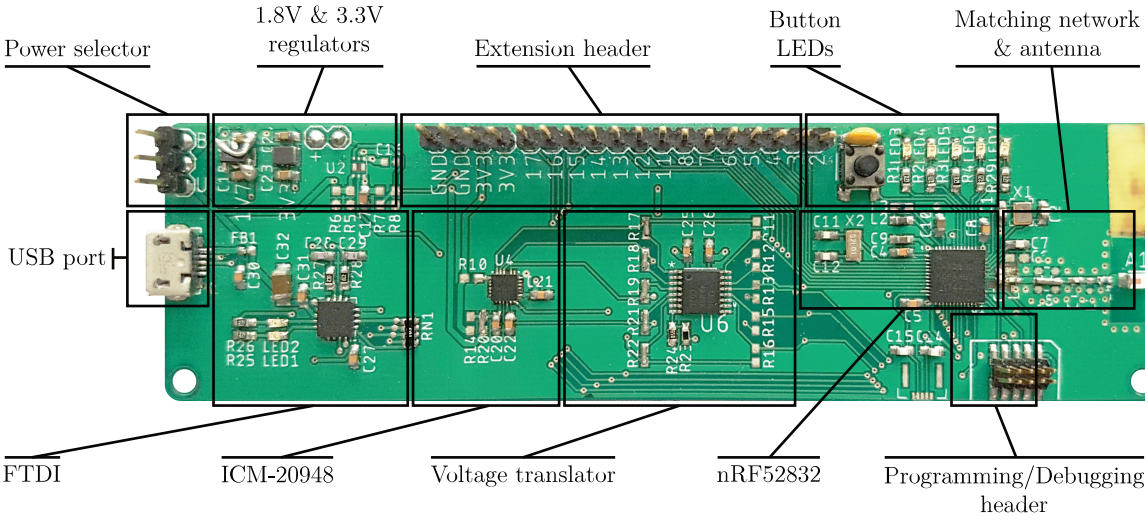


Figure 4.1. An image of the development board with labels of all its modules.

The accompanying hardware is presented as a block diagram in Figure 4.2. On the left there are two power sources: a battery and a USB port. They are connected to a power selector, which allows the power source to be selected.

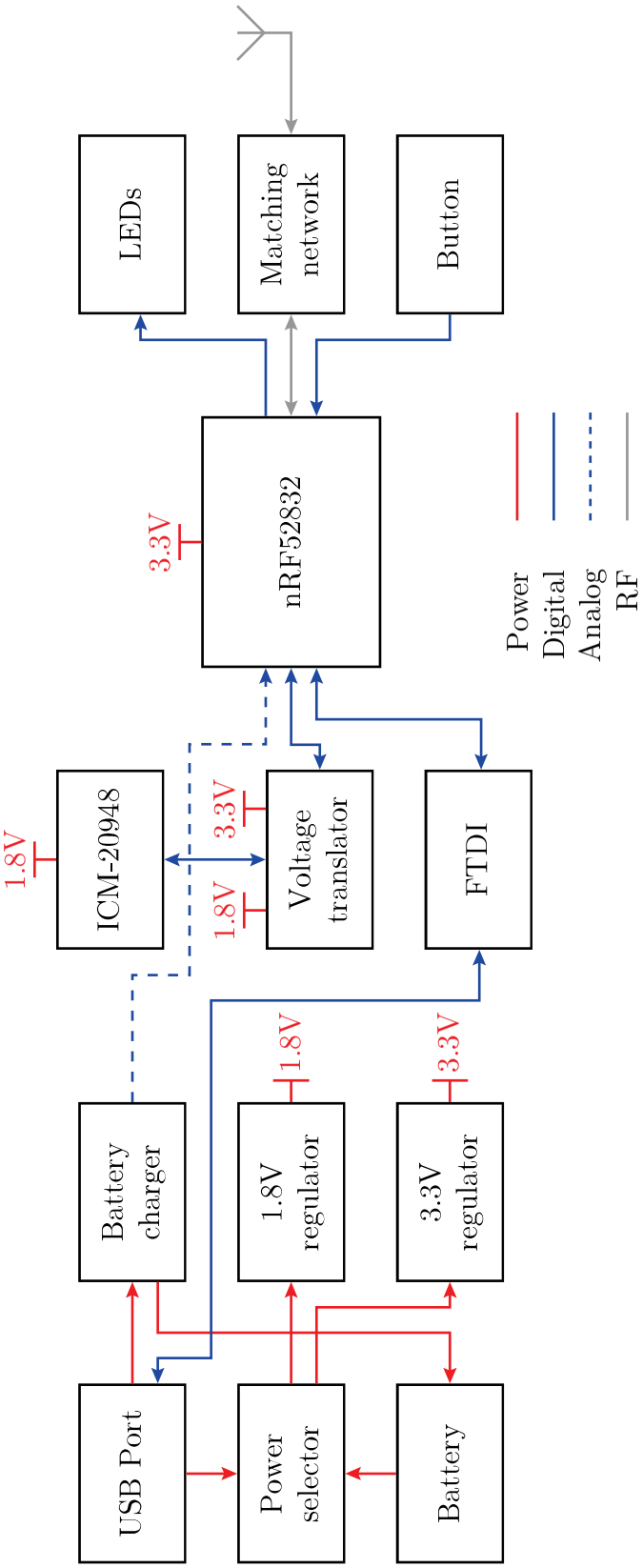


Figure 4.2. A logical block diagram of the development board.

The unregulated power from the power selector is then distributed to two switching voltage regulators: one for the $3.3V$ circuitry and one for the $1.8V$ circuitry. There is also a battery charging IC which gets power from the USB port.

The FTDI block resembles a USB-to-UART converter; this is achieved by the FT230X chip. The communication speed is limited to $1M\,Baud$ by the microcontroller. This is just enough to log the data from all sensors at their highest output rate.

Next in the diagram is the sensors block. It is realised by the ICM-20948 IC, which integrates tri-axial accelerometer, gyroscope and magnetic sensors. Since the operating voltage of this IC is $1.8V$, whilst that of the microcontroller is $3.3V$, there is a voltage translator in between. The translation is performed by the TXB0106.

On the right of the microcontroller there are 5 LEDs for status indication of the system and a button for fast input from the user.

Lastly, there is the 2450AT18B100 chip antenna which is connected through an impedance matching network to the microcontroller. The component values for the matching network were taken from the reference design documents of the nRF52832 [31]. Schematics of the development board and its stick mount are available in Appendix B.

4.2 The Pilot Board (Prototype 2)

Based on numerous tests and evaluation with the development board, a few hardware modifications were made that resulted in the pilot board.

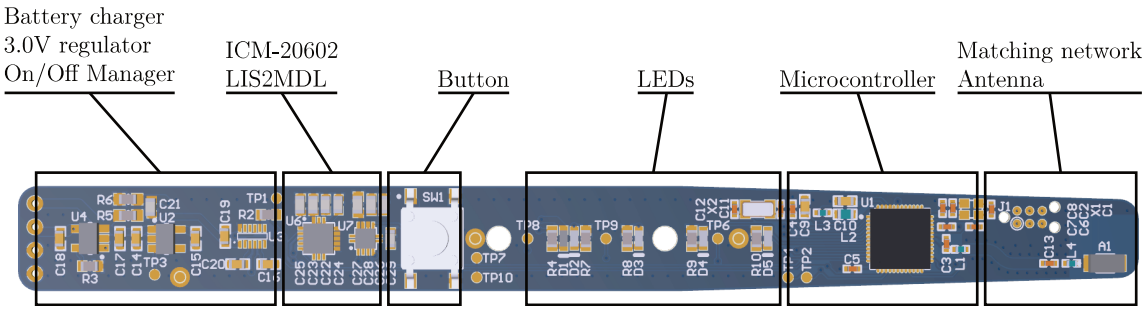


Figure 4.3. A rendering of the pilot board with labels of all its modules.

Since the aim of the pilot board is to only have the essential components and be as tiny as possible, all components for data logging and hardware expansion have been removed since they are not needed in the final product.

Apart from removed hardware, the pilot board also has some modifications when compared to the development board. The sensor IC on the development board

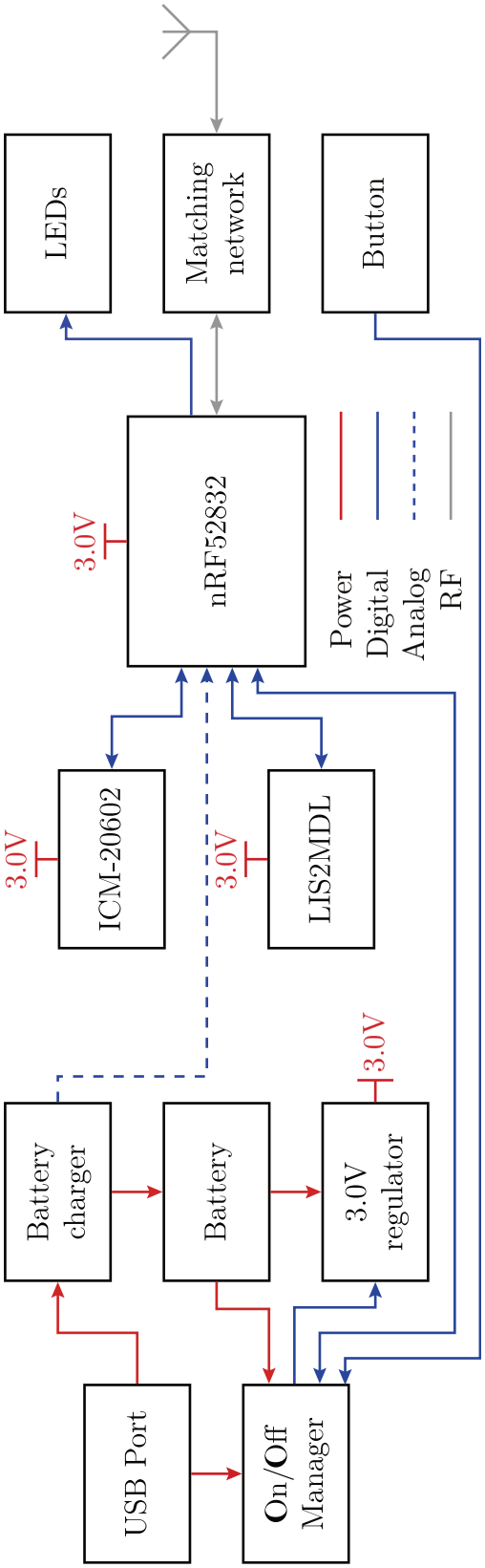


Figure 4.4. A logical block diagram of the pilot board.

required a $1.8V$ supply and communication with the compass required an SPI to TWI conversion, which was slow and unreliable. These problems were solved by replacing the ICM-20948 with ICM-20602 (accelerometer and gyroscope) and LIS2MDL (digital compass). The replacement sensors have the same characteristics as the original one but can be powered by a $3.0V$ supply and support SPI natively. As a consequence of the sensor replacement, the $1.8V$ regulator and the voltage translator were removed, which saves PCB area and cost.

Another major hardware upgrade in the pilot board is the power block. In the new board, power is only supplied from the battery. To allow long stand-by times, an on/off manager chip (STM6600) disables the voltage regulator when the device is not in use. It can be reenabled by pressing the button. The switching regulators from the development board were replaced by a linear one (MIC5501) since the switching noise could be observed in the sensor measurements. The regulated voltage was also changed from $3.3V$ to $3.0V$ because this way the internal microcontroller regulators are more efficient [32].

Schematics of the pilot board and its casing are available in Appendix B.

5 Drum Recognition

5.1 Strike Detection – the Signal Processing Approach

As mentioned in the background section, strike detection is a typical signal processing problem which includes some peak detection and filtering.

5.1.1 Simple Peak Detection

From calculus it is known that a peak in a signal (a.k.a. local extremum) occurs when the first derivative of the signal is zero. The derivative of a discrete signal can be found by calculating the average slope between three adjacent points (central-difference) [33]:

$$\dot{y}_t = \frac{y_{t+1} - y_{t-1}}{2\Delta x} \tag{5.1}$$

Then an algorithm can be implemented to monitor the sign of two adjacent measurements of the vertical acceleration's derivative; a difference in the signs means a zero-crossing (Figure 5.1).

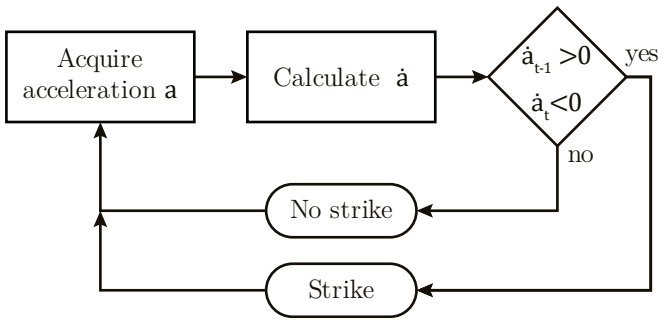


Figure 5.1. A state diagram of a simple peak-finding algorithm.

5.1.2 Improved Peak Detection

Although the algorithm above would find all strikes, it would also get triggered by noise and other motion of the drumstick that is not a strike but creates an acceleration peak.

Since noise has a much lower amplitude when compared to acceleration peaks caused by drumming, it can be filtered by applying a threshold to the amplitude of the acceleration data.

To filter out random, no strike motions, that generate peaks, the width of these peaks must be considered. Acceleration peaks due to striking will be much narrower than peaks from some random motion. Therefore, a threshold to the width of the peak can be applied to filter out the detection of random motion. According to [33],

the width of a peak can be inferred from the amplitude of the second derivative of the signal. This is shown in the simulated signal in Figure 5.2.

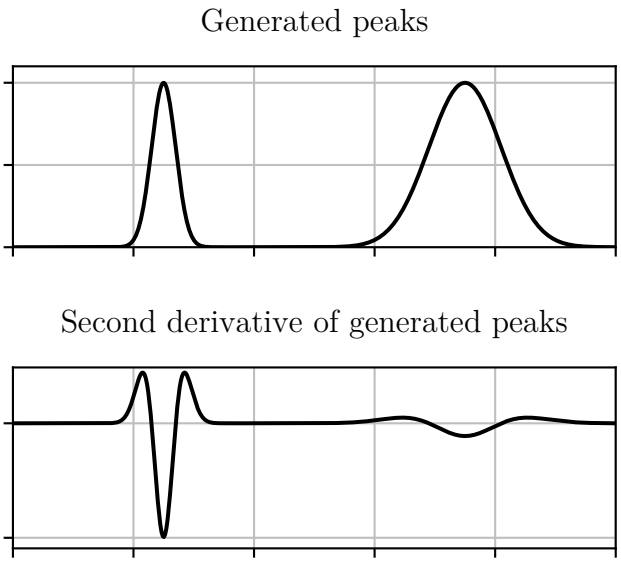


Figure 5.2. Simulated peaks and their second derivative. The derivative of the narrow peak is much bigger.

Lastly, sometimes the acceleration peak of a strike consists of a few smaller peaks (Figure 5.3). To filter this, a time threshold can be added, that sets a minimum allowed time between the detection of 2 strikes.

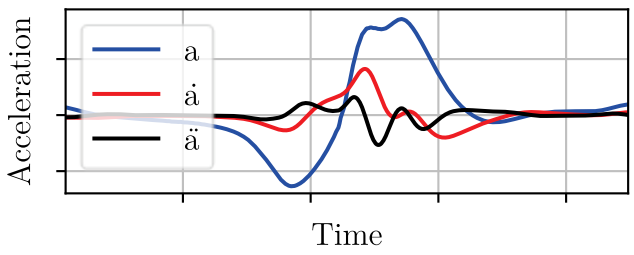


Figure 5.3. A strike peak consisting of two smaller peaks.

The state diagram of the improved strike detection algorithm is shown in Figure 5.4. This algorithm rejects peaks that are wide and are of low amplitude.

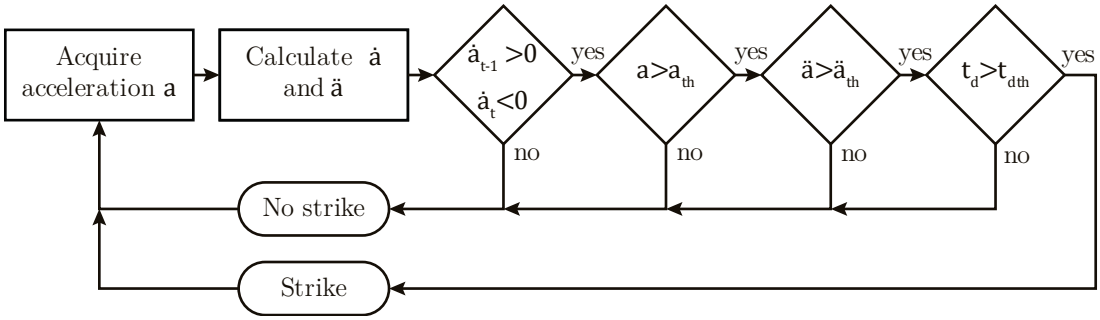


Figure 5.4. A state diagram of the improved strike detection algorithm.

The improved algorithm introduces three new threshold parameters that need to be tuned. By experimentation, it was found that the following values result in a robust strike detection:

$$a_{th} = 1.953ms^{-2} \tag{5.2}$$

$$\ddot{a}_{th} = 0.733ms^{-4} \tag{5.3}$$

$$t_{dth} = 20ms \tag{5.4}$$

5.1.3 Latency Improvement

It was noticed that when performing a striking motion with the drumstick, the peak in the angular velocity leads the peak in acceleration by nearly a constant time (Figure 5.5).

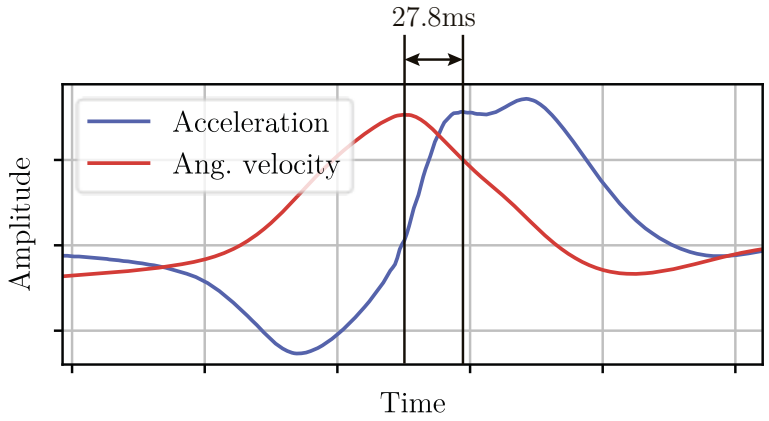


Figure 5.5. The time difference between the peaks of acceleration and angular velocity when performing a strike.

To get more insight, 100 strikes were performed and the time difference between the peaks of the angular velocity and the acceleration were recorded. It turned out that on average angular velocity peaks lead acceleration peaks by $27.8ms$ with a standard deviation of $3.76ms$. Despite the introduction of some jitter, using angular velocity for strike detection resulted in a more responsive system.

The algorithm for detecting angular velocity peaks is the same as in 5.1.2. The only difference are the threshold values:

$$a_{th} = 250°s^{-1} \tag{5.5}$$

$$\ddot{a}_{th} = 93.8°s^{-3} \tag{5.6}$$

5.2 Velocity Detection

The loudness of a sound can be inferred from the angular velocity of the drumstick prior to the strike. Through experimentation, it was found that this is best achieved by taking the value of the angular velocity at a peak that corresponds to a strike and map it to a range that is compatible with MIDI:

$$\overbrace{[a_{th}, 2047]°s^{-1}}^{Angular\ velocity\ range} \rightarrow \overbrace{[27, 127]}^{MIDI\ range} \tag{5.7}$$

The MIDI range is the same as a 7-bit unsigned integer. The first 27 values are skipped since they produce inaudible sounds.

5.3 Drum Detection

The type of drum that is struck is derived from the orientation of the drumstick. The orientation information is available from the sensor fusion library discussed in section 2.2.3 and is represented in Euler angles. The frame of the drumstick and the rotation axes are shown in Figure 5.6.

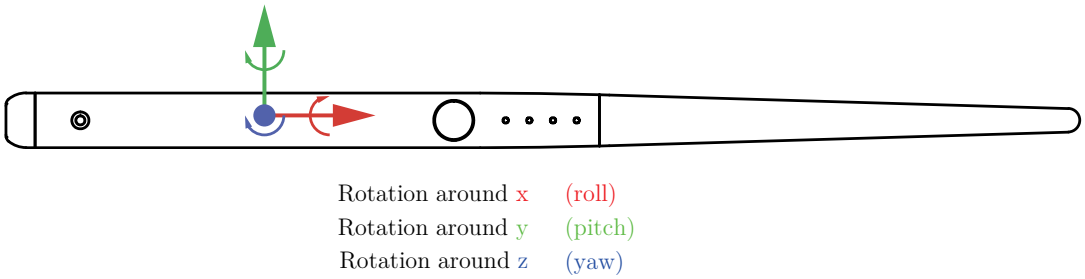


Figure 5.6. Representation of the drumstick's frame and definition of yaw, pitch and roll.

It can be seen from the figure above that the roll of the drumstick is irrelevant to the type of drum that is being hit. Based on the pitch and yaw values the configuration in Figure 5.7 was created. It provides all basic drums without being overwhelming. However, this configuration can be changed to the user's likings.

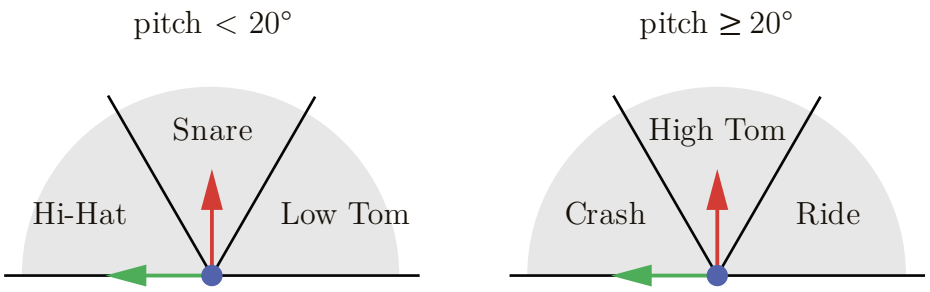


Figure 5.7. An example of drum zones. Each arc in the diagram is 60°.

6 Motion Classification Generalisation Using LSTM

This chapter is about work that tries to replace the specific strike detection algorithm from section 5.1 with a more general motion classification LSTM network. It proposes an innovation in labelling motion data and gives an example of the network with strike detection.

6.1 A Novel Approach for Labelling Motion Data

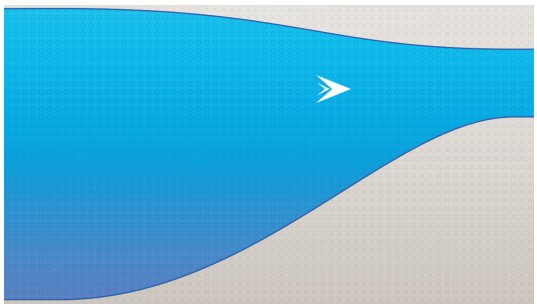
As mentioned in the background section, having much labelled data is crucial for training a neural network. However, it would be nearly impossible to collect and label much data manually. This section introduces a novel method for efficiently labelling motion data in the form of a computer game. Although the example given is specific to air-drumming, it will become apparent below how this can be easily extended to any type of motion.

This paragraph explains the core idea of the game. The character of the game is the white arrow in Figure 6.1 (top, left). Its position is controlled by the orientation of the drumstick (e.g. the drumstick must be tilted up, to move the character up etc.). The character constantly moves forward (right) in the game's map, and the player (the person that is controlling the character) must make sure that the character stays on the blue path by moving the drumstick. If the character goes out of the path, the game is over. This means that the shape of the path "controls" the way the player moves the drumstick – that is, in order to finish the game, the player has to move in a specific way. Therefore, by carefully designing the shape of the path, the player can be "forced" to perform the drumming motion in order to stay on the path and finish the game. Consequently, since the motion of the drumstick is known, the motion data can be labelled. On the other hand, if the player does a different motion, the character will get out of the path and the game will be over – the data will then be discarded.

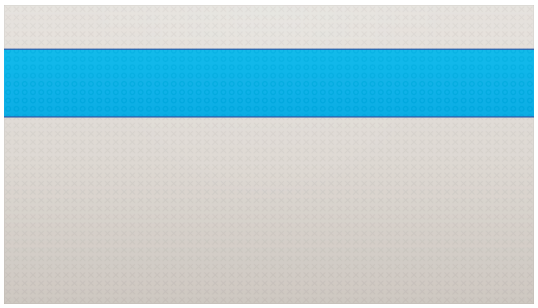
Since multiple sources [6, 18, 34] have confirmed that human activity recognition and strike detection using a neural network is possible with acceleration data only, only this data is labelled and used in this work. However, it would be trivial to adapt the code to label other data such as angular velocity, and orientation.

To allow easy modifications, the game is made up of sections (Figure 6.1). These can be rearranged to alter the length of the map, and the motions that the user must make. The only condition is that there should be only one start section and only one finish section, and these should be in the beginning and the end

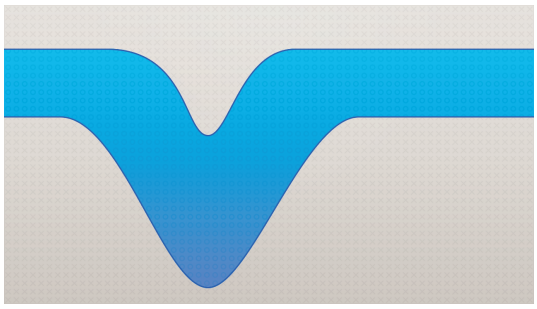
respectively. The purpose of these sections is to allow some time for the user to adjust the character's height, and not get out of the path immediately as the game starts. Therefore, data is not logged in these sections.



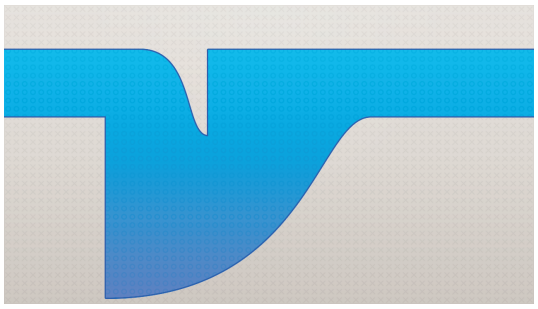
Start path



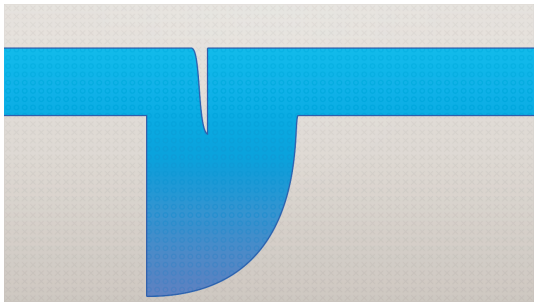
Straight path



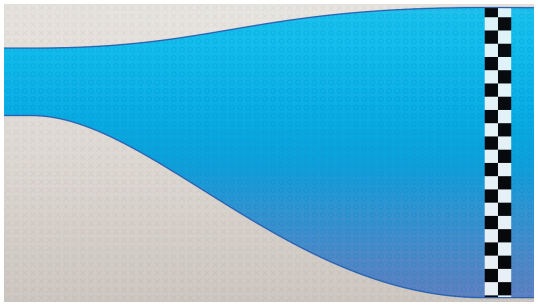
Curvy path 1



Curvy path 2



Curvy path 3



Finish path

Figure 6.1. All sections that can be combined to make up a level in the game. These sections have been designed to make the player mimic a drumming motion while playing the game. In theory, other path shapes can be added to track all sorts of motions.

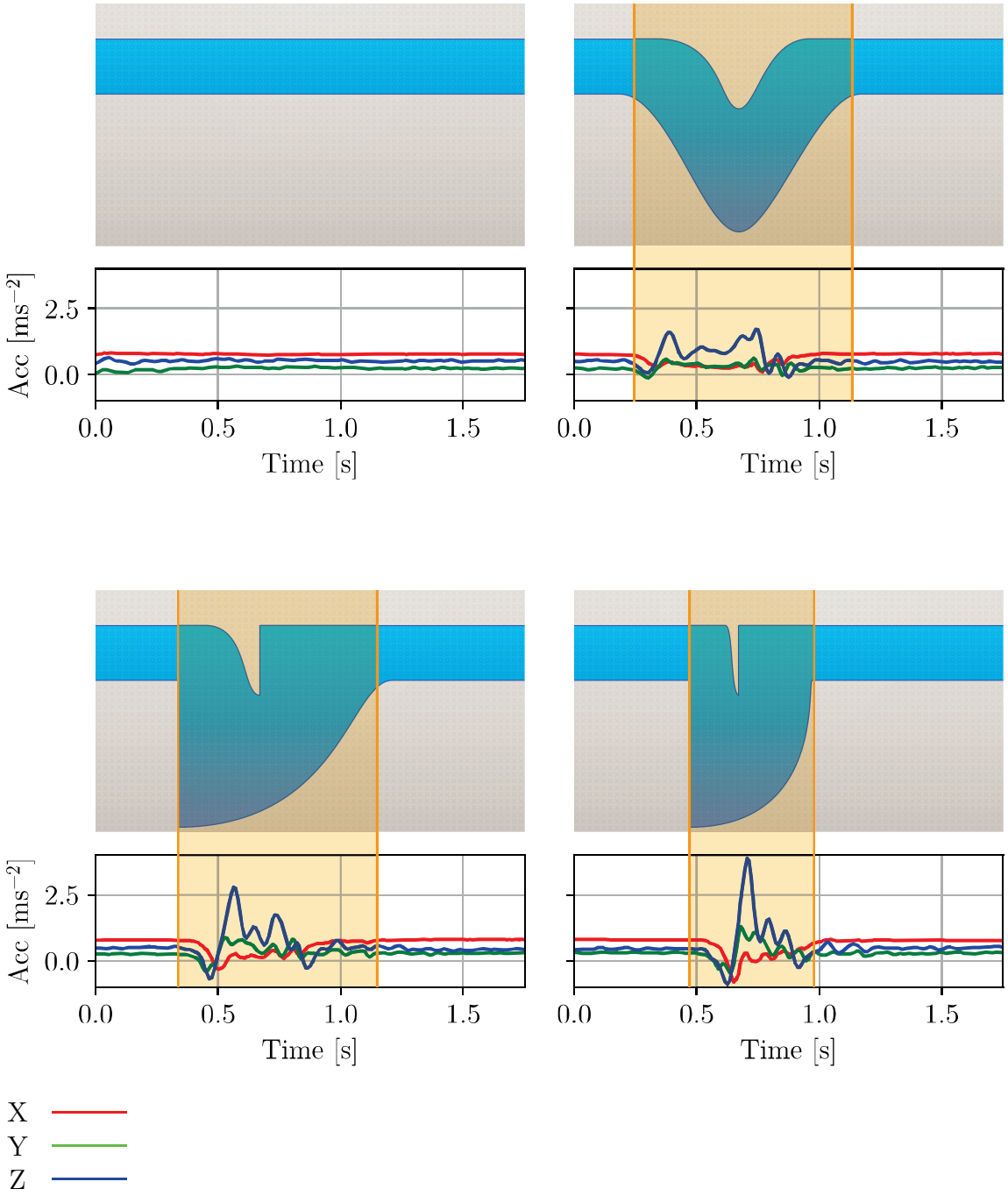


Figure 6.2. Plot of the logged acceleration data while passing certain sections of the game. The orange bounds represent the parts in which the acceleration data is labelled as "strike"; at any other time, the data is labelled as "no strike". The distinctive acceleration peaks of air strikes are clearly visible at the curvy paths.

Figure 6.2 shows when data is labelled as a "strike" and "no strike". When the character is in the straight path a "no strike" label is assigned to the data since the player is expected to keep the drumstick steady. In the other three cases a "strike" label is assigned when the character is inside the curvy part of the path since the player is expected to perform a drumming motion.

Curvy path 1 makes the player perform a smooth and slow strike with the drumstick. On the other hand, curvy paths 2 and 3 require a more aggressive strike.

The reason there are multiple curvy paths is that the neural network will be able to learn more types of striking.

To collect data for network training, 4 players were asked to play the game for a total of 2 hours. This resulted in 1100 labelled examples of acceleration data.

6.2 Strike Detection – the Machine Learning Approach

TensorFlow r1.13 was the machine learning platform of choice. The reasons for this are several: it is developed by Google and it is kept up to date with the latest developments in machine learning and artificial intelligence; it has a large community for support. The Keras frontend was used as recommended by Google.

6.2.1 Network structure

The first thing to be considered was the LSTM network structure. Although this can be iteratively derived through evaluation, a good starting point must be found. In a study [18], an LSTM network was used to perform human activity recognition. Through cross-validation, the researchers have found an optimal network structure (Figure 6.3) that achieved 96% classification accuracy. The network takes as an input the accelerometer readings from each axis through a linear activation. The inputs are then followed by two LSTM layers and a third fully-connected layer with a ReLU activation. At the end there are two outputs with a softmax activation. Therefore, the output of the network gives the probability of the input acceleration data to correspond to a strike. The hyperparameters[1] of the table are shown in Table 6.1.

Layer	N_{units}	Activation	Dropout	Regularisation
Input	3	Linear	N/A	N/A
LSTM 1	64	Activation: Tanh Recurrent activation: Sigmoid	N/A	l2 ($\lambda = 0.0015$)
LSTM 2	32	Activation: Tanh Recurrent activation: Sigmoid	N/A	l2 ($\lambda = 0.0015$)
Dense	16	ReLU	N/A	l2 ($\lambda = 0.0015$)
Output	2	Softmax	N/A	l2 ($\lambda = 0.0015$)

Table 6.1. Hyperparameters for the strike detection LSTM network in Figure 6.3.

[1] All parameters that can be tuned by the user within the network.

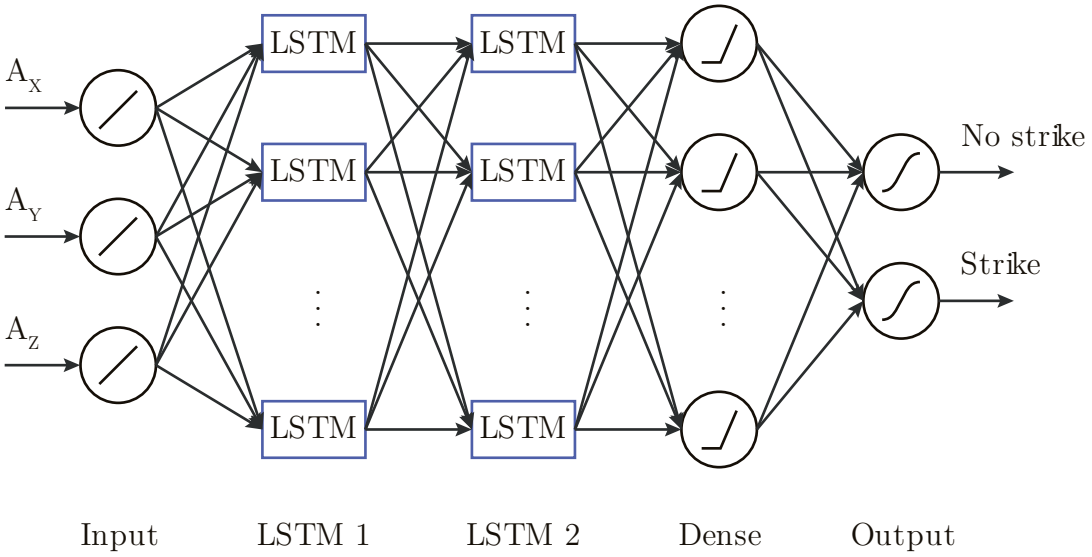


Figure 6.3. LSTM network structure.

6.2.2 Data Pre-Processing

In order to train the network, the data should undergo a slight formatting first. The formatting routine is visualised in Figure 6.4. In the first step, the logged data is sliced at the sections where the label changes from "no-strike" to "strike" and vice-versa. It can be seen that the resulting cuts of "no-strike" data are much longer than the rest. It is unlikely that this will bias the network in any way. However, it introduces extra data that does not bring any information. Therefore, without loss of accuracy, the data is trimmed in the next step. The maximum length of a cut is set to 125 samples (this is approximately the size of the longest "strike" sequence). At the last step, the data is grouped by label.

The result of the formatting is two groups of data: one containing examples of the striking motion, and one with examples of no motion. Since these groups are unbalanced – that is, the "strike" group has less examples than the other – the "no strike" group had randomly selected examples deleted to make the groups balanced. The reason for this is that an unbalanced dataset might cause biasing towards the bigger group when training a neural network.

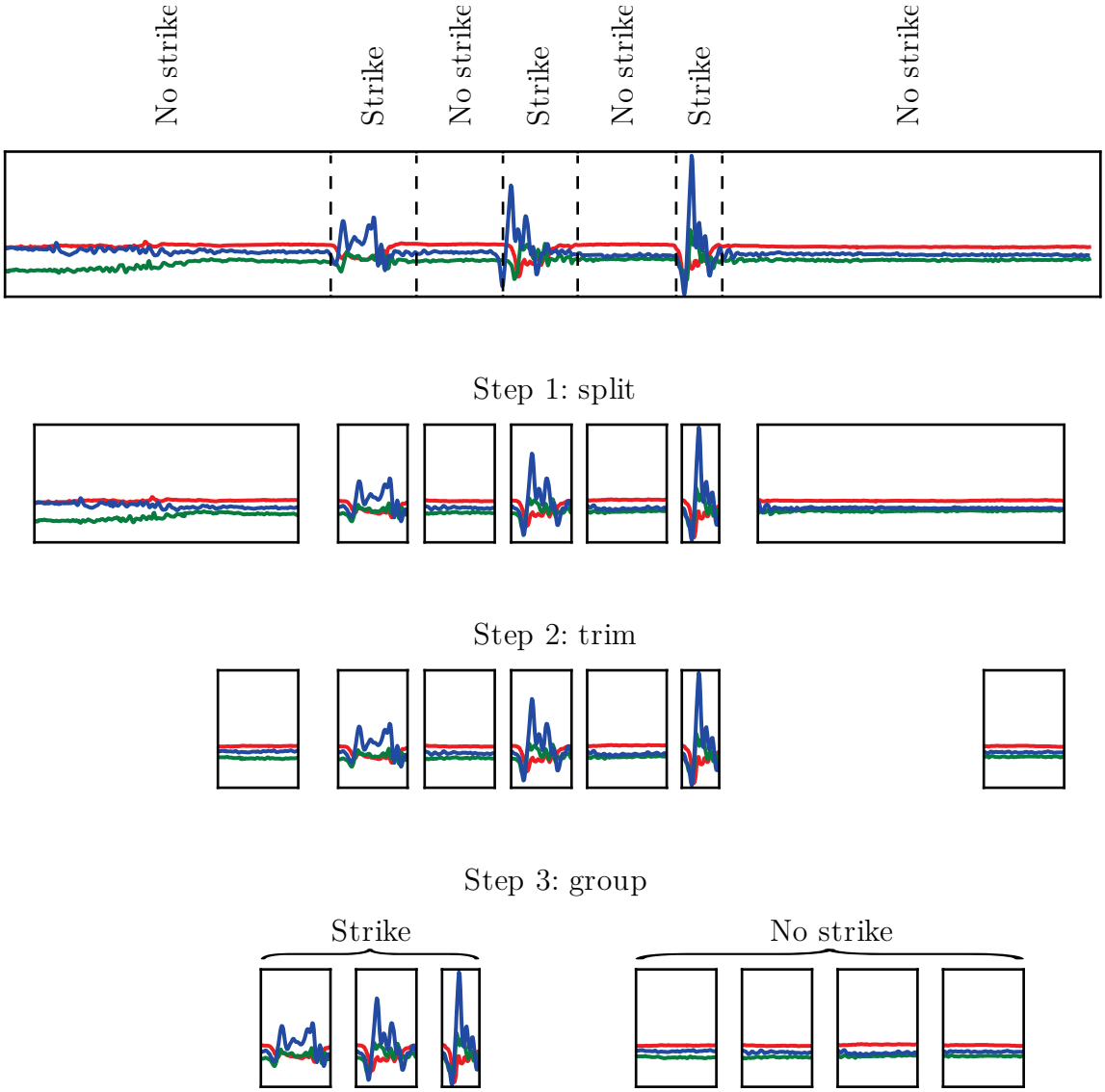


Figure 6.4. A visualisation of the data formatting process. The plotted data is the acceleration from the three axes against time.

6.2.3 Training

To train the network, 550 examples of "strike data" and 550 examples of "no strike" data were collected. They were split into a training set and a validation set with a ratio of $0.7 : 0.3$ and the data was randomly shuffled. The network was trained using stochastic gradient descent; this means that the weights were updated after each example.

The model mentioned above was compiled with the following parameters:

Loss	-	Categorical cross-entropy
Optimiser	-	Adam
Metrics	-	Loss, Accuracy

Table 6.2. Model compilation parameters.

The Adam optimiser adaptively changes the learning rate based on the first and second order moments of the gradients to speed up the convergence of the model [35]. The loss and accuracy were recorded at the end of each epoch so that convergence and overfitting can be tracked. Figure 6.5 shows that the losses converge to virtually 0, whilst the accuracy settles at 1; this means that the model successfully learned the data. The fact that there is no gap between the training and validation metrics means that the model has generalised well, and it is not overfitting.

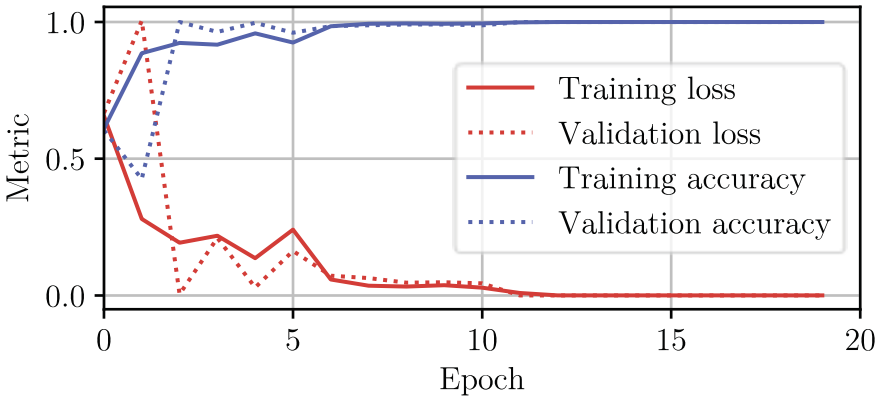


Figure 6.5. Plot of the model metrics with epochs.

Lastly, a confusion matrix was generated (Figure 6.6). It can be seen that the model correctly identifies the true class every time, without making any false classifications.

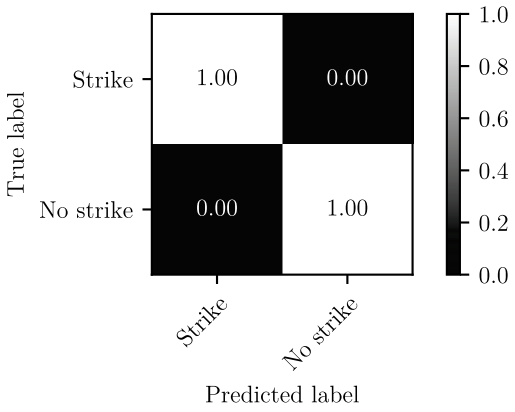


Figure 6.6. Normalised confusion matrix of the trained model. The model correctly identifies the class at every instance.

6.2.4 Pulse Generation

The network mentioned above can correctly classify drum strikes. However, the "strike" output of the network will stay high for a few iterations of the network when a strike is detected (from the beginning to the end of the strike peak), rather than creating a pulse signal which can trigger a sound. This can be fixed by stacking

a new pulse generation network (Figure 6.7) at the output of the one above. The output of this network only goes high on the rising edge of the "strike" output from the LSTM network.

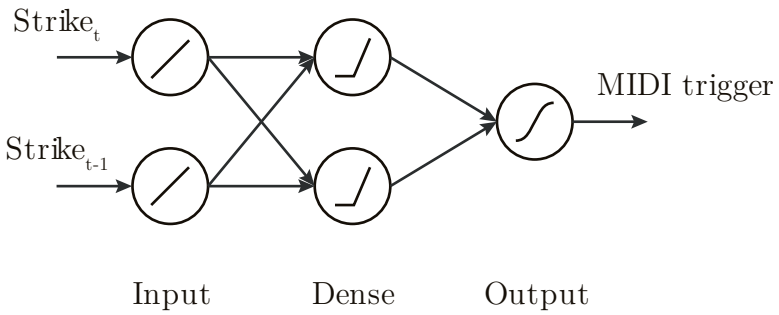


Figure 6.7. Structure of the pulse generation network. The activation of the Dense layer is ReLU; the activation of the output is a sigmoid.

This new network takes as inputs the "strike" output from the LSTM network above at times t and $t-1$. This way, it can track when the "strike" signal changes from low to high and output a pulse. To achieve the functionality above, the network was trained with the following artificially created data:

Strike_t	Strike_{t-1}	MIDI trigger
0	0	0
1	0	1
0	1	0
1	1	0

Table 6.3. Training data for the pulse generation network.

6.2.5 Increasing Performance

Since the neural network is used for real-time classification, it is important that the forward pass is as fast as possible. One way to achieve this is by reducing the number of mathematical operations (e.g. by reducing the number of units in the layers). It was found that by reducing the number of units in layers LSTM 1 and LSTM 2 to 16, the performance of the system increased by a factor of 8.2, while the accuracy did not change at all.

7 System Integration

7.1 Encoding for Data Logging

Data logging to a computer was required for numerous reasons: visualising sensor data, training a neural network, creating a peak-finding algorithm etc. The output rate of the data in this project is pushing the limits of the UART speed of the microcontroller. That is why an efficient communication protocol had to be created.

The microcontroller sends data to the computer in the form of frames. An example of a frame is shown in Figure 7.1.

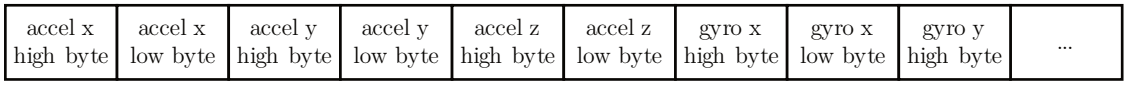


Figure 7.1. A typical frame from the microcontroller. Sensor data is usually stored in 16-bit variables, so a single sensor reading is split into 2 bytes.

Although it might seem trivial to reconstruct the data from that frame, it is much harder in reality since the computer does not know when a frame starts and finishes. That is why the data has to be encoded on the transmitting side and decoded on the receiving side.

A common approach for framing serial data is the "flag bytes with byte stuffing" method [36]. Flag bytes are predefined byte values that mark the beginning and the end of a frame (Figure 7.2).

Figure 7.2. The frame has been surrounded by a start flag and end flag bytes. (Adapted from [36])

A problem with this approach is that at some point the transmitted data will have the same value as the flag bytes. The solution is to prepend another special byte called an escape byte (ESC) every time a byte in the transmitted data is equal to the start flag, the end flag or the escape byte itself. This is illustrated below.

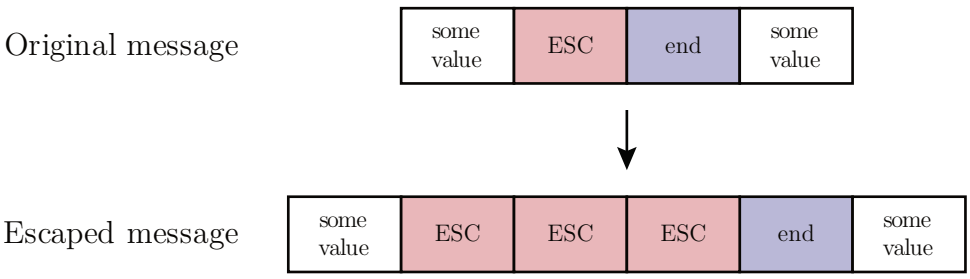


Figure 7.3. An example of a frame, that has ESC and the end byte as some of its values (top). The bottom shows how this message can be "escaped". (Adapted from [36])

By utilising this encoding scheme, the computer is able to synchronise with the data from the drumstick.

7.2 Producing a Sound

So far, the system has the ability to detect a strike, its velocity and the type of drum that was struck. However, the drumstick still has to inform a musical software on a mobile device to play a sound on the occurrence of a strike. The mobile device that was chosen for this work is the 2013 MacBook Pro, and the software that was used is GarageBand. An Apple product was chosen, since they generally have a much lower audio latency than other brands.

As specified in the background, the sound triggering signal was sent via BLE-MIDI. The microcontroller was set up as a BLE peripheral. A custom MIDI service was implemented according to Spockeli's tutorial [37]. The MIDI service and characteristic UUID[1]s are shown in the table below:

	UUID
MIDI Service	03B80E5A-EDE8-4B33-A751-6CE34EC4C700
MIDI Data I/O Characteristic	7772E5DB-3868-4112-A1A9-F2669D106BF3

Table 7.1. Service and characteristic UUIDs for BLE-MIDI [25].

7.3 Putting Everything Together

All functionality was integrated in two FreeRTOS tasks: an AFPT[2] task that is responsible for acquiring data, detecting strikes, velocity and drum type, and transmitting MIDI messages; a FUSE task that performs the sensor fusion. The need for two tasks comes from the fact that the sensor fusion takes longer to run than the required sampling rate of the sensors.

The AFPT task is activated by a data ready interrupt from the sensor at a rate of $1.125kHz$. The FUSE task runs exactly 4 times slower. Below is a time diagram of both tasks.

[1] Universally Unique Identifier

[2] Acquisition, Fast Processing, Transmission

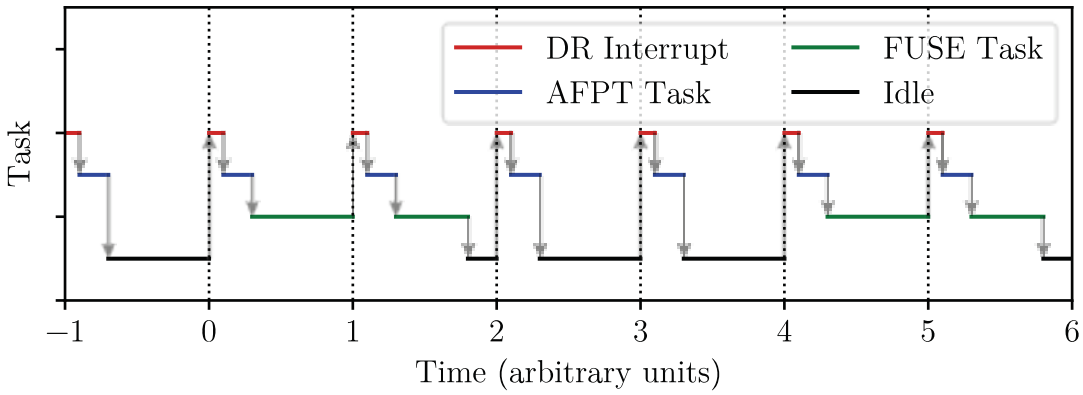


Figure 7.4. A time diagram of the RTOS tasks. Note how the data ready (DR) interrupt activates the AFPT task. Note also that the FUSE task runs 4 times slower. When there is no task running, the microcontroller is put into Idle mode to save power.

8 Testing and Results

8.1 Measuring Latency

As a reminder, the latency is the time between the occurrence of a strike and the playing of a drum sound. The definition of a strike was defined to be the sharp change in direction of the tip of the drumstick. This event causes a sharp peak in the acceleration of the drumstick.

To measure the latency, the system in Figure 8.1 is proposed. An additional analogue accelerometer (ADXL335) is added to the drumstick and its z-axis output is connected to an oscilloscope. The audio output from the laptop is also connected to the oscilloscope. The oscilloscope is set to trigger on the rising of the audio signal with a threshold of $8mV$. A single strike is then performed with the drumstick and the time between the acceleration peak and the time at which the sound starts playing is measured; this is the latency.

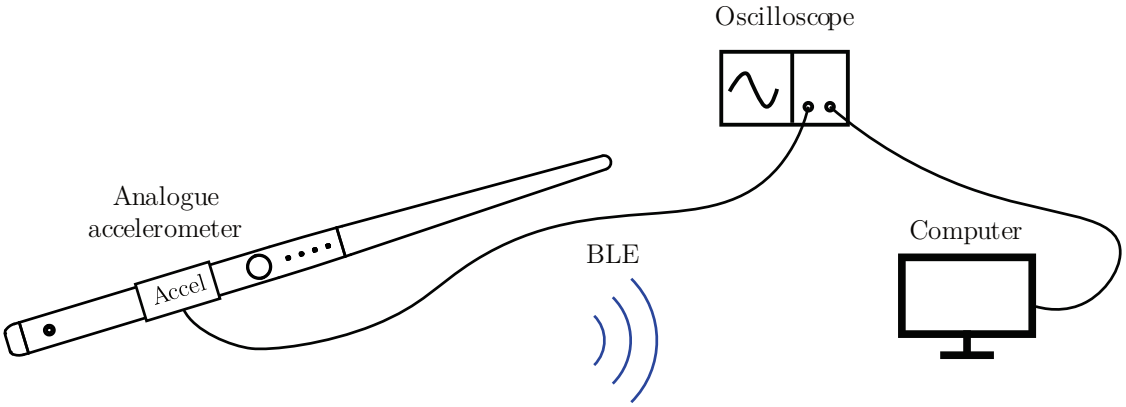


Figure 8.1. Proposed setup for latency measurement. The audio output from the computer and the z-axis output from the analogue accelerometer are connected to an oscilloscope.

Using the method above, 20 latency measurements were taken for both strike detection algorithms: the signal processing one, and the LSTM one. Unintuitively, in both cases the sound starts playing before the actual strike has occurred; the latency is negative (Figure 8.2).

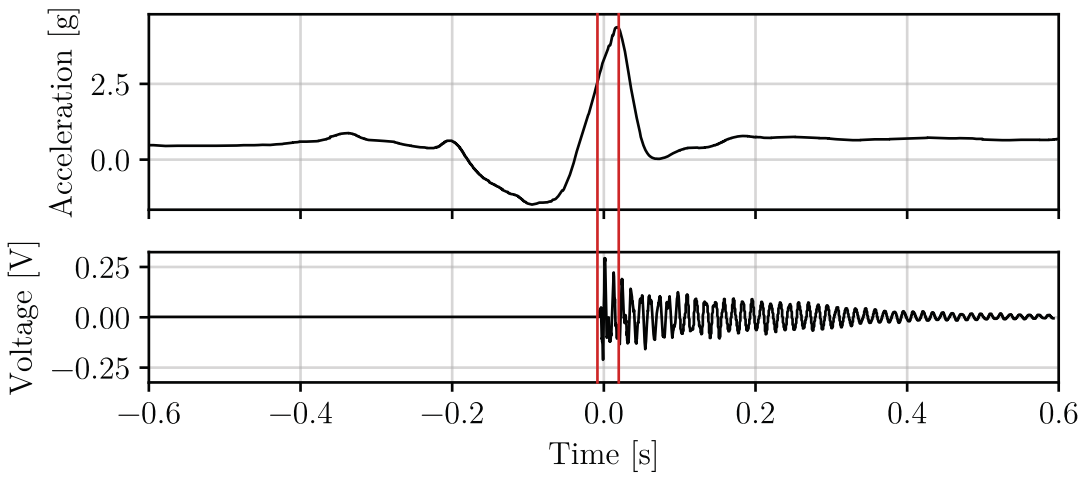


Figure 8.2. The time difference between an acceleration peak (strike), and the beginning of the drum sound. The sound starts playing before the strike, because the algorithm implementation predicts the strike.

The reason for this in the signal processing approach is due to the latency improvement technique that was introduced in section 5.1.3. By using that technique, the system was able to predict the hit $27.8ms$ before it occurs.

Unfortunately, there is no way to say with certainty why the negative latency occurs in the LSTM strike detection due to the complexity of neural networks. Most probably the reason for this is the fact that the "strike" label in Figure 6.4 captures the whole acceleration curve rather than just the peak.

It also worth noting that the signal processing algorithm has a much more consistent latency than the LSTM one (Figure 8.3). The signal processing algorithm achieved a mean latency of $-20.7ms$ with a standard deviation of $1.25ms$. This small latency jitter is due to the finite sampling rate of the sensor $1.125kHz$ and the resolution of the BLE-MIDI timestamp ($1ms$). The LSTM algorithm has a slightly higher mean ($-17.2ms$) and a much higher standard deviation ($7.2ms$). The higher metrics are due to the longer time to run the LSTM model ($3.77ms$) since it is significantly more computationally expensive than the signal processing algorithm. What is more, it is uncertain at which point the LSTM model decides that a strike has occurred, which contributes to the higher standard deviation.

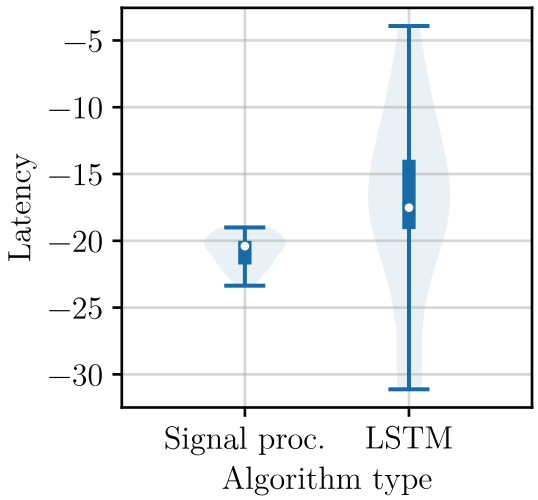


Figure 8.3. Latency distribution of 20 strikes from both strike detection algorithms.

8.2 Measuring Predictability

The system was tested for predictability of strike detection, velocity detection and drum detection by generating three sets of 100 strikes and counting the number of times when the strike did not produce the intended sound.

For the strike detection test, the tester was asked to perform 100 strikes on the snare drum with a medium velocity. The resulting predictability was 100%.

For the velocity detection test, the tester was asked to perform 50 strikes of steadily increasing and then decreasing velocity; 35 strikes with a velocity pattern 3 x “low”, 1 x “high”; and 15 strikes with a randomly chosen velocity on each strike. The resulting predictability was 100%.

For the drum detection test, the tester was asked to perform 50 strikes with the pattern 3 x “hi-hat”, 1 x “snare”; then 50 strikes with the pattern “high tom”, “snare”. The resulting predictability was 97%.

8.3 Battery Life

The battery life of the system was not measured quantitatively so this section will try to estimate it. The battery used in the drumstick has a capacity of $130mAh$. The microcontroller draws on average $0.417mA$ [38], the accelerometer and gyroscope together draw 2.790mA [39] and the compass draws $0.200mA$ [40]. The total draw is $3.407mA$. Then according to (8.1) [41], the battery should last 26 hours of continuous usage. Implementation imperfections are taken into account with the 0.7 factor.

$$t_{bat}[h] = 0.7 \frac{capacity\ [mAh]}{load\ [mA]} \tag{8.1}$$

9 Critical Evaluation

The outcome of the project greatly exceeded my expectations. When compared to all previous work in this field, the proposed system outcompetes their results in every aspect. The fact that even professional drummers confirmed the exceptional performance of the device and learnt to use it fluently within minutes, means that a solid solution has been found for an air drumming system. Moreover, the work introduced a completely novel approach for labelling motion data.

9.1 Portability

Unlike most of the previous attempts [4, 6], the proposed system in this work is a standalone device, that has its own integrated microcontroller and a battery. This allows the system to be easily extended to multiple drumsticks and even drum pedals without any modifications.

The resulting PCB is extremely compact, and it can be fit inside a drumstick – this is something no one else has yet achieved. What is more, the whole system is extremely power efficient, and in theory it can last over a week[1] on a single charge (although my testing unit lasted 2 months).

These factors makes the air drumming system a compelling substitution for real drums. It is also an excellent option to carry around on tours and trips, since it can fit in a backpack and it may not even need a charger depending on the duration of the trip.

9.2 Latency

Unexpectedly, the system has a negative latency, which is another precedent in the field. This means that it can predict a strike and play the sound before it has actually occurred. This is largely beneficial for many reasons. Firstly, it allows the system to be used with cheap mobile phones and laptops that have inherently high latency. Furthermore, it allows vibrational feedback to be utilised. Normally, haptic feedback actuators such as piezo actuators and linear resonant actuators have an actuation latency of $4 - 20ms$ [42]. The strike prediction will allow these actuators to start vibrating at just the right time.

[1] Assuming the device is used approximately 3.5 hours per day.

9.3 Predictability

All previous work done has had significant issues with predictability, making their solutions virtually unusable in a professional environment. My device achieved 100% predictability in strike and velocity detection and 97% in drum detection. The 3% error in the drum detection is due to imperfections in the sensor fusion algorithm which was not developed as a part of this project but rather an off-the-shelf library from InvenSense was used. With some extra work on a better fusion algorithm, the proposed system would easily become suitable for live performances.

9.4 LSTM Network

The proposed LSTM network achieved great results with strike classification with accuracy of 100%. However, the data that was presented to train the network corresponded to a limited range of orientation of the drumstick and thus the network could not recognise strikes outside the trained range. An extension to the game, including more section which cover a bigger range of orientations, will likely solve this issue. However, this might also require the complexity of the network to be increased and consequently a more powerful microcontroller might be needed.

The more important implication however is the introduction of the data labelling game. Collecting much labelled data is a big challenge in machine learning and my process is an innovation in the field, which can be extended to applications far beyond drumming.

9.5 Further Work

As mentioned above, the only part of the air drumming device that is not always accurate is the drum recognition. A suggestion for future work would be the development of a custom fusion algorithm based on some of the common architectures [11, 12, 13].

More importantly however, there is plenty of room for expansion. The presented work reveals not only a complete and professional air drumming device, but also a platform for motion detection based on machine learning. There are endless possibilities from creating unseen musical instruments to fields far beyond music such as sports, medicine and robotics. Future work would implement other types of neural networks and other games that will allow the labelling of any type of motion.

9.6 Project Management

Thanks to good project management, I managed to take the project beyond the initial planned work and achieve excellent results. The rough plan of work was set out in a Gantt chart in the beginning of the project together with a contingency plan (Appendix H). This plan was definitely not a big contributor for short-term decisions. However, it was good for indicating when the completed work was leading or lagging schedule and it allowed for task rearrangements and including extra work after a discussion with my supervisor. Most work went fluently, apart from a one month delivery delay due to a mistake of the ordering team. After quick rescheduling, this made no negative impact.

To protect the work from hard drive failure or other data loss, all work was automatically backed up online on MEGA and locally on two computers. Furthermore, monthly back-ups on a hard drive were done. Most code was also stored on private repositories on GitHub. CAD work was backed up remotely on Autodesk's servers.

10 Conclusion

A complete, portable air drumming solution was provided in this work that achieves professional level of performance. The proposed system utilises MEMS gyroscope, accelerometer and a compass integrated into a drumstick to perform motion tracking. Thanks to the utilisation of an SPI connection between the microcontroller and the sensors, the proposed device achieves an update rate of $1125Hz$. Two algorithms were introduced to perform strike detection. The first one – a signal processing based, peak-finding algorithm allowed extremely consistent, low latency strike detection, when applied on the acceleration data from the drumstick. The latency was further decreased by $27.8ms$ when this same algorithm was applied to the angular velocity of the drumstick. This improvement essentially allowed the prediction of drum strikes before they occur. The second algorithm is an LSTM based neural network for general motion detection from acceleration data. To train it, a game was developed, that introduced a novel approach for efficiently and automatically labelling much data. Although with a slightly higher latency and more jitter, the LSTM network still performed well. More importantly, it showcased the ability of the innovative labelling method.

The finished system comes in a compact form factor of $10 \times 113mm$. This allows it to be placed inside a drumstick or an attachment for a foot pedal, making it extremely portable.