



M Ű E G Y E T E M 1 7 8 2

Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Távközlési és Médiainformatikai Tanszék

Horváth Márton Áron

Arm Movement Recognition with Deep Learning

Consultant

Dr. Fehér Gábor

Budapest, 2020

Contents

Kivonat.....	3
Abstract	4
1. Introduction.....	5
2. Technological background.....	6
2.1 Microcontrollers	6
2.2 Sensors	6
2.3 Neural networks	6
3. Hardware construction and programming.....	7
3.1 Select the hardware components	7
3.2 Hardware programming	10
3.2.1 How to Initialize the device	10
3.2.2 Read value from MPU6050	13
3.2.3. What the microcontroller needs to use?	15
3.3 Read the data when an event happened	16
4. Data processing, and preparation of the samples for learning.....	18
4.1 Selection of samples.....	18
4.2 Receive samples on the PC.....	19
5 Neural network and teaching.....	23
6 Problems with implementation on a microcontroller.....	26
7 Experimenting with efficiency.....	34
7.1 Efficiency of model2	35
7.2 Efficiency of model3	36
7.3 Efficiency of model4	37
7.4 Efficiency of model5	38
7.5 Evaluation of results	39
8 Frequency reduction tests.....	40
8.1 Efficiency of model6	42
8.2 Efficiency of model7	43
8.3 Evaluation of the results.....	44
9 Conclusion	47
Bibliography.....	48

Kivonat

Számos technológia hozzájárul a mai világunk modernségéhez, szeretnék három területet kiemelni, amik nagyban hozzájárultak a technológiai fejlettséghez. Az első az AI (Artificial Intelligence – Mesterséges Intelligencia), ami a hétköznapi életünket több aspektusban is megkönnyíti. A mikrokontrollerek, amik mindenhol ottvannak, szinte mindenben. És végül az IoT (Internet of Things- Dolgok Internetete), ami azt jelenti, hogy a tárgyak körülöttünk adatot gyűjtének, információval látnak el, és egy kis feldolgozási munkálat után hasznos következtetéseket vonhattunk le a gyűjtött adatokkal.

De mi lenne, ha lenne valami, amivel kombinálni lehetne ezen három technológiát és elvinni egy teljesen másik irányba? Habár az IoT képes nagymennyiségű hasznos adatot gyűjteni, amit aggregálás után analizálhatunk a felhőben, majd ezt követően konklúziót vonhatunk le. A kommunikációval bithibákat okozhatunk és lelassíthatjuk a döntési eljárást.

De mi lenne, ha az adatok feldolgozásának egy részét helyben végeznénk el? Ha a dolgok körülöttünk nem csak mérnének és mentenének adatokat, hanem fel is dolgoznák, és egy if-else ágban komolyabb döntések meghozására lenne képes? Az alkalmazások megpróbálja a felsorolt technológiák előnyeit kihasználni. Amit létrehozok az gyakorlatilag egy kézmozgásfelismerő, amely különböző mozdulatok felismerésére alkalmas. Két gyorsulásmérő és giroszkóp szenzort használ fel ehhez.

Kiolvashatjuk az adatokat a szenzorból és kimenthetjük a számítógépre. Habár a számítógépeknek határtalan számítási kapacitása van a mikrokontrollerekhez képes, a mikrokontrollerek is nagy teljesítménybéli fejlődésen mentek át, néhány éve még elképzelhetetlen volt, hogy mesterséges neurális hálót futassunk egy egyszerű mikrokontrolleren. De a technológia eleget fejlődött, hogy lehetővé tegye.

A munkám megpróbálja kideríteni, hogy hogyan lehet mesterséges neurális hálót futtatni mikrokontrollereken, mik a lehetőségeink. A mikrokontrollerek limitált memóriával rendelkeznek, mégis mennyi memória szükséges az egyes megoldásokhoz? Mennyi hatékonyságot kell feláldoznunk ehhez?

Az dolgozatomban ezekre a kérdésekre választ fogok adni. Kifejtettem különböző speciális mérési eredményeit a jól ismert mélytanuló algoritmusoknak, mint például a TensorFlow, és megmértem, hogy mennyi hatékonyság veszett oda a mikrokontrollerre optimalizált verzió használatával. Adatokat szolgáltatok arról, hogy az egyes megoldások mennyi memóriát használnak fel és hatással voltak-e a sebességre akár negatív, akár pozitív irányba.

Abstract

Plenty of technology has contributed to the modernity of our world today, I would like to highlight three different, which are playing a huge role in technological advancement. The first is AI (Artificial Intelligence), which has made everyday life easier in many aspects. The microcontrollers, which are almost everywhere, and they are in everything. And finally, the IoT (Internet of Things), which means, the objects around us collect data, provide us with information, and after doing a little processing task, we can make useful conclusions with the gathered data.

But what if there was something that could combine these three technologies and take them in a whole new direction? Although IoT can collect a wealth of useful data, which can be analyzed after aggregation in the cloud, and after then we can make useful conclusions. But it takes a lot of bandwidth, we must collect a lot of data and process them in the cloud. With communication, we can cause bit errors and slow down the decision-making process.

But what if some of the data processing has already been done locally. If the objects around us could not only measure data and save, but also process it, and make more complex decisions, than if-else statements? My application seeks to take advantage of these benefits of the listed three technology. What I create is actually a hand motion recognizer, and that can detect different gestures. It uses two accelerometer and gyroscope sensors for this.

We can read the data out of the sensor, and save it on the computer. Although the computer has limitless counting capacity compared to the microcontrollers, microcontrollers had a huge performance improvement too, a few years ago, that was unimaginable, to run an ANN (Artificial Neural Network) on a simple MCU (Microcontroller Unit). But technology has improved enough and it has become possible.

My work would like to figure out, how can we run an ANN on microcontrollers, what are the solutions, the microcontroller has limited memory, how many memory each solution requires? How much do we need to let go of efficiency?

In the dissertation, I gave the answers to these questions. I described specific measurement results of the effectiveness of a well known deep learning algorithm, such as TensorFlow, and I measured how much efficiency get lost with a microcontroller-optimized version. I provided data on how much memory each solution requires and whether it affected speed, either in a positive or negative direction.

1. Introduction

Microcontrollers play a major role in the technological advancement of our world today, they are almost everywhere, and although they are outnumbering traditional computers, we can find them in almost every electrical device around us. Whether it is a microwave oven, a washing machine, or a more serious embedded system, the performance development of microcontrollers opened a lot of new ways for technology. Plenty of already imagined concepts became feasible, which was a pre-existing concept, only the right resource requirements were not given for implementation. An example is IoT, a concept coined by Kevin Ashton in 1999, although the first IoT device is said to be a cola machine on the campus of Carnegie Mellon University, the performance of microcontrollers at the time was insufficient to implement the entire concept. Today's high-performance and energy-saving microcontrollers have given way to the rise of sensors and sensor networks.

Another crucial technology is Artificial Intelligence. It can be found in more and more places today, it is starting to replace people in many areas, for example, there are websites where chatbots have replaced the role of support staff, that makes everyday life easier. It can also be divided into several groups in the field of artificial intelligence. Artificial intelligence is the widest concept, which, although it has several definitions, includes all artificially created, consciously shaped intelligence. There are several types, variants of which require continuous intervention to achieve learning, but there are variants of which take place without human intervention. One branch of AI Artificial Intelligence is Deep Learning, a category within Machine Learning. The special feature of Deep Learning is that it tries to virtually mimic the neurons found in the human body and thus the learning can be realized.

Both technologies are the flagships of the modernity of our modern world today. Undoubtedly many other things are responsible for the existence of the technological revolution, but these two technologies are tremendously advancing, and there are still plenty of untapped areas to explore. There are many applications that we already consider essential and based on these technologies, such as the pedometer in our phone, the activity sensor on our smartwatch that tells us whether we are sitting, standing, walking, running, or possibly climbing stairs.

In our project, we would like to use two sensors with gyroscopes and accelerometers, and with the data, we would like to predict the movement of the arm. It can be very useful in many areas.

If we can categorize certain events, take the right number of samples that can be well separated from each other, then coupled with artificial intelligence, a plethora of applications can be implemented. The primary role of artificial intelligence in our case is to be able to assign labels to certain situations, of course, this is just a very small slice of artificial intelligence, but this is the main purpose and main application of deep learning.

2. Technological background

I have divided the technological background section into three different areas, which include microcontrollers that perform calculations. Sensors, which make a connection with the outside world. And finally, neural networks, which, like many other engineering applications, virtualize a biological function, and implement it artificially.

2.1 Microcontrollers

The fundamental difference between microcontrollers and microprocessors is that the microprocessor, which is also found in computers, requires even more peripherals, while the microcontroller is a microprocessor-based control system built on a single chip. It is small and practical, but capable of everything a microprocessor can do and even contains the necessary peripherals (however, its clock speed is usually slower, and its memory is smaller).

2.2 Sensors

By its broadest definition, a sensor is a device, a module, a machine, or a subsystem designed to detect events or changes in their environment and transmit this information to other electronic devices. The sensors are used in everyday objects such as touch-sensitive lift buttons (tactile sensors) and lamps.

With the development of microcontroller platforms, the use of sensors goes beyond traditional temperature, pressure, or flow measurement ranges. Besides, analog sensors such as potentiometers, thermistors, and force-sensing resistors are still widely used. They are used in many fields, such as manufacturing, aerospace, space, automobiles, medicine, robotics, and many other areas of our daily lives.

Several other sensors measure the chemical and physical properties of materials. Some examples are optical sensors for measuring refractive index, vibration sensors for measuring the viscosity of a liquid, and electrochemical sensors for checking the pH of liquids.

2.3 Neural networks

Artificial Neural Networks (ANN) are computer systems inspired by the biological neural networks that make up the brain. Such systems "learn" to perform tasks by considering examples, mostly without programming the specification of the task. For example, in image recognition, they can learn to identify images of cats by analyzing sample images labeled cat or "not cat" and using the results to identify cats in other images. This is done without a priori knowledge, do not function like our brains, for instance, cats have fur, tails, whiskers, and cat-like faces. Instead, they automatically create identifiers from the processed examples.

3. Hardware construction and programming

The work can be divided into two parts. The first part is building and programming the hardware. The right devices must be selected and then the right data must be extracted from the sensors, and obviously in the right format.

The second part is that we need to create a model by using the extracted data, then test the model with new samples, and finally we have to make it all possible to run on a microcontroller.

3.1 Select the hardware components

To implement the application, we need a hardware that can measure the movements and store and process the measured results. Once everything is working properly, ANN should also be run on the hardware.

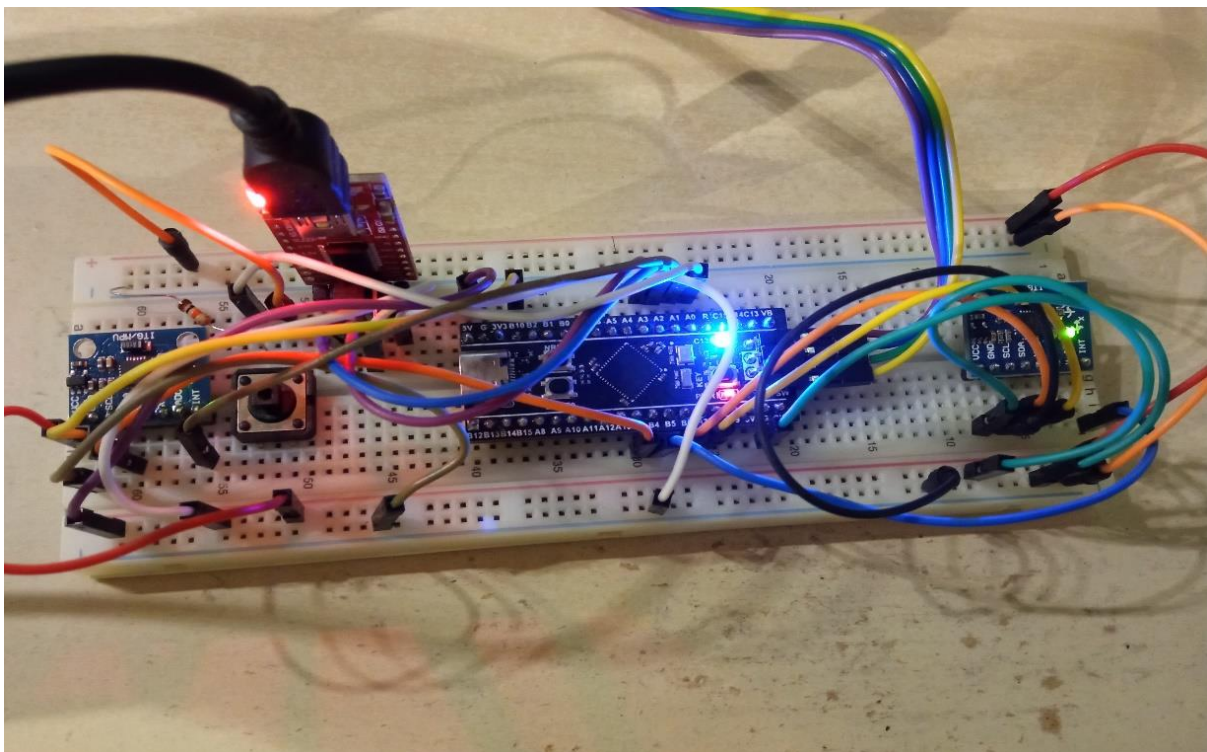


figure 1 The built circuit

We only need a limited number of hardware components for the development. First, we will need an adequate microcontroller that will perform the necessary calculations and the required controlling functions for us, and so on. Two pieces of sensors, which will perform the measurements. And finally, other axillary elements such as wires, a serial-to-parallel converter, and a programming device, which is suitable for our microcontroller.

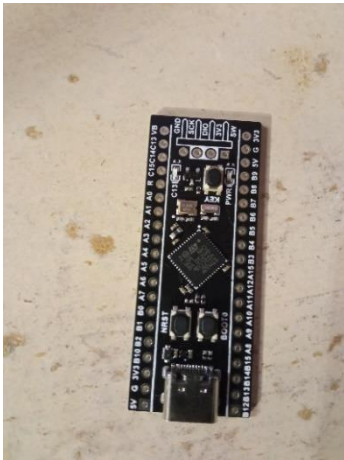


figure 2 STM32F401CCU6



figure 3 MPU-6050 (front)



figure 4 MPU-6050 (back)

The most important component is the microcontroller, which is the brain of the application. I have chosen a printed circuit board with a STM32F401CCU6 microcontroller (figure 2 STM32F401CCU6), because it has a small size, unlike the most popular development boards in STM series, like Nucleo and Discovery boards. The MCU (Micro Controller Unit) has the following parameters:

- Max clock speed: 84 MHz
- Flash: 256 Kbytes
- SRAM: 64 Kbytes
- Up to 3 × I2C interfaces (1Mbit/s, SMBus/PMBus)

The MCU has all the requirements, it has multi-channel DMA, and multiple lines of I2C, the only deficiency is the small size of memory. That is the reason, why I needed to change it to a little bit stronger version later, after I recorded the samples, because that memory size was insufficient for the ANN (Artificial Neural Network).

As a sensor I have chosen two MPU-6050 (figure 4 MPU-6050 (back)³, figure 3 MPU-6050 (front)), they have gyroscopes and accelerometer sensors, which means we will have 12 different data: x, y, z axes of gyroscopes, and accelerometer sensors.

Sensor parameters:

- Gyro full scale range: ± 250 , ± 500 , ± 1000 , ± 2000 ($^{\circ}$ /sec)
- Accel full scale range: ± 2 , ± 4 , ± 8 , ± 16 (g)
- 400kHz fast mode I2C for communicating with all registers.

Now we have all the necessary components for the measurement, but somehow, we should program the microcontroller. We have more option, I have chosen the official programmer (figure 5 ST_LINK V2 Programmer), called ST-LINK V2, but we could solve the problem with serial-parallel converter, too.



figure 5 ST_LINK V2 Programmer

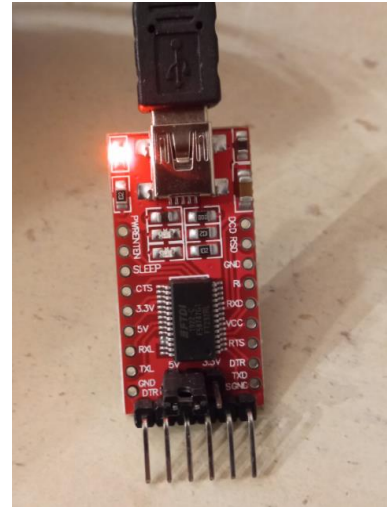


figure 6 Serial parallel converter

Wiring is very simple:

- GND – GNG
- SWCLK – SWCLK
- SWDIO – SWDIO
- 3V3 – 3V3

The last component is the serial-parallel converter (figure 6 Serial parallel converter), because we would like to send data to the computer, we use the UART communication protocol for that purpose. We need to send data to the computer, because we will build the ANN on the computer side, which means we need to send data to computer, prepare it for the network and build the model. Once the model is ready, we no longer need to send the data to the UART, but we still need to send the result of the prediction.

Wiring is the following:

- GND – GND
- VCC – VCC
- TX – USART_RX
- RX – USART_TX

We have built the connection vice versa, but we never send data from pc to the controller, so we could spare the TX-USART_RX cable, but does not interfere with the measurement, I have kept is.

We already have all the components, can start the programming phase.

3.2 Hardware programming

The first step in programming is to configure the two sensors. This is the first function that runs, when the microcontroller starts until it has not run down, we cannot make measurements with the sensors. To configure properly, we need to thoroughly review the datasheet, try out a lot of settings, and be familiar with the options. MPU6050 has a lot of functions and I will not cover all of them, but the most necessary ones.

3.2.1 How to Initialize the device

I don't want to go into details, how to connect the sensors to the microcontroller, because we have to do it in the standard way, which means setting on the microcontroller which I2C channel we want to use and then connecting the appropriate legs of the sensor to the selected pins. If the transmission is working properly, you can start configuring the sensor.

Let us start with the initialization of the MPU6050. To initialize the sensor, we need to perform the following actions:

- **First**, we need to check if the sensor is responding by reading the "**WHO_AM_I (0x75)**" **Register**. If the sensor responds with 0x68, this means it is available and good to go.
- In **Second** step we will wake the sensor up and to do that we will write to the "**PWR_MGMT_1 (0x6B)**" Register. The register content is below:

PWR_MGMT_1

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
6B	107	DEVICE_RESET	SLEEP	CYCLE	-	TEMP_DIS	CLKSEL[2:0]		

Figure 7 PWR_MGMT_1 register

On writing (0x00) to the **PWR_MGMT_1 Register**, sensor wakes up and the Clock sets up to 8 MHz.

- **Third**, we must set the Data output Rate or Sample Rate. This can be done by writing into "**SMPLRT_DIV (0x19)**" **Register**. This register specifies the divider from the gyroscope output rate used to generate the Sample Rate for the MPU6050.

SMPLRT_DIV

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
19	25	SMPLRT_DIV[7:0]							

Figure 8 SMPLRT_DIV register

As the formula says **Sample Rate = Gyroscope Output Rate / (1 + SMPLRT_DIV)**. Where Gyroscope Output Rate is **8KHz**, To get the sample rate of **1KHz**, we need to use the **SMPLRT_DIV** as '7', but during

the development I ran into the problem that 1kHz is too fast, so I needed reduce it, so I loaded a value of **0x15**. As a result, the frequency dropped to around **400 Hz**, but so the speed was already manageable.

- **Fourth** Step is configuring the Accelerometer and Gyroscope registers and to do so, we need to modify "**GYRO_CONFIG (0x1B)**" and "**ACCEL_CONFIG (0x1C)**"Registers.

To configure the gyroscope the following table must be examined:

GYRO_CONFIG

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1B	27	XG_ST	YG_ST	ZG_ST	FS_SEL[1:0]		-		

Figure 9 GYRO_CONFIG register

For us now, bits 3 and 4, which are important, can be used to set the measuring range of the gyroscope, in the following table we can see which ranges we can choose from.

FS_SEL	Full Scale Range
0	± 250 °/s
1	± 500 °/s
2	± 1000 °/s
3	± 2000 °/s

Figure 10 gyroscope measurement ranges

Initially, I set the measurement range to **± 500 °/s**, but when I took measurements, I realized that there are still a lot of signal values between **± 500-1000 °/s** , and in this way it filter out all the data in that range. So, I loaded a value of 0x10 into the register.

We can configure the accelerometer in the same way by examining this table:

ACCEL_CONFIG

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1C	28	XA_ST	YA_ST	ZA_ST	AFS_SEL[1:0]		-		

Figure 11 ACCEL_CONFIG register

An again, the bits 3 and 4, which are important, can be used to set the measuring range of the accelerometer, in the following table we can see which ranges we can choose from.

FS_SEL	Full Scale Range
0	± 2g
1	± 4g
2	± 8g
3	± 16g

Figure 12 accelerometer measurement ranges

Initially, I set the measurement range to $\pm 4g$, but when I took measurements, I realized that there are still a lot of signal values between $\pm 4-8$, and in this way it filter out all the data in that range. So, I loaded a value of **0x10** into the register.

Then the sensors are configured, we can already read the values from the corresponding registers, but since we want to read the values from FIFO, we even must set it on the sensor.

To do this, you must first enable FIFO, using the following table:

USER_CTRL

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
6A	106	-	FIFO_EN	I2C_MST_EN	I2C_IF_DIS	-	FIFO_REST	I2C_MST_RESET	SIG_COND_RESET

Figure 13 USER_CTRL register

Since we want to enable FIFO, we need to set bit 6 to one. This means that **0x40** must be loaded.

Then we need to choose which measurement data we want to load exactly into the FIFO, the selected values will be placed in the FIFO in order so that we will be able to decode them properly.

FIFO_EN

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
23	35	TEMP_FIFO_EN	XG_FIFO_EN	YG_FIFO_EN	ZG_FIFO_EN	ACCEL_FIFO_EN	SLV2_FIFO_EN	SLV1_FIFO_EN	SLV0_FIFO_EN

Figure 14 FIFO_EN register

The value **0x78** must be loaded because we want to get both the Accelerometer data and the values of all the axes of the gyroscope.

Our last task is to set the sensor to send an interrupt after the FIFO is full and ready to read.

INT_ENABLE

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
38	56	-	-	-	FIFO_OFLOW_EN	I2C_MST_INT_EN	-	-	DATA_RDY_EN

Figure 15 INT_ENABLE register

For that we need to set **FIFO_OFLOW_EN** bit.

This completes the configuration of the sensors.

3.2.2 Read value from MPU6050

We can read 1 BYTE from each Register separately or we can just read 6 BYTES all together starting from **ACCEL_XOUT_H** Register.

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
3B	59	ACCEL_XOUT[15:8]							
3C	60	ACCEL_XOUT[7:0]							
3D	61	ACCEL_YOUT[15:8]							
3E	62	ACCEL_YOUT[7:0]							
3F	63	ACCEL_ZOUT[15:8]							
40	64	ACCEL_ZOUT[7:0]							

Figure 16 registers where the accelerometer values stored

AFS_SEL	Full Scale Range	LSB Sensitivity
0	±2g	16384 LSB/g
1	±4g	8192 LSB/g
2	±8g	4096 LSB/g
3	±16g	2048 LSB/g

Figure 17 AFS_SEL register

As shown above, The **ACCEL_XOUT_H (0x3B) Register** stores the higher Byte for the acceleration data along X-Axis and Lower Byte is stored in ACCEL_XOUT_L Register. So, we need to combine these 2 BYTES into a 16-bit integer value. Below is the process to do that: -

$$ACCEL_X = (ACCEL_XOUT_H \ll 8 | ACCEL_XOUT_L)$$

we are shifting the higher 8 bits to the left and then 'OR' it with the lower 8 bits.

For Example, if ACCEL_XOUT_H = 11101110 and ACCEL_XOUT_L = 10101010, we will get the resultant 16 bit value as 1110111010101010

Similarly, we can do the same for the **ACCEL_YOUT** and **ACCEL_ZOUT** registers. These values will still be the RAW values and we still need to convert them into proper 'g' format.

You can see in the picture above that for the Full-Scale range of **± 8g**, the sensitivity is **4096 LSB/g**. So, to get the **g** value, we need to divide the RAW from **4096**.

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
43	67	GYRO_XOUT[15:8]							
44	68	GYRO_XOUT[7:0]							
45	69	GYRO_YOUT[15:8]							
46	70	GYRO_YOUT[7:0]							
47	71	GYRO_ZOUT[15:8]							
48	72	GYRO_ZOUT[7:0]							

Figure 18 registers where gyroscope values stored

FS_SEL	Full Scale Range	LSB Sensitivity
0	$\pm 250 \text{ }^\circ/\text{s}$	131 LSB/ $^\circ/\text{s}$
1	$\pm 500 \text{ }^\circ/\text{s}$	65.5 LSB/ $^\circ/\text{s}$
2	$\pm 1000 \text{ }^\circ/\text{s}$	32.8 LSB/ $^\circ/\text{s}$
3	$\pm 2000 \text{ }^\circ/\text{s}$	16.4 LSB/ $^\circ/\text{s}$

Figure 19 FS_SEL register

Reading the gyroscope data is like reading the acceleration. We will start reading 6 BYTES of data from the **GYRO_XOUT_H** Register, Combine the 2 Bytes to get 16 bit integer RAW values. As we have selected the Full-Scale range of $\pm 1000 \text{ }^\circ/\text{s}$, for which the sensitivity is **32.8 LSB / $^\circ/\text{s}$** , we have to divide the RAW values by **32.8** to get the values in **dps ($^\circ/\text{s}$)**.

3.2.3. What the microcontroller needs to use?

The microcontroller performs the control, it is important to have the right pin assignment and the right tools. The pin assignment of the microcontroller looks like this:

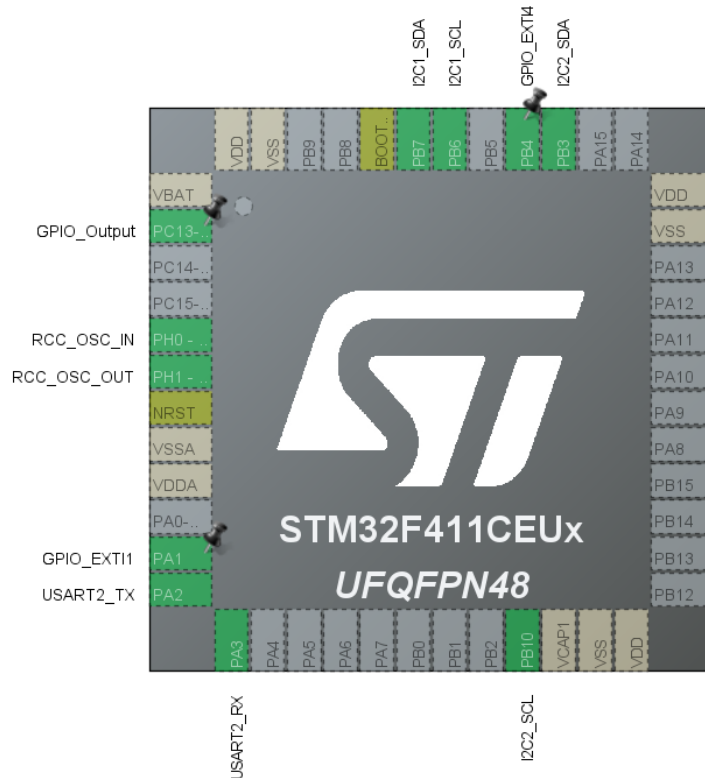


figure 20 pinout of the microcontroller

We communicate with the sensors via I2C, which requires 2 I2C channels, the first channel using pins **PB7** and **PB8** and the second using pins **PB3** and **PB10**.

We use the I2C channels in fast mode, that means the speed is **400000 Hz**.

We use the I2C channels with **DMA**, each of them in **FIFO** mode. It is very important to use each **DMA** with different priority, because in case of conflict it should decide which one goes first. If we use the same priority level it can be confusing for the MUC.

We use two **EXTI** interrupts, we need it, because MPU-6050 can send interrupt, when its FIFO is full, and we would like to handle that interrupt. The priority is important in that case too, we should use them with different priority level and the explanation is the same as in the case of DMA.

We need UART interface to communicate with the world. It uses pin **PA2** and **PA3**. The baud rate is **115200** bits/s, I tried to use it with extra high speed, with **1000000** and **2000000** bits per second, but it caused a lot of bit errors, so I have decided to use it only on **115200** bit/s.

As clock speed I use the maximum **100 MHz**.

We have configured everything except **CUBE AI**. We need to configure it, when the model is ready and we just need to upload it. In the first period of the development we do not need CUBE AI.

3.3 Read the data when an event happened

The most difficult challenge during the programming of the hardware was, how we can determine when a sample starts, when I should start the recording. To solve this, I figured out that after a trigger event occurred, we should start to measure the sample and save it to memory. I have developed an algorithm to solve this problem.

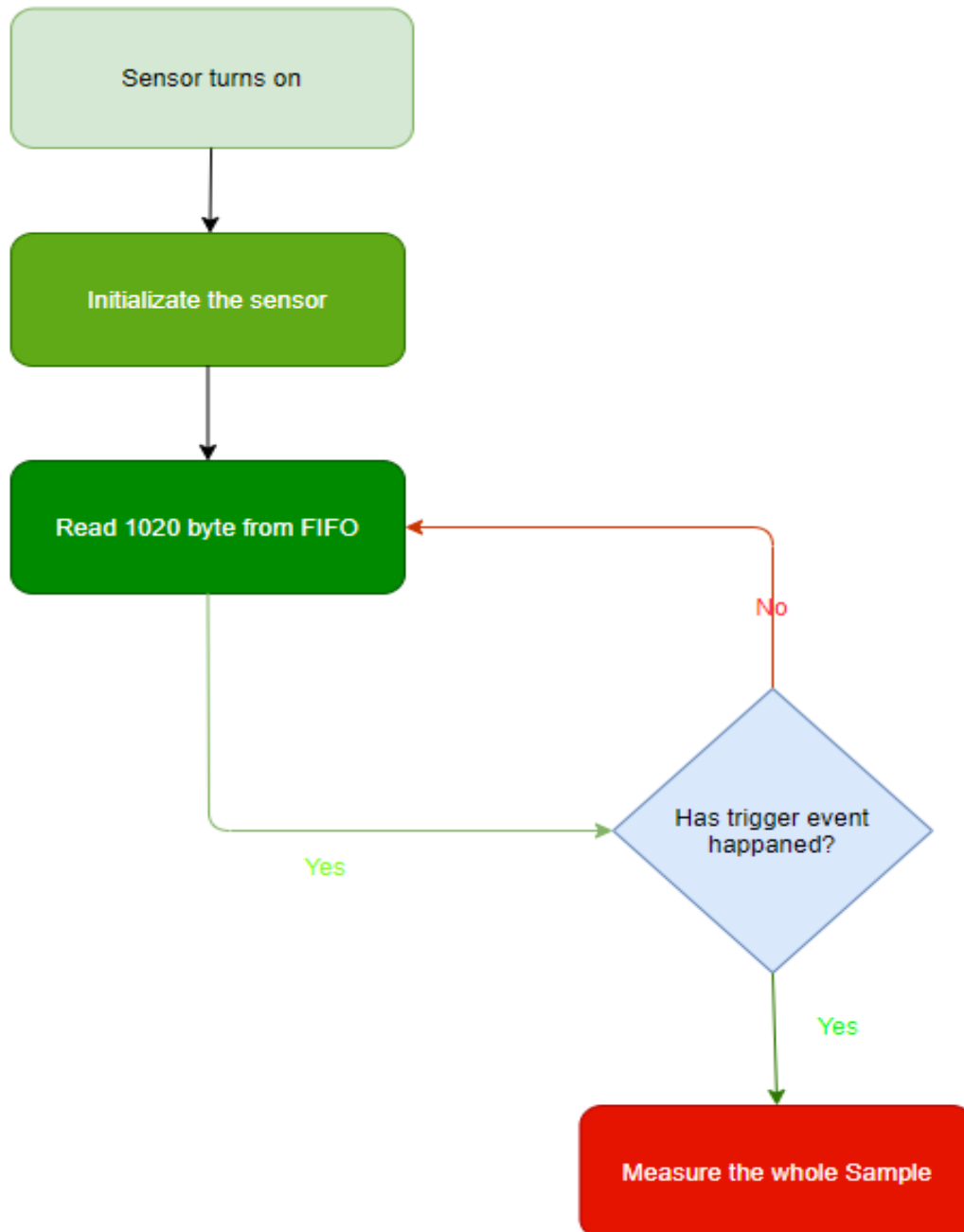


figure 21 Flow chart of the trigger event

Once the sensors are turned on, we initialize them, then we start measuring. Since we are measuring with FIFO, we need to make sure that the FIFO is full. If it is full, it discards the oldest value and accepts the new value instead. This is bad for us because the bytes come in line. AXH (Accelerometer, X axis, High Byte), AXL, AYH, AYL, AZH, AZL, GXH, GXL, GYH, GYL, GZH, GZL, however, if it is overwhelmed, the correct order may slip, and the data cannot be decoded afterwards. If we read more data, than the contents of the FIFO, it reads the last bit again and again, so the order also shifts. The correct read speed, which I could only achieve with DMA, is important. DMA is also important, because it saves us resources, because the CPU does not have to bother with reading, the DMA takes over this task from it.

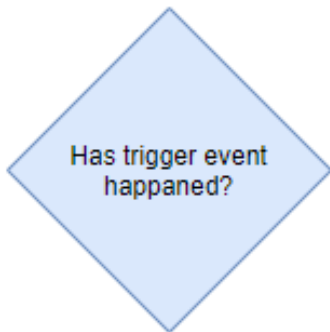


figure 22 The trigger event

So, the greatest difficulty was, how we could detect the trigger event. I solved this by taking the High Byte of the 3 axes of the accelerometer, converting them to int, and then taking their absolute values, summing them, and checking that, if the value thus obtained is greater than a certain value. This certain value is the sensitivity, increasing it makes the sensor more insensitive to movements, reducing it can make it hypersensitive.

$\text{abs (AXH)} + \text{abs (AYH)} + \text{abs (AZH)} < \text{value}$

Once I was able to save the samples, I had to send them to the computer for process them and create the artificial neural network.

4. Data processing, and preparation of the samples for learning

The most important resource of Deep learning is data, the most difficult steps in implementing Deep learning is to make and process the data. The network can interpret data only in suitable form, otherwise learning is no feasible. Before we can take samples, we need to define the application, what we want to realize with the network, and then we need to record the samples in the best way possible.

4.1 Selection of samples

As an application I have chosen an arm movement recognizer, the question was, what movements we would like to detect exactly. The movements are not so important in our case, once we have built a model, we will be able to build a different one with other movements. Thus, only the samples will affect what our application recognizes.

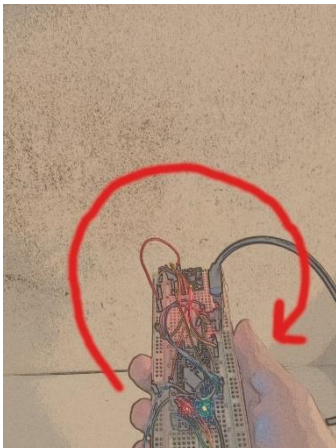


figure 23 clockwise-circle

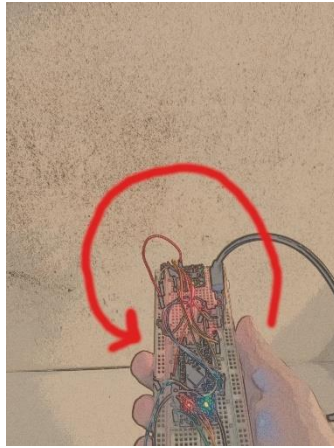


figure 24 counter-clockwise-circle

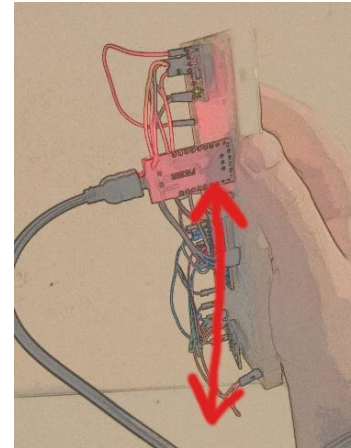


figure 25 train

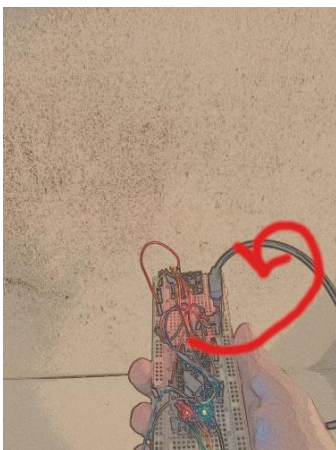


figure 26 twist (start position)



figure 27 twist (end position)

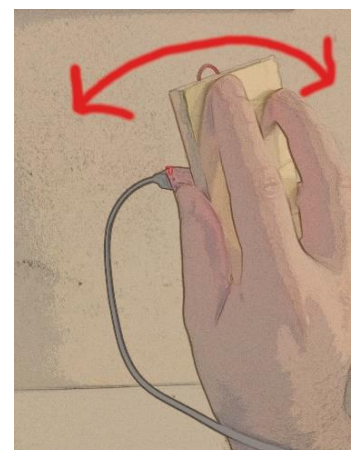


figure 28 wave

I came up with 5 different labels, which are:

1. Clockwise-circle: This movement is a description of a clockwise hoop in the air.
2. Counter-clockwise-circle: It is the same move, just in the opposite direction.

3. Twist: It is a twisting gesture, initially the sensor faces up and the end of the gesture is already down.
4. Train: It is an up-and-down movement, the name was given from the honking movement of the steam locomotive driver.
5. Wave: It is a simple waving gesture.

4.2 Receive samples on the PC

As I mentioned earlier, the samples are sent over UART and then we must receive that data somehow on the PC side. I saved the samples in a CSV (comma-separated values) file, during processing we will read the samples from here and after we will teach the ANN with these data.

To take the samples, I wrote a program that can record and draw our samples. Drawing samples is important, because if we work with bad samples, it can spoil our efficiency of the ANN and after that, it is very difficult to trace back where the problem was, because we need to find the problem in millions of lines.

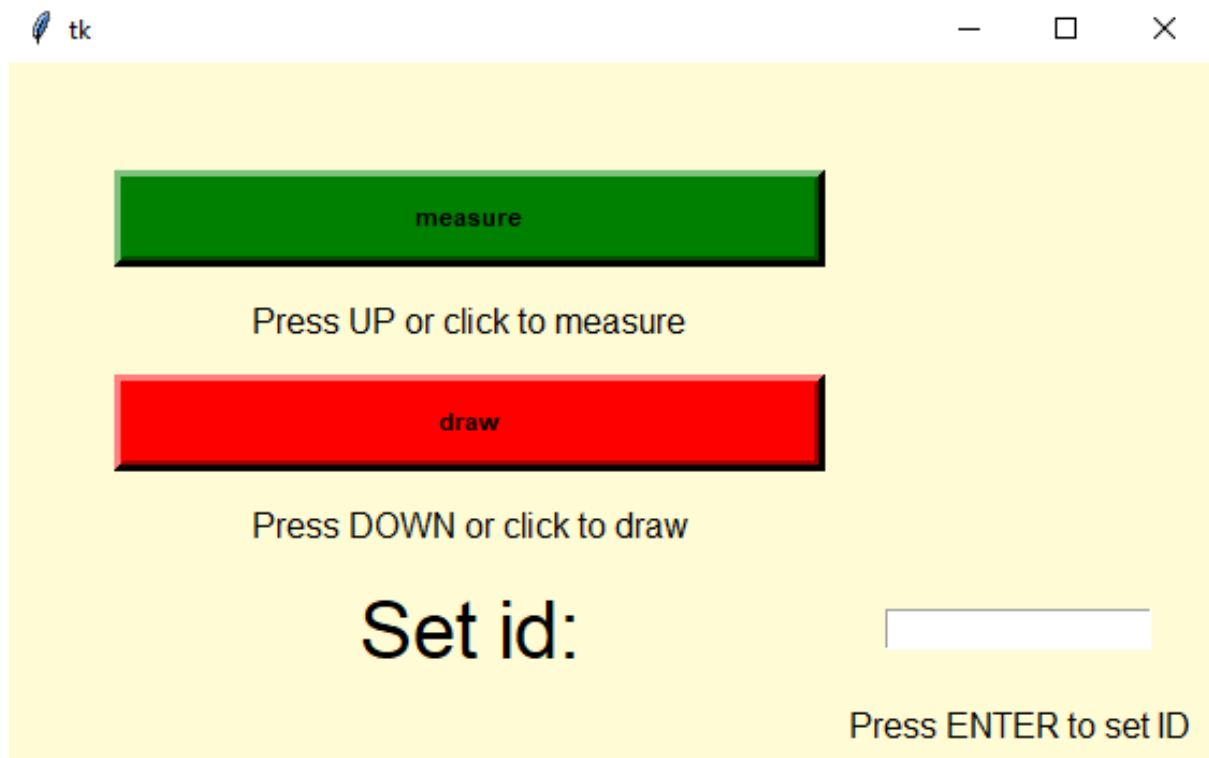


figure 29 Program to measure samples

The program, what I have written can record and draw samples. In the lower right corner, you can enter the sample ID. This means that when reading, assign a particular pattern to this ID, which means that if a pattern with such an ID already existed, it will be added to it again, so it is advisable to avoid this. And when drawing, use the ID variable to know which pattern to draw.

To help with the operation, I also added keyboard shortcuts to the program, the down arrow can be used to draw the given sample, the up arrow can be used to start the measurement and if we have entered a value for the ID, it can be confirmed with ENTER. Basically, I can record 1 sample when I press the record button, but when recording the samples I set it to call itself again after it has finished the measurement, so since the sensor hardware was also set to pick up the next one immediately after sending the recorded sample, so the samples could be taken up much faster than if everything had to be started one by one. Sampling is always a monotonous and lengthy job, with these features it was slightly easier to perform sampling. I wrote the GUI for my program with a python package called Tkinter, which is part of the python by default.

The data saved in the CSV file looks like this:

	A	B	C	D	E	F
1	1,1,clockwise-circle,0,9,141,255,235,22,68,5,169,14,231,5,50					
2	1,1,clockwise-circle,1,9,110,255,167,21,198,5,174,14,159,5,19					
3	1,1,clockwise-circle,2,9,145,255,140,21,106,5,194,14,113,4,247					
4	1,1,clockwise-circle,3,9,141,255,61,20,202,5,224,14,82,4,231					
5	1,1,clockwise-circle,4,9,100,255,13,20,18,6,21,14,58,4,222					
6	1,1,clockwise-circle,5,9,99,254,209,19,116,6,86,14,49,4,230					
7	1,1,clockwise-circle,6,9,66,254,199,19,30,6,167,14,35,4,243					
8	1,1,clockwise-circle,7,9,22,254,170,18,181,6,238,14,3,5,5					
9	1,1,clockwise-circle,8,8,224,254,153,18,131,7,59,13,215,5,38					
10	1,1,clockwise-circle,9,8,171,254,166,18,108,7,128,13,167,5,67					
11	1,1,clockwise-circle,10,8,87,254,200,18,85,7,170,13,121,5,88					
12	1,1,clockwise-circle,11,7,245,255,0,18,50,7,204,13,45,5,107					
13	1,1,clockwise-circle,12,7,93,255,76,18,43,7,219,12,219,5,105					
14	1,1,clockwise-circle,13,6,131,255,163,18,90,7,234,12,121,5,87					
15	1,1,clockwise-circle,14,6,49,255,217,18,60,7,237,11,234,5,64					
16	1,1,clockwise-circle,15,5,195,255,207,17,204,7,242,11,64,5,38					
17	1,1,clockwise-circle,16,5,210,0,30,17,186,7,250,10,162,5,9					
18	1,1,clockwise-circle,17,6,5,0,29,17,126,7,247,10,5,4,233					
19	1,1,clockwise-circle,18,6,92,255,241,17,124,7,235,9,125,4,222					
20	1,1,clockwise-circle,19,6,183,255,233,17,154,7,215,9,16,4,205					
21	1,1,clockwise-circle,20,6,209,255,171,17,179,7,196,8,187,4,203					

figure 30 Data structure in CSV file

A row contains one measurement data, one sample contains 2550 rows, which since 2 sensors it is de facto 1275.

Data in a row in order:

```
fieldnames = ['id', 'sensorID', 'label', 'count', 'Ax1', 'Ax2', 'Ay1', 'Ay2', 'Az1', 'Az2', 'Gx1', 'Gx2', 'Gy1', 'Gy2', 'Gz1', 'Gz2']
```

figure 31 fieldnames

- **id:** This is the identifier of the sample, it separates the samples from each other.
- **sensorID:** We take the data from 2 sensors, first take 1275 measurements from sensor1, and then take the same amount from sensor2. This field is used to distinguish this (sensors measure data simultaneously, but we save it into two different arrays).
- **label:** That is what the ANN will predict.
- **count:** It counts the measurements in samples.
- **Ax1, Ax2, Ay1, Ay2, Az1, Az2:** Accelerometer data stores data in 2 bytes per axis. I saved raw data to the CSV file, because I need to process data on the microcontroller as well.
- **Gx1, Gx2, Gy1, Gy2, Gz1, Gz2:** The gyroscope works in a similar way to the accelerometer.

After recording the data, we can then draw it, using the matplotlib package.

The patterns look like this:

- clockwise-circle:

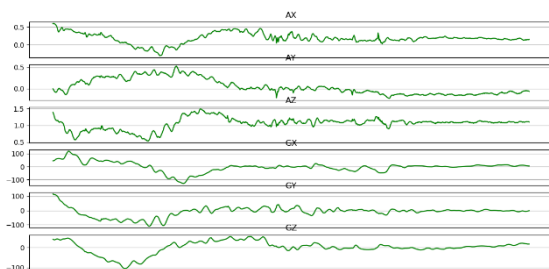


figure 32 clockwise-circle (sensor1)

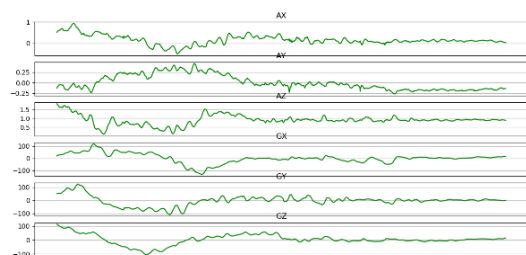


figure 33 clockwise-circle (sensor2)

- counter-clockwise-circle:

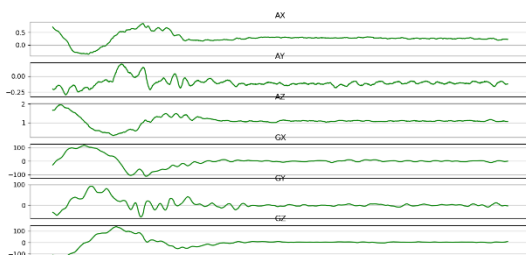


figure 34 counter-clockwise-circle (sensor1)

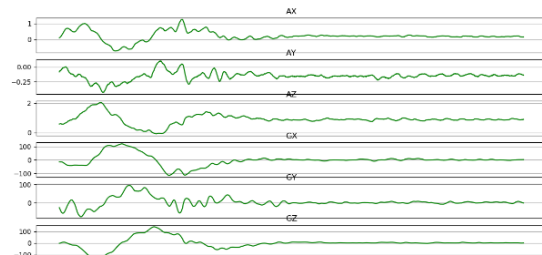


figure 35 counter-clockwise-circle (sensor2)

- twist

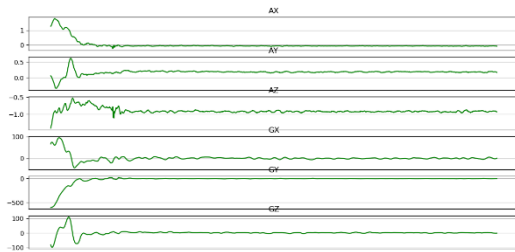


figure 36 twist (sensor1)

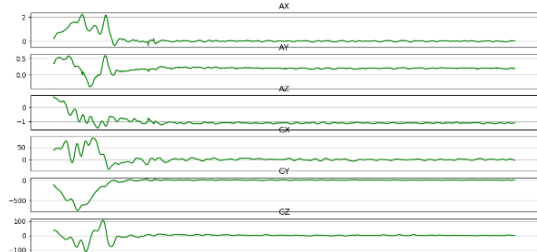


figure 37 twist (sensor2)

- train

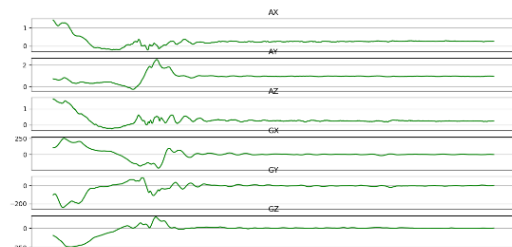


figure 38 train (sensor1)

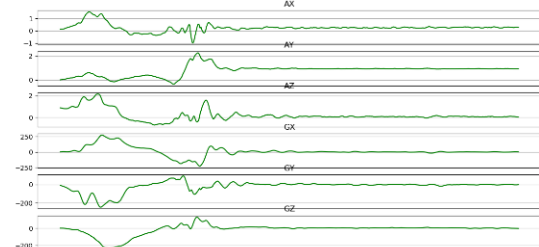


figure 39 train (sensor2)

- wave

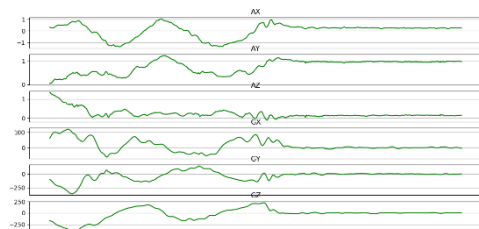


figure 40 wave (sensor1)

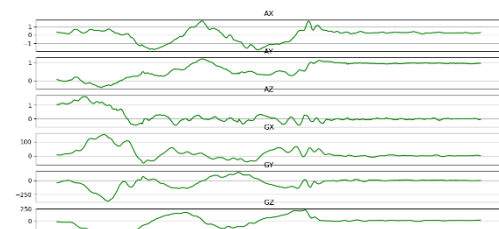


figure 41 wave (sensor2)

Once we have the samples, we can start the processing phase, so the data get suitable to ANN, and then we can start building up the ANN.

5 Neural network and teaching

The samples are done, all that is left is to transform the data properly and create the neural network, then teach it. There are several frameworks available for deep learning, I used Tensorflow and Kerast. Tensorflow is a framework developed by Google, the essence of which is to process data in the form of a tensor, as the name implies. A tensor is a mathematical object that is a generalization of the concept of scalar and vector. Most of us are not unfamiliar with the concept of tensor.

The processing process is summarized as follows:

1. Label the data: During the data processing, we convert labels into numbers, so we will have an int value between 0-4.
2. Data normalization: The network can only work with values between -1 and 1, so we need to normalize our data. We can achieve this by dividing the values with their maximums.
3. We break separate data into samples: The samples need to be separated, so that we can treat one large data set as several small samples.
4. We divide the data into two sets of 'train' and 'test': During learning, the network learns on the train data and then tests on the test data, and then evaluate, how successful the learning was.
5. The data must be brought into the appropriate tensor shape: In the case of tensorflow, the network has its own expectation, that means we need to provide data in suitable dimension tensor.

We can create the neural network and begin the process of teaching. The model uses 5 layers:

1. 2D convolution (window size: 2×2 , number of output channels: 16, activation: ReLu)
2. 2D convolution (window size: 2×2 , number of output channels: 32, activation: ReLu)
3. Flatten layer
4. Dense (number of neurons: 64, activation: ReLu)
5. Dense (number of neurons: 9, activation: Softmax)

When selecting layers, this layer selection proved to be the most appropriate in terms of effectiveness. That's the number of layers needed, more are unnecessary.

I drew the layers with an application called netron, the web application of that is available at the following link: <https://lutzroeder.github.io/netron/>

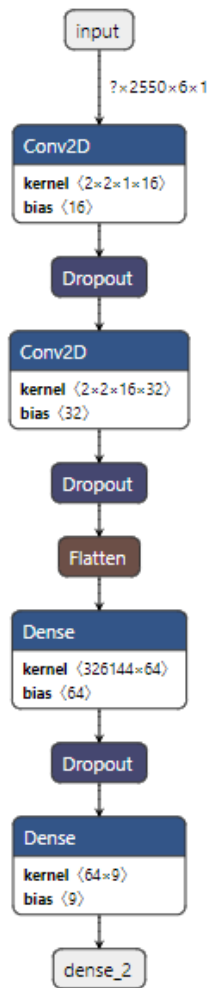


figure 42 model0

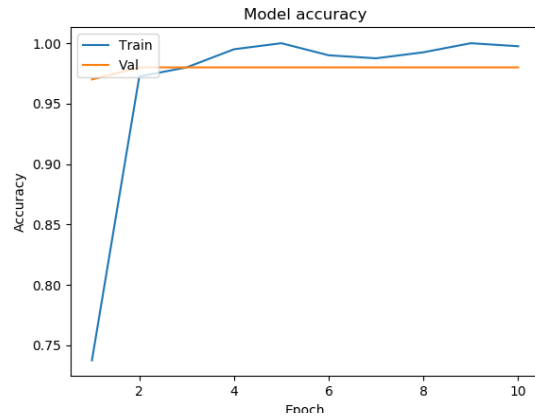


figure 43 model0 accuracy

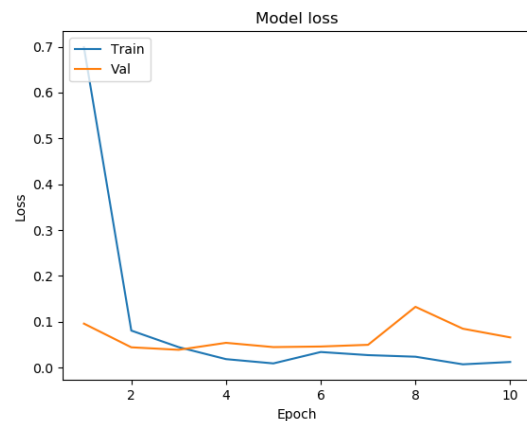


figure 44 model0 loss

As a rule of thumb, it is generally worth choosing ReLU (Rectified Linear Unit) for the activation of the intermediate layers and Softmax for the output layer.

The net is ready to start teaching, learning uses 10 epochs.

The blue line indicates the effectiveness on the data used for learning, and the orange line indicates the effectiveness on the data used for testing. For us, the effectiveness of the test is important. As can be seen in the figure, the efficiency quickly settled to a certain value, in our case it fluctuates around 98%. This is because the data set aside for learning has already been learned and there is no room for further development. You may also learn train data too well and may no longer be able to create normal prediction on test data. This explains why the efficiency of the model cannot be increased indefinitely. To achieve even better efficiency, we would need many more samples, but 98% is more than enough.

The model is ready, we have nothing else to do, but save and apply it in real situations as well. I have written a program that sign the readiness of a measurement with a sound of beep, after the beep sound, we can perform a movement, the trigger event start the recording and the data will have saved in an array. After we have the full sample, the program processes the data it formats it to suitable form, and we load the data in the model, and it makes a prediction for us.

The model creates a probability value for each output in some way like this:

	0	1	2	3	4
0	0.999999	9.99928e-07	1.1635e-09	9.38776e-13	2.79277e-12

Figure 45 array of predictions

This can be interpreted as follows: ANN tells each output a percentage of what it thinks is the chance that this is the correct output. From the predictions, we select the maximum and say that according to the network, this is the prediction. From the figure it can be read that it is 99.99% sure that it was a clockwise-circle. Although it does not matter to the outside observer that the model is 51% or 99% certain in the output, it is interesting to note that the mesh run on the microcontroller will not only lose its efficiency, but if it makes a good prediction, it will be much less confident. The accuracy graph drawn during training does not always correspond to reality, so I took 20-20 measurements on each label and looked at the actual accuracy.

The results were the following:

- clockwise-circle: **20** / 20
- counter-clockwise-circle: **20** / 20
- twist: **20** / 20
- train: **20** / 20
- wave: **20** / 20

This means that our efficiency is appropriate, we can even say it is perfect. We might just get a wrong result, if we perform 1000, 10,000 or 1,000,000 measurements. Nevertheless, this means that ANN works well, it is not worth to improve its efficiency further.

It makes sense to work with that neural network, it can be applied to an MUC. In that case if the network would not work properly, the performance degradation would not be representative.

6 Problems with implementation on a microcontroller

The microcontrollers have limited amount of memory, at first, I was working with an STM32F401CEU6 with a flash memory of 256 KB and a RAM of 64 KB. The size of the RAM was almost completely utilized already during the sampling process, and I didn't even have an idea of the size of the ANN at that time, so I switched to the one more powerful STM32F411CEU6, which has twice the size of Flash memory and RAM, 512 KB and 128 KB.

Some stronger microcontroller memory sizes:

Product lines	Flash(Kbytes)	Ram(Kbytes)
STM32F469	512 – 2056	384
STM32F429	512 – 2056	256
STM32F427	1024 – 2056	256
STM42F446	256 – 512	128
STM32F407	512 – 1024	192
STM32F405	512 – 1024	192
STM32F401	128 – 512	up to 96
STM32F410	64 – 128	32
STM32F411	256 – 512	128
STM32F412	512 – 1024	256
STM32F413	1024 - 1536	320

Figure 46 memory size of microcontrollers

I obtained the data from the STMicroelectronics website at the link: https://www.st.com/content/st_com/en/products/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus/stm32-high-performance-mcus/stm32f4-series.html I looked at the M4 series, because this series is already capable of complex mathematical calculations and its energy consumption is also favorable.

If we look at the table (figure 46), we can see the maximum Flash size is 2056 Kbytes (2 MB), and the maximum RAM is 384 Kbytes. We could use external memory if it needed, but they have not limitless sizes.

Our model has a huge memory usage, 238 MB (figure 47), it is too big even for the toughest microcontrollers, so we need to shrink it as much as we can.

I tried to make it smaller with Tensorflow Lite, it decreases the model sizes. A very useful tool for mobile applications, because today data scientists use larger, and larger models for imitate the human brain better, but we can't use hundreds of Megabytes large models on mobile phones, because average application size is about 15 MB. I could shrink the model for 79,6 MB with Tensorflow Lite, but it too large for a microcontroller, so I had to find another solution.



 model.h5	2020.10.04. 19:10	H5 fájl	244 676 KB
 model.tflite	2020.10.08. 20:33	TFLITE fájl	81 549 KB

Figure 47 size of the original model

STM has a framework for run ANN on STM microcontrollers, and it has size decreasing effect too, so I tried to upload my model, and checked the result:



figure 24 size of the model0 in CUBE AI

It uses a huge chunk of flash and RAM too, I had to find a way to reduce the size to 100 KB.

I wanted to reduce the size of the model, so I was forced to modify the model by reduce the number of neurons, vary the window size, and so on. The experimentation was mainly empirical, of course it took a good background knowledge, so that, and it is worth trying, there were many models, which immediately reduced the efficiency to a very low level, but in the following figures I will only illustrate those that were within the tolerance in loss of efficiency.

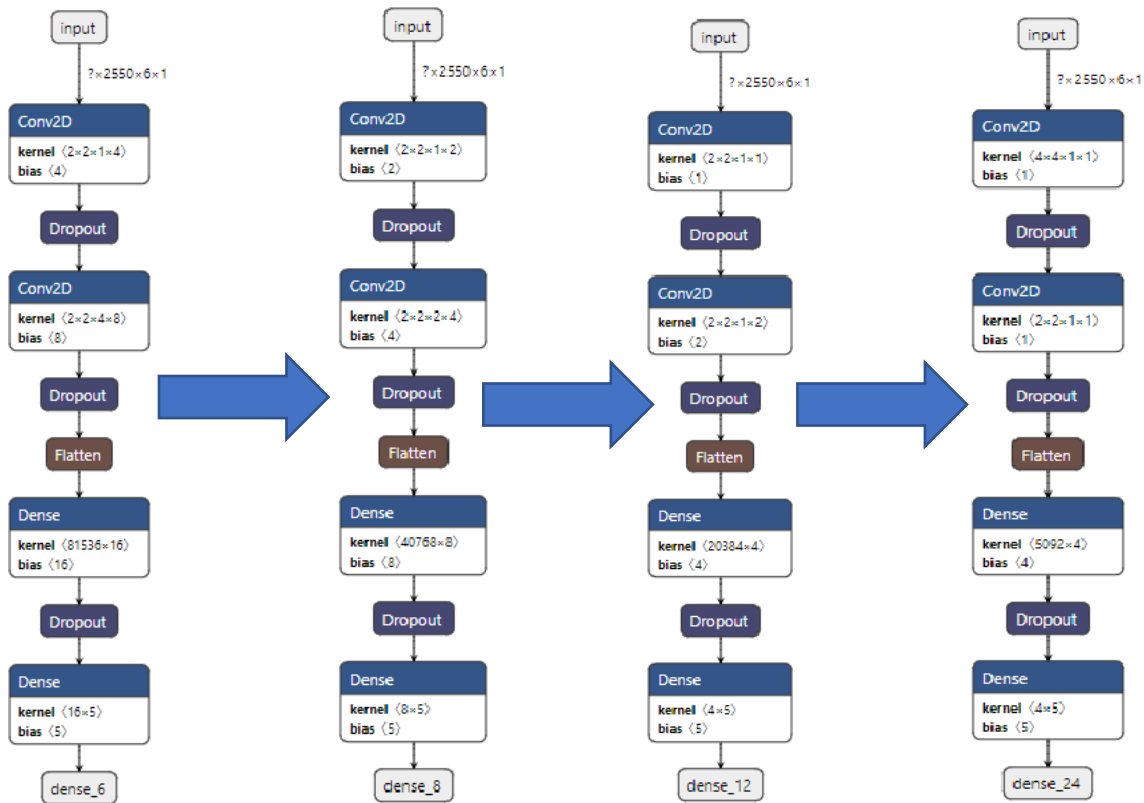


Figure 48 model_4

Figure 49 model_5

Figure 50 model_6

Figure 51 model_7

14,9 MB

3,76 MB

431 KB

272 KB

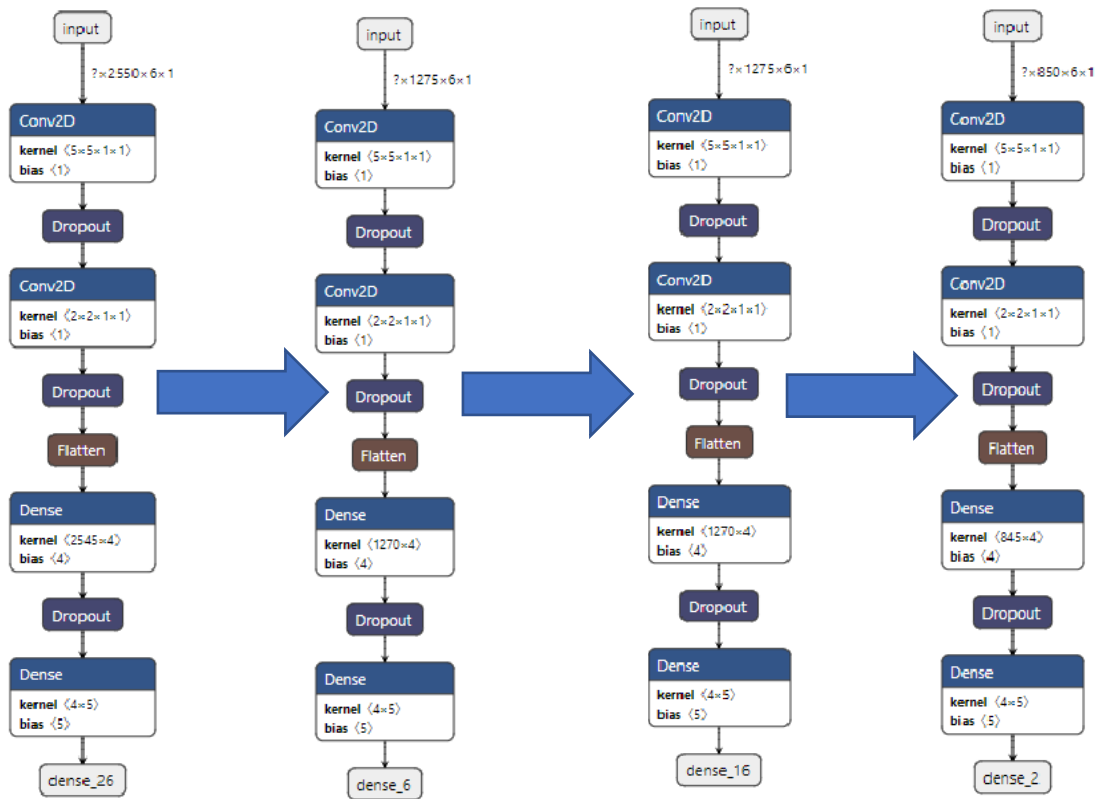


Figure 52 model_8

Figure 53 model_9

Figure 54 model_10

Figure 55 model_11

154 KB

94 KB

94 KB

74 KB

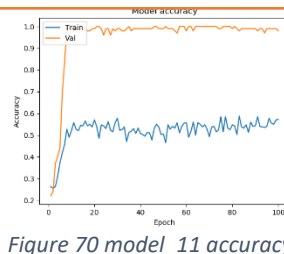
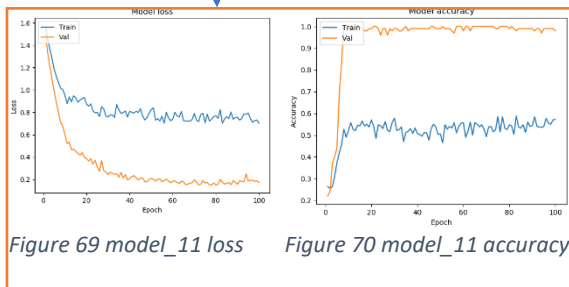
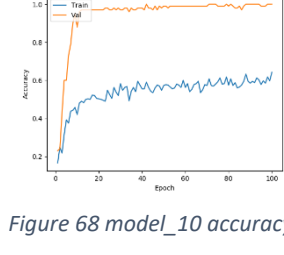
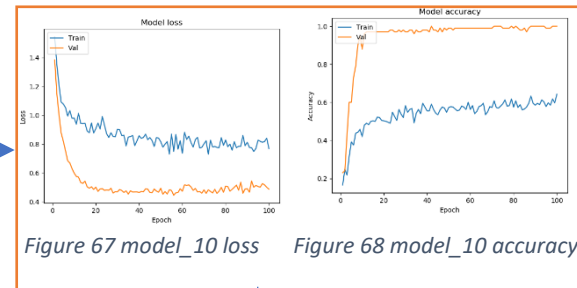
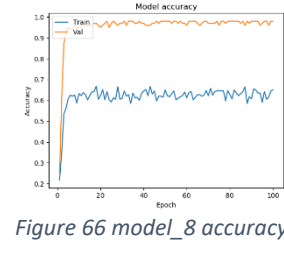
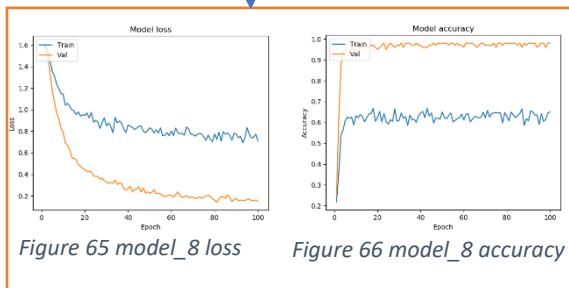
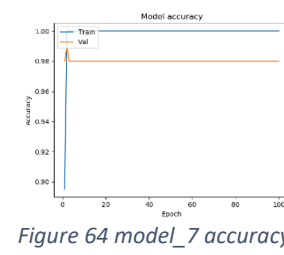
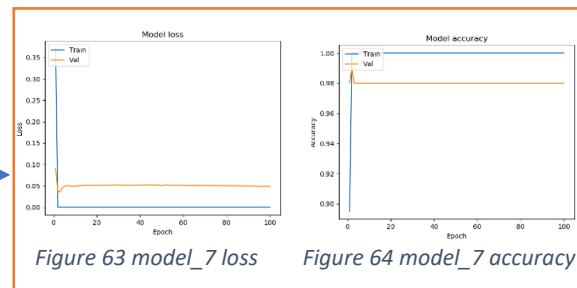
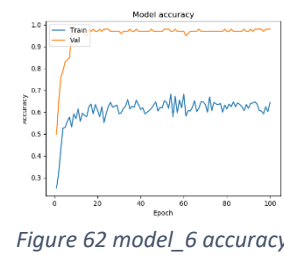
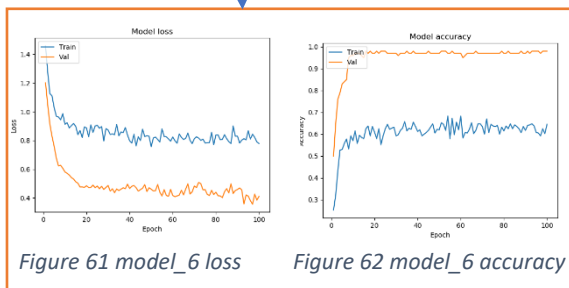
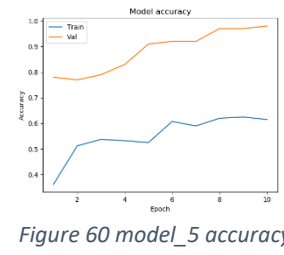
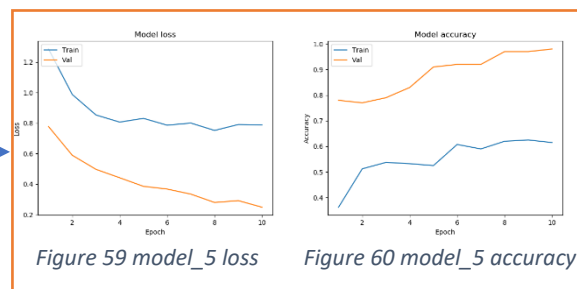
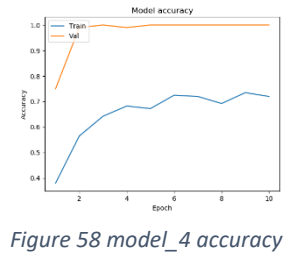
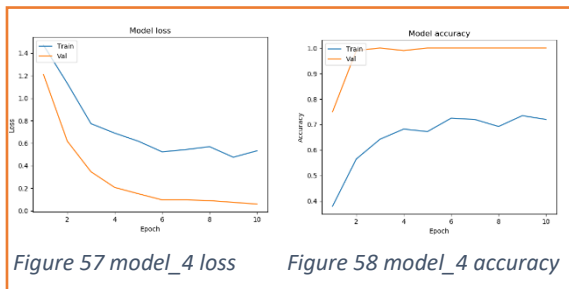
The size shrink in one figure:

Név	Módosítás dátuma	Típus	Méret
model_4.h5	2020.10.12. 20:18	H5 fájl	15 325 KB
model_5.h5	2020.10.12. 20:20	H5 fájl	3 857 KB
model_6.h5	2020.10.12. 20:31	H5 fájl	342 KB
model_7.h5	2020.10.13. 22:20	H5 fájl	273 KB
model_8.h5	2020.10.13. 22:25	H5 fájl	154 KB
model_9.h5	2020.10.15. 20:07	H5 fájl	94 KB
model_10.h5	2020.10.15. 20:17	H5 fájl	94 KB
model_11.h5	2020.10.17. 15:55	H5 fájl	74 KB



Figure 56 shrinkage the size of the model

The size of the model by this time is optimal, but its efficiency is an equally important issue, I have attached the figures below about the efficiencies of the models during teaching.



These graphs are not entirely reliable, they are merely just the results of teaching process. If we take a closer look, we can see that the validation accuracy remained around 98%, but training accuracy was significantly reduced to around 60%. It is our only statistic during the teaching process, but the real efficiency always loss that we test with new samples.

Too nice is also suspicious, in the case of model_7 the accuracy is 1, but after I tested the model with real data, I experienced that the inaccuracy was high.

At model_8 we had already reached the limit we wanted, it occupies a size of 80 KB, I saved the data at 2x15 KB, so it was border-line case, so the size of the model already looked fine.



Figure 71 model_8 memory usage

The next problem came when I would have started using ANN on the microcontroller. All that numpy, pandas and other python libraries can easily solve, I had to solve on the microcontroller at a low level. This was not a problem either, the single-line code can be implemented in many loops, but we need temporarily more arrays for processing, in which we can save the format we need. For example, the network expects normalized floats, and we save integer values to the arrays. The conversion is simple, it stores an axis value in 2 bytes, and the task can be easily accomplished with bit shifts and type casting. The problem is that the 2 bytes are doubled in this case since the size of the float is 4 bytes. I tried to make the samples with the highest possible sampling frequency, although I knew that similar solutions would be sampled at a much lower frequency. The reason for this was that it is easier to reduce backwards from oversampling than to increase posteriorly the sampling. In practice, this would mean re-sampling, which is a very lengthy task and it is not known in advance how effective it will be. Initially, I tried to sample at 1 KHz, this was too fast for the FIFO, the bytes started to shift, and other problems arose. To remedy this, I reduced the frequency to around 400 Hz, and I have taken samples with that frequency. Since leaving multiple neurons was no longer possible without the model completely losing its function, I was forced to change a frequency.

At first, I wanted to halve the frequency, this can be seen at model_9 and model_10, but unfortunately this was not feasible, because the value thus obtained could not be divided by 1020. The divisibility with 1020 comes from the fact that FIFO has a memory of 1024 bytes, so I wanted to use as many slices as possible from its capacity. Since I am reading 12 bytes (AXH, AXL, AYH, AYL, AZH, AZL, GXH, GXL, GYH, GYL, GZH, GZL), hence it must be divisible by 12, otherwise the data will slip after I measured a whole sample, because I read 4 byte from the next sample and that 4 byte will have been missed from the next sample. In 1024, the largest number is 1020, which is divisible by 12, so I have chosen it.

I could have decided to halve the data, but if the FIFO reads only 510 bytes, I will not use half of its capacity. The FIFO plays a major role in ensuring that the data is read using the DMA without a particular load on the CPU, so that the CPU can handle the other computational tasks. Although I tried to keep as many resources as possible to use ANN, I stopped reading while using ANN so it was not disturbed by the reads. The importance of the DMA was in the detection of the trigger event, as it had to be examined during a read.

Since I wanted to devote the resource to data processing and ANN, I decided to trisect the frequency instead of halving it, because it was divisible by 1020. The scan frequency was decreased, it has dropped down to 140 Hz.

Since I reduced the size of the samples, I also reduced the size of the model, due to size decreasing of the input layer. In the case of model_9 and model_10 I halved the size of the samples, so the size was reduced from 154 KB to 94KB. After trisecting the frequency, it decreased to 74 KB.

The memory space used by CUBE AI is 13.44 KB flash memory and 26.56 KB RAM. Since the original model has the name model0, I will call this model model1 from now, because this was the first model I have successfully used on the microcontroller.

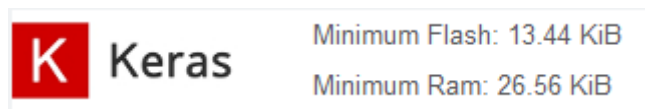


Figure 72 model1 memory usage

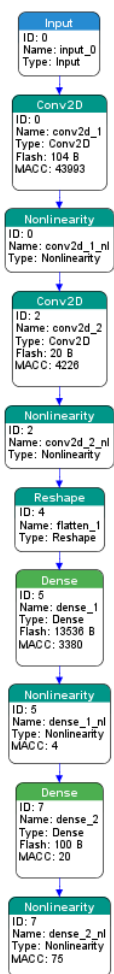


Figure 73 graph of model1

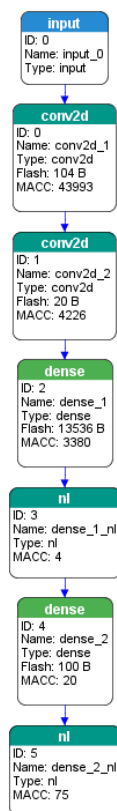


Figure 74 C graph of model1

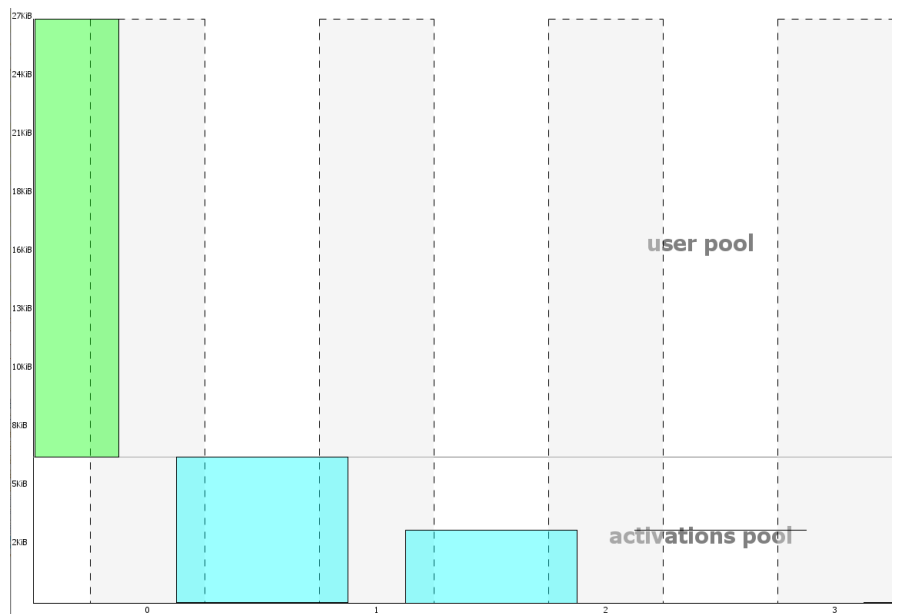


Figure 75 memory usage diagram of model1

On the uploaded model, the simplifications in Figure 73/74 are performed by CUBE AI, and memory usage is indicated in Figure 75.

The uploaded network is just binary data, it looks like this:

```
ai_handle ai_network_data_weights_get(void)
{
    AI_ALIGNED(4)
    static const ai_u8 s_network_weights[ 13760 ] = {
        0x4a, 0x86, 0xf2, 0x3d, 0x7b, 0x4a, 0x8d, 0x3e, 0x93, 0x74,
        0x24, 0x3f, 0xce, 0x65, 0xcd, 0x3e, 0xf4, 0x42, 0xb1, 0x3e,
        0x95, 0x0c, 0x0f, 0x3f, 0x80, 0x9e, 0x33, 0x3e, 0xc8, 0xee,
        0xcf, 0x3e, 0x23, 0xd6, 0xcb, 0x3d, 0xb7, 0x65, 0x94, 0x3e,
        0x6c, 0x7f, 0x03, 0x3f, 0x13, 0xca, 0x01, 0x3f, 0x5c, 0xae,
        0x24, 0x3f, 0xf0, 0x0a, 0xb9, 0x3e, 0xb2, 0xad, 0x72, 0x3e,
        0x0e, 0x87, 0xf1, 0x3e, 0x9f, 0xb2, 0x34, 0x3f, 0x0f, 0x8b,
        0x4d, 0x3d, 0x5c, 0xf1, 0x3f, 0x3e, 0xc4, 0x11, 0x2d, 0x3e,
        0x89, 0x8d, 0x18, 0x3f, 0xeb, 0x14, 0xa6, 0x3e, 0x7d, 0x6d,
        0x25, 0x3d, 0xbe, 0xf0, 0xac, 0x3e, 0x87, 0x8e, 0xec, 0x3e,
        0x5e, 0x6a, 0x71, 0x3c, 0x43, 0x93, 0xbf, 0x3c, 0xca, 0x26,
        0x09, 0x3f, 0xce, 0xa0, 0x82, 0x3f, 0xdc, 0x9b, 0x7e, 0x3f,
        0xee, 0x0e, 0x01, 0x3d, 0xbb, 0x7f, 0x16, 0x3d, 0xf7, 0xf1,
        0x8c, 0x3c, 0x91, 0x75, 0x61, 0x3d, 0xb4, 0xcf, 0x81, 0xb7,
        0x13, 0xb2, 0xea, 0xbc, 0x7b, 0x14, 0xaa, 0xbd, 0xa5, 0x76,
        0x87, 0xbd, 0xd1, 0x96, 0x6a, 0xbe, 0xe8, 0x47, 0xf1, 0xbd,
        0x8c, 0x74, 0xf5, 0x3b, 0xfe, 0xfb, 0x98, 0x3d, 0x3d, 0xa9,
        0x13, 0x3e, 0x02, 0x4a, 0x3d, 0x3e, 0x5d, 0x4a, 0xf3, 0x3d,
        0x49, 0x9f, 0x31, 0x3e, 0x09, 0xd4, 0x08, 0x3e, 0x88, 0x9c,
        0x4e, 0x3d, 0x1b, 0x1f, 0xa3, 0x3d, 0x31, 0x1b, 0xc5, 0xbc,
        0x04, 0x76, 0xc5, 0xbd, 0x0f, 0xde, 0x99, 0xbd, 0xe6, 0xe8,
        0x1e, 0xbb, 0xc6, 0x08, 0x9e, 0xbd, 0x2d, 0x6b, 0x07, 0xbd,
        0x62, 0xbe, 0xc8, 0xbd, 0xdd, 0xa9, 0x9a, 0xbc, 0x80, 0x6a,
```

Figure 76 The model in hex

To use the model, CUBE AI gives us auxiliary functions and variables, and we need to use them. There was not much documentation for these, so it was hard to figure out how to use them.

I wrote the code, which converts the measured data into the required form, and it uploads it for the network. It prints out the results, which are 5 predictions. It finds the maximum value of these predictions, and it print out as the prediction of the model.

```
0 Output: 0.576942
1 Output: 0.224674
2 Output: 0.000000
3 Output: 0.198382
4 Output: 0.000002

0
clockwise-circle
```

Figure 77 prediction of the microcontroller

In our case, although it guessed well, it was not 99.99% sure, only 57%.

I also took 20-20 measurements on this model and looked at how many predicted well:

- clockwise-circle: **16** / 20
- counter-clockwise-circle: **12** / 20
- twist: **20** / 20
- train: **19** / 20
- wave: **18** / 20

The accuracy is possibly **85%**. It has a problem with, especially between circles, is not being able to decide in which direction the movement took place.

In our case, we predict much faster, than when recording to a computer, because sending the data to the computer take a lot of time, we send the data through **UART** at a rate of **115200 Baud**. We transmit characters with a size of **1 byte**, the number of characters depends on how many digits the measured data consists, for example, if the measured data is one digit, then it is one byte, if it is 3 digits then 3 bytes. It transmits an average of about **35 bytes** per measurement. We perform **850** measurements per sensor, **1700** measurements for 2 sensors, **1700 * 35 bytes** for **59500 bytes**, this is multiplied by **8** for **476000 bits**, so it takes a little over **4 seconds** to transmit with **115200 Baud**. This time has already been eliminated by performing the calculation locally, and the continuous use of UART does not consume a lot of energy, so this solution is more energy efficient.

7 Experimenting with efficiency

Examining the samples more closely, we can see that there is a more marked signal change at the beginning of the samples. If we would take shorter samples, the remaining memory area would also increase severalfold, because of the size of the arrays, which are needed to transform the data to correct form are also decreasing, and the size of ANN would also decrease as a result of the input layer decreasing, so more neurons could be used to build the neural network, it means we can use more complex models.

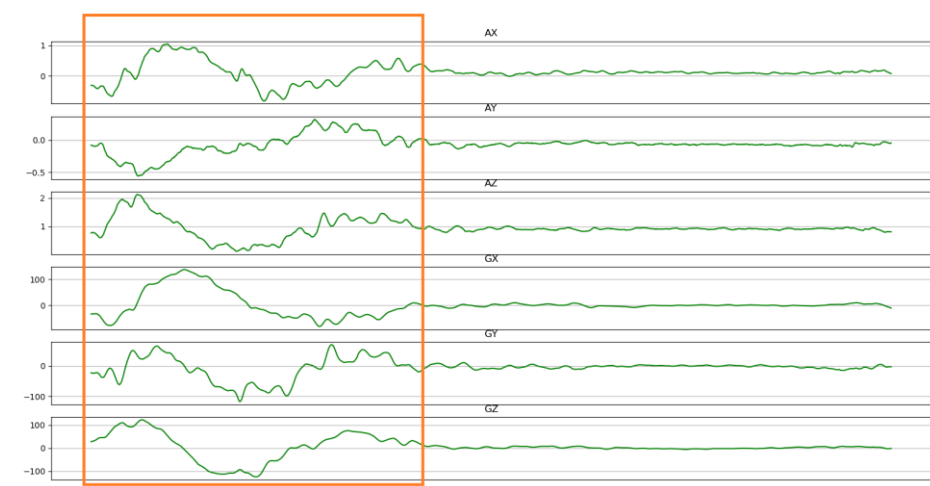


Figure 78 The beginning of the samples is the most significant

I read 1020 bytes 5 times, so I need to do 5 measurements, I have already measured the first one, its efficiency was **85%**. I will see what happens if I only record $4 * 1020$, $3 * 1020$, $2 * 1020$, $1 * 1020$ bytes, but invest the earned memory in the size of the model. It is expected that at first the efficiency will start to increase and then it will suddenly decrease, because sooner or later I will start to cut useful things from the recorded samples.

The size of the samples are fixed in length, the longest pattern is labeled with a wave. It is expected to be the first sample whose efficiency will be fall during the reduction of the length.

7.1 Efficiency of model2

Before I created the model, I had reduced the size of the samples to 4/5 and increased the size of the model to improve its efficiency.

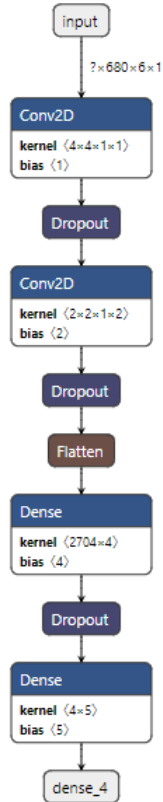


Figure 79 model2

162 KB

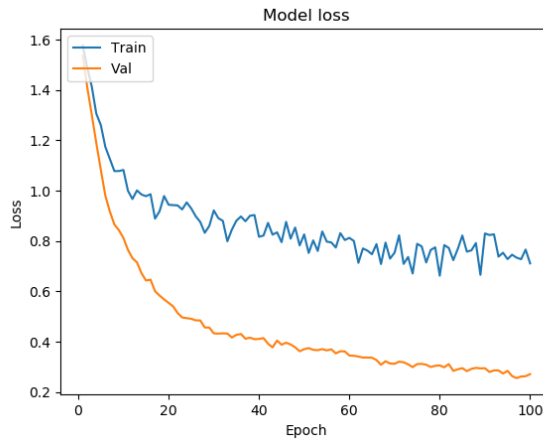


Figure 80 model2 loss

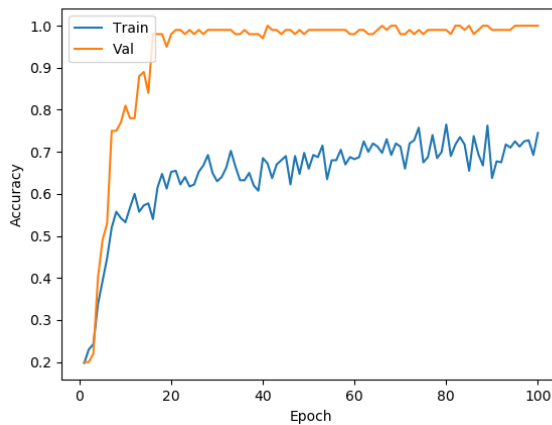


Figure 81 model2 accuracy

The size of the model increased to **162 KB**, the efficiency apparently increased by a tiny bit according to the statistics, which have generated during teaching.

I also took 20-20 measurements on this model and looked at how many predicted well:

- clockwise-circle: **19** / 20
- counter-clockwise-circle: **13** / 20
- twist: **17** / 20
- train: **20** / 20
- wave: **20** / 20

Efficiency increased to **89%**.

7.2 Efficiency of model3

Before I created the model, I had reduced the size of the samples to 3/5 and increased the size of the model to improve its efficiency.

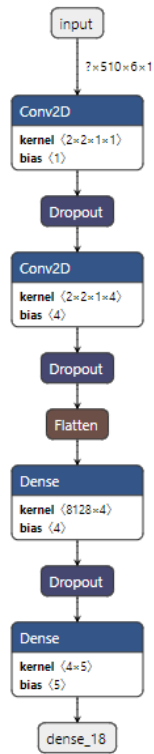


Figure 82 model3

416 KB

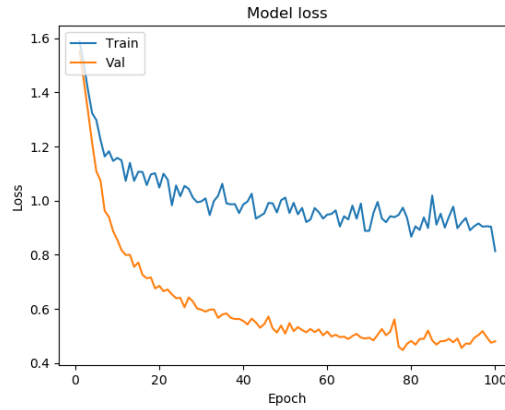


Figure 83 model3 loss

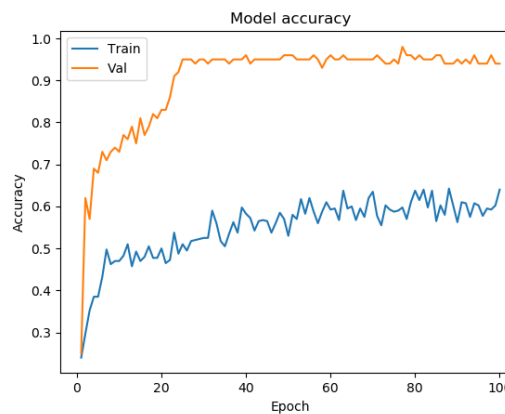


Figure 84 model3 accuracy

The size of the model increased to **416 KB**, the efficiency apparently decreased by a tiny bit according to the statistics, which have generated during teaching.

I also took 20-20 measurements on this model and looked at how many predicted well:

- clockwise-circle: **9 / 20**
- counter-clockwise-circle: **8 / 20**
- twist: **20 / 20**
- train: **19 / 20**
- wave: **2 / 20**

Efficiency decreased to **59%**, we have already reached our peak efficiency, we have probably cut off important parts of the wave pattern.

7.3 Efficiency of model4

Before I created the model, I had reduced the size of the samples to 2/5 and increased the size of the model to improve its efficiency.

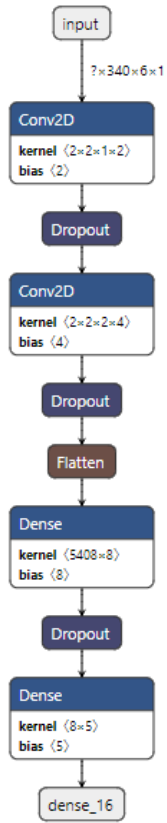


Figure 85 model3

542 KB

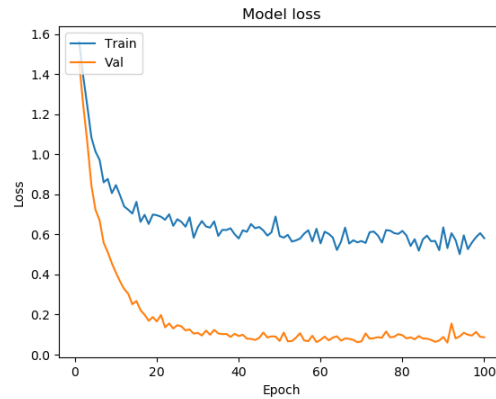


Figure 86 model4 loss

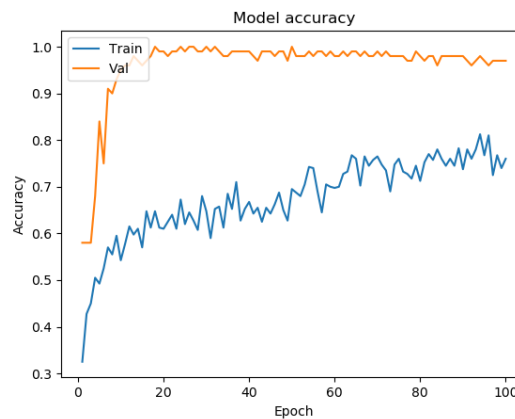


Figure 87 model4 accuracy

The size of the model increased to **542 KB**, the efficiency apparently increased by a tiny bit according to the statistics, which have generated during teaching.

I also took 20-20 measurements on this model and looked at how many predicted well:

- clockwise-circle: **7** / 20
- counter-clockwise-circle: **0** / 20
- twist: **20** / 20
- train: **0** / 20
- wave: **2** / 20

Efficiency decreased to **27%**, and probably it will be worse, if we decrease the length more.

7.4 Efficiency of model5

Before I created the model, I had reduced the size of the samples to 1/5 and increased the size of the model to improve its efficiency.

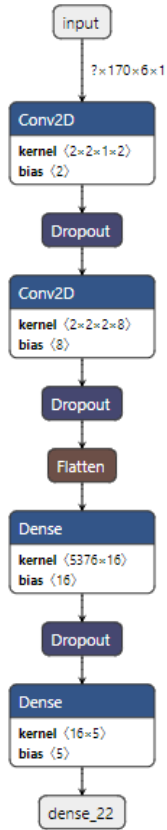


figure 88 model5

1 MB

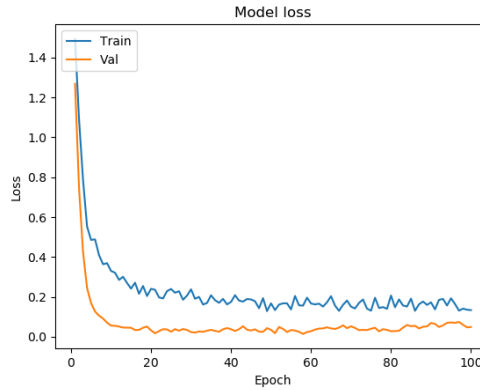


Figure 89 model5 loss

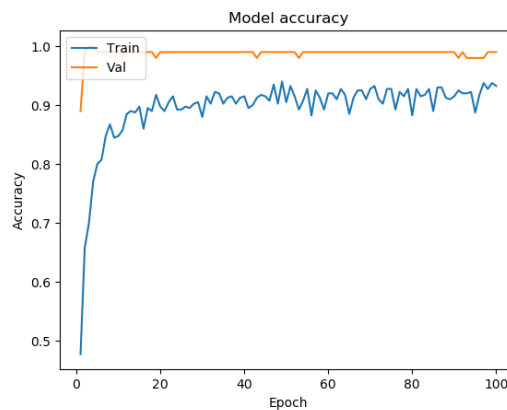


Figure 90 model5 accuracy

The size of the model increased to **1 MB**, the efficiency apparently increased a lot according to the statistics, which have generated during teaching.

I also took 20-20 measurements on this model and looked at how many predicted well:

- clockwise-circle: **7 / 20**
- counter-clockwise-circle: **0 / 20**
- twist: **20 / 20**
- train: **8 / 20**
- wave: **20 / 20**

Efficiency increased to **67%**, it was a huge astonishment, that means the complexity of the model plays a key role in efficiency.

7.5 Evaluation of results

I plotted the measured results on a diagram:

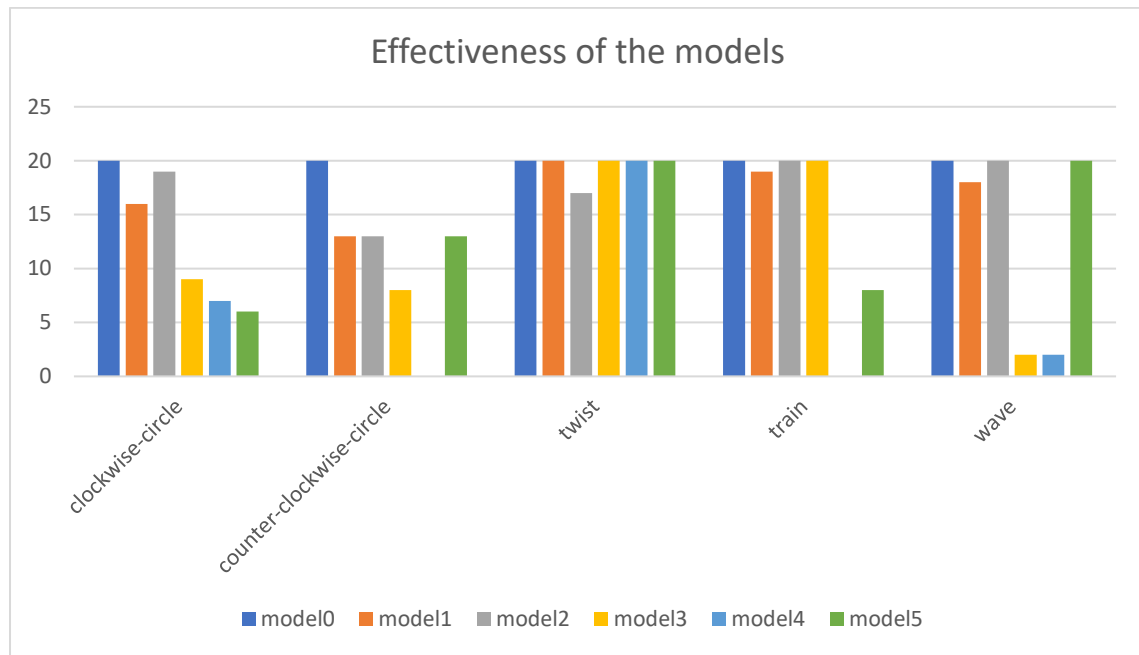


figure 91 Effectiveness of the models

Observing the efficiency of the models, the most surprising results was the efficiency of model5, because after model4 I expected its efficiency to converge to around 0, but nevertheless an increase in performance was observed, in that case we achieved even better results than model3.

Although I calculated the effectiveness of the models by including each tag with equal weight in the aggregation, a kind of personal characteristic can still be observed for each tag.

The twist was the best-hit tag, which may have been, because one distinct pattern, vastly different from the others, was easier to distinguish. For clockwise-circle and counter-clockwise-circle, the uncertainty was constant for all models. The biggest oscillation was for the train and the wave, some models took the obstacle well, some models were almost unable to determine that output. The wave was the one that lasted the longest, when choosing the length of the samples I mainly considered that it would still fit, so the time window was originally 3 seconds. Reducing this slightly, it was not surprising that the sample did not become undetectable, but in the case of model4, we have already cut off important parts of the sample. Nevertheless, we were again able to determine with high accuracy with model5.

I originally planned no more measurements, but the result of model5 was surprised and I wanted to do some more measurements, during which I did not reduce the length, but I take the most successful time window model, in our case model2 and I measure with that window size so that I reduced the frequency once to half and once to a quarter.

8 Frequency reduction tests

First, let us look at a sample at the original 400Hz, then the trisected 140 Hz, then the further halved 70 Hz, and finally the halved once more 35 Hz.

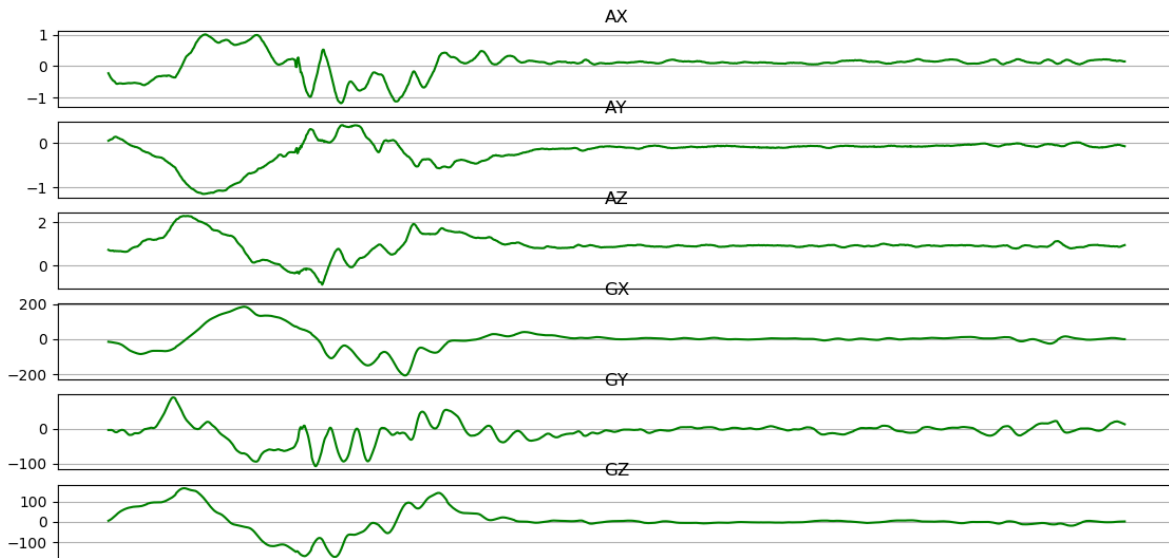


Figure 92 A sample with 400Hz sampling frequency

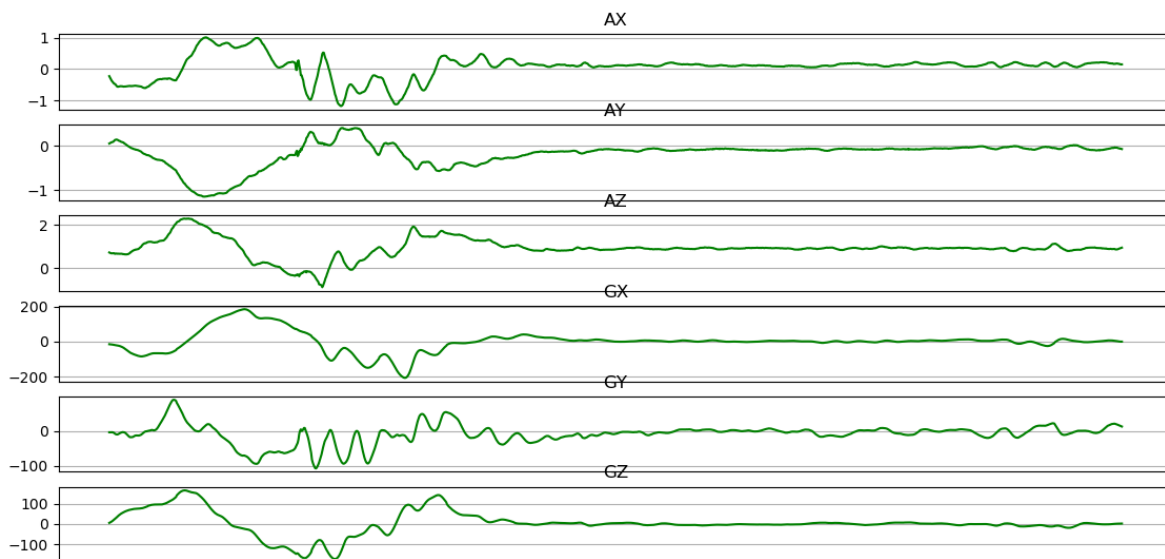


Figure 89 A sample with 140 Hz sampling frequency

There was little visible difference between the 400 Hz and 140 Hz sampling, when we look at it with bare eye, and from the information could not be lost too much.

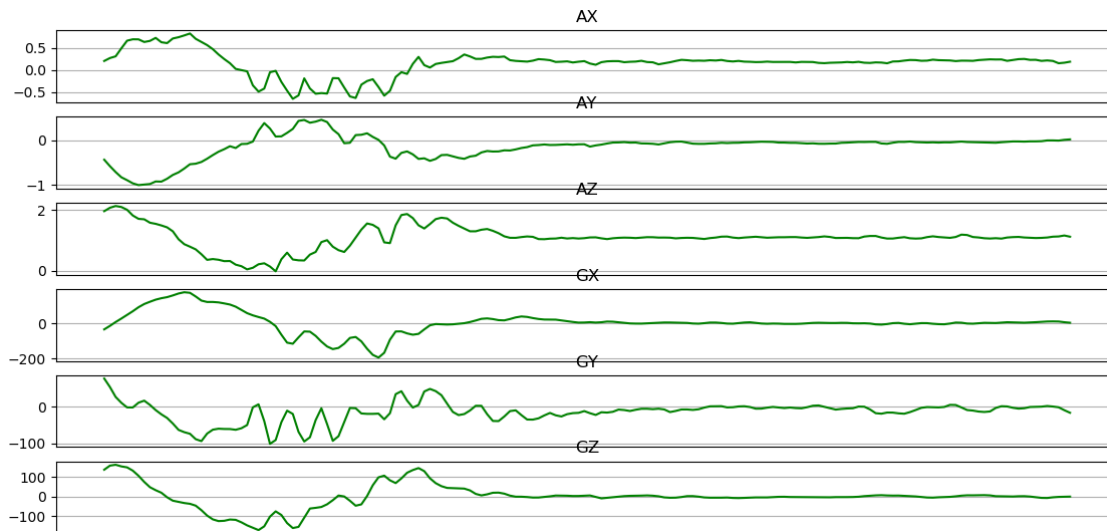


Figure 93 sample with 70 Hz sampling frequency

At 70 Hz, it noticeably cuts things off the figure.

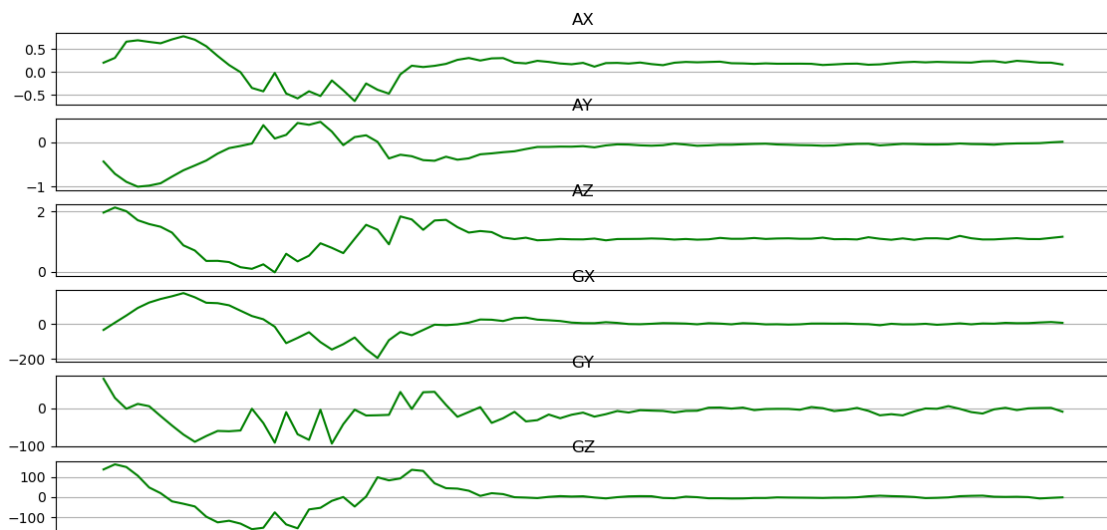


Figure 94 sample with 35 Hz sampling frequency

At 35Hz, the picture is almost completely square, hopefully it still contains the information.

8.1 Efficiency of model6

Before I created the model, I had halved the frequency and increased the size of the model to improve its efficiency.

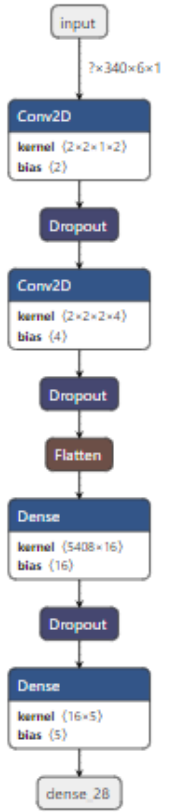


Figure 95 model 6

1 MB

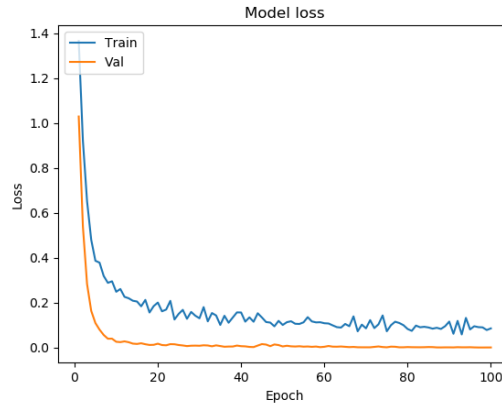


Figure 96 model6 loss

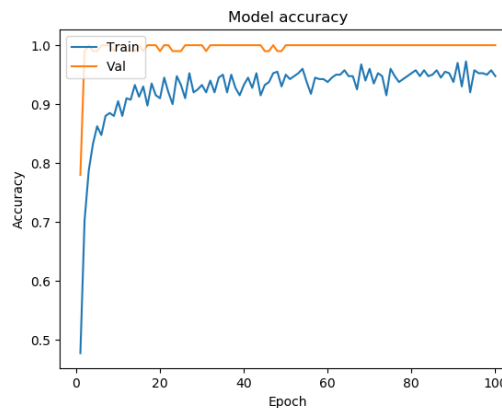


Figure 97 model6 accuracy

The size of the model increased to **1 MB**, the efficiency apparently increased by a lot according to the statistics, which have generated during teaching.

I also took 20-20 measurements on this model and looked at how many predicted well:

- clockwise-circle: **8** / 20
- counter-clockwise-circle: **13** / 20
- twist: **20** / 20
- train: **20** / 20
- wave: **7** / 20

Efficiency decreased to **68%**, we have probably cut off important parts from samples.

8.2 Efficiency of model7

Before I created the model, I had quartered the frequency and increased the size of the model to improve its efficiency.

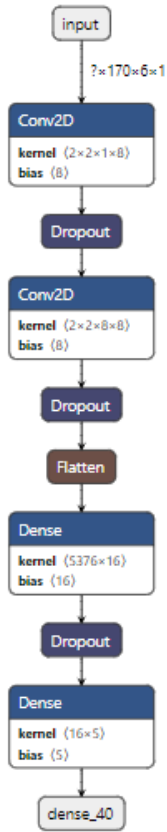


Figure 98 model7

1 MB

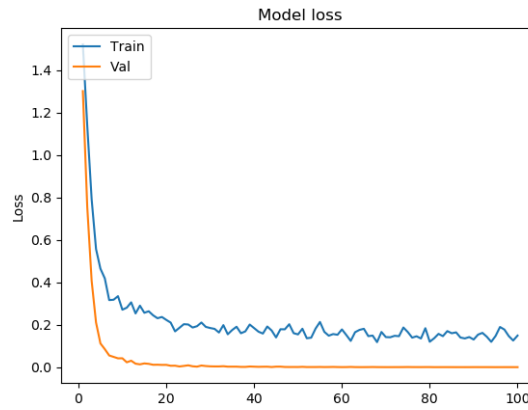


Figure 99 model7 loss

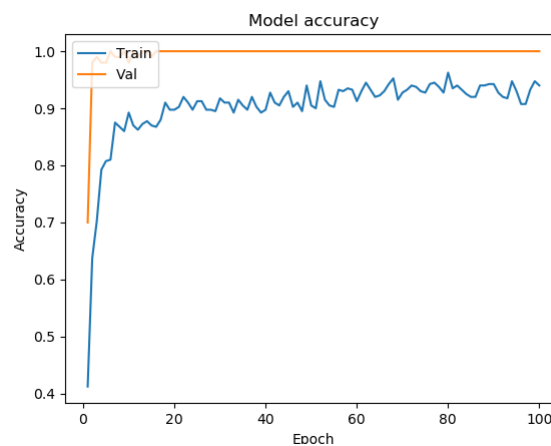


figure 100 model7 accuracy

Although the size of this model is **1 MB**, as the input layer decreases, the model is more complex, a larger model can no longer fit on the microcontroller. I also took 20-20 measurements on this model and looked at how many predicted well:

- clockwise-circle: **9** / 20
- counter-clockwise-circle: **18** / 20
- twist: **18** / 20
- train: **12** / 20
- wave: **18** / 20

Efficiency increased to **68** it was also a huge astonishment, that is another proof for the complexity of the model plays a key role in efficiency.

8.3 Evaluation of the results

Although the decrease in frequency had a negative effect on the results, it was a surprising result that the quarter-frequency model performed better than the halved one. From this, it can be concluded that the results can also be improved by minimally increasing the complexity of the model, so that you probably do not need a quarter GB of storage space for a working model.

Albeit I assigned relative efficiencies to the models as to what percentage were effectiveness, we still can noticed that each model achieved different efficiencies on each sample, on this basis we should look at the model efficiencies on each label.

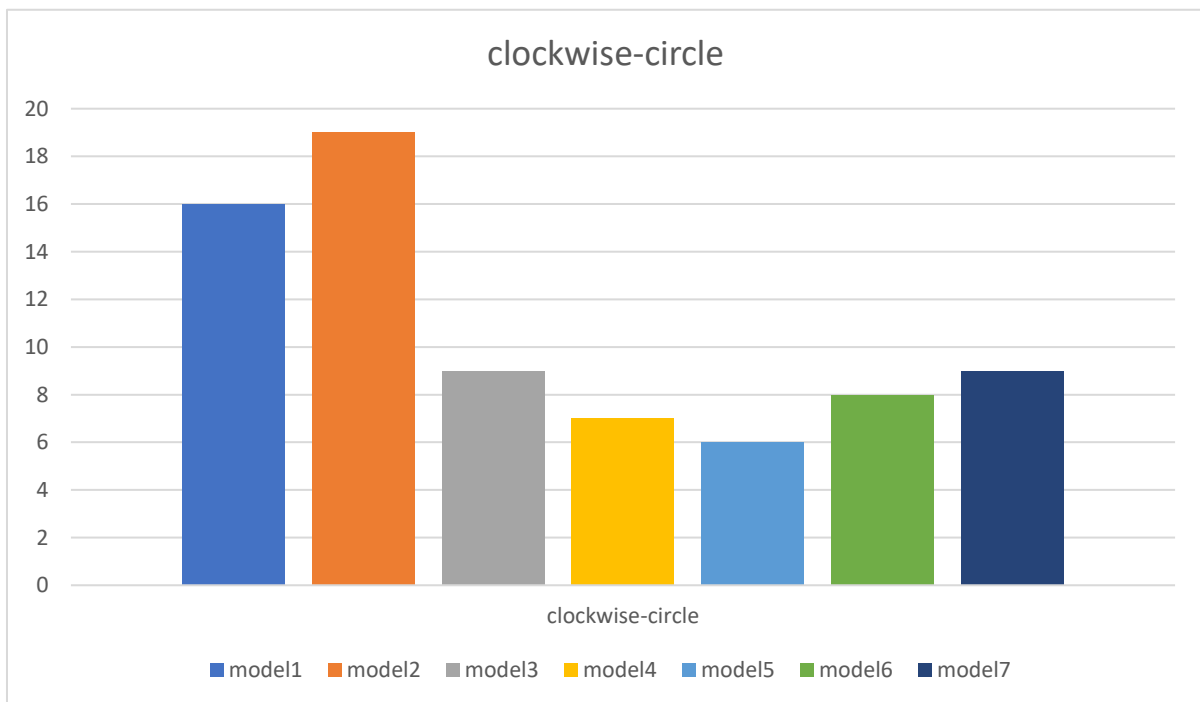


Figure 101 clockwise-circle accuracy on models

This sample needed a decent sampling frequency, when I reduced the time window just a little bit, it caused no problem as we can see on the efficiency of model2, but it is important to do the full movement, the full circle, it is difficult to make an accurate prediction only from the beginning of the sample.

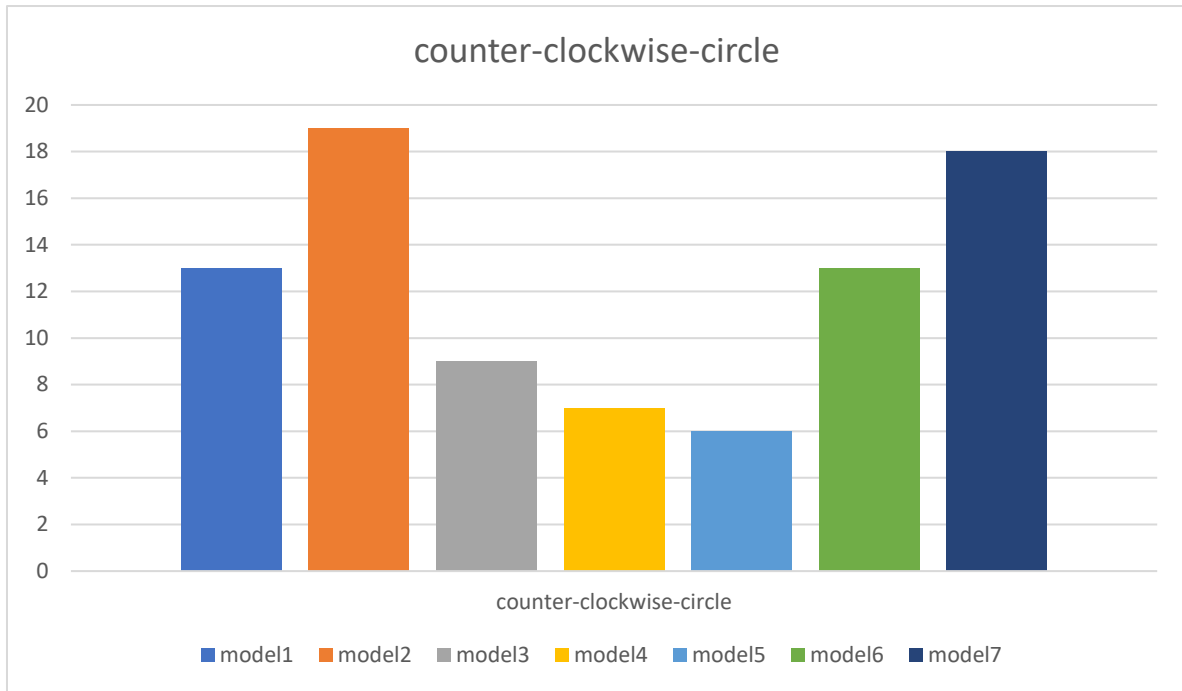


Figure 102 counter-clockwise-circle accuracy on models

The same is true for this pattern as clockwise-circle. The reason that model6 and model7 showed greater accuracy than in the case of the clockwise-circle is that the resulting model tended to look at everything counter-clockwise-circle, which may have been due to the fact that the smoothed samples noise was also very similar to counter-clockwise-circle. If I left it on the table and only moved it a little, it always gave a counter-clockwise-circle as prediction.

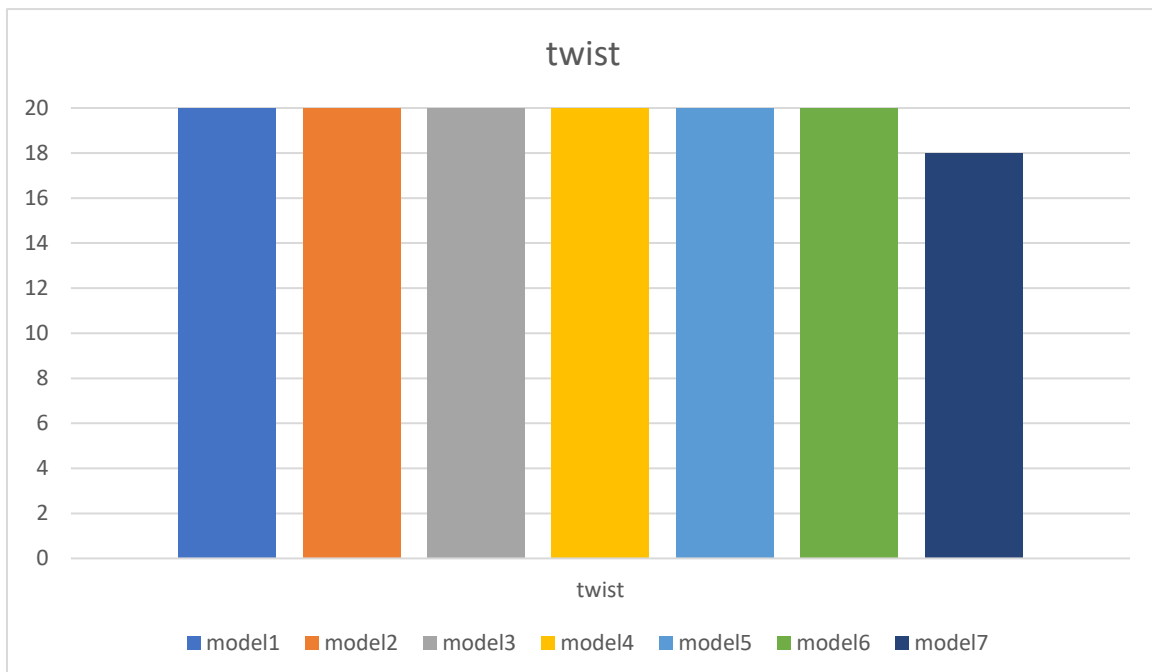


Figure 103 twist accuracy on models

This pattern is the most characteristic, this was very well recognized by all models, only model7 had less uncertainty, but 90% of the models could predict it perfectly.

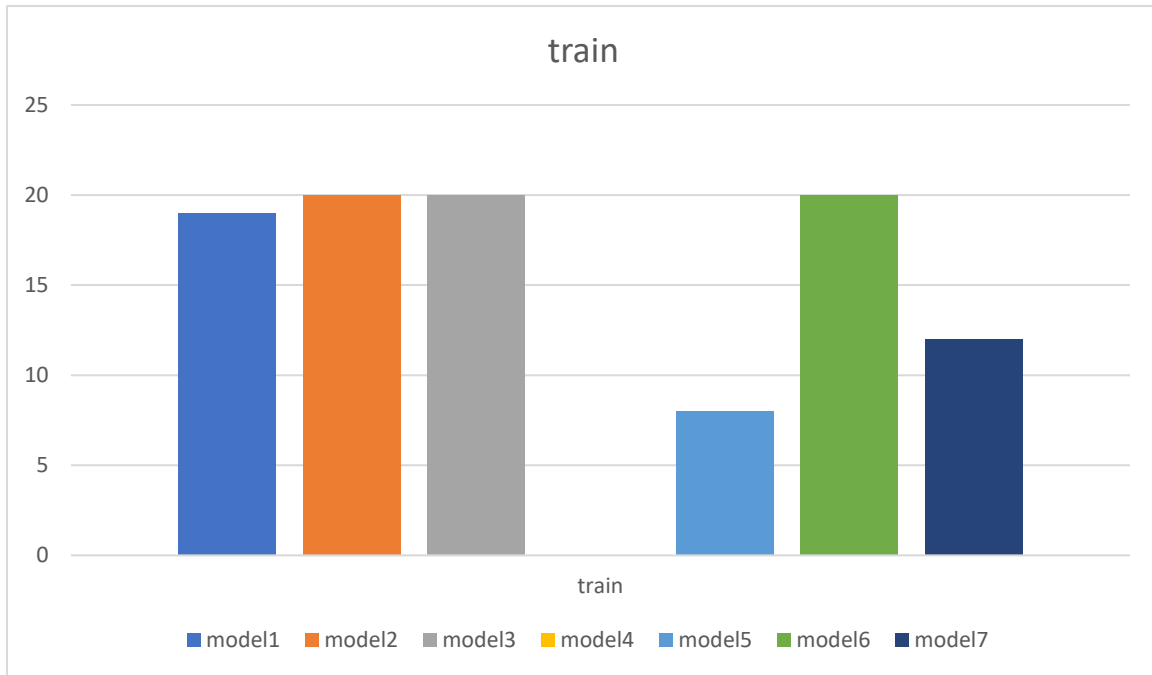


Figure 104 train accuracy on models

In the case of this sample, as the measurement time interval decreased, the efficiency did not decrease at first, but then dropped suddenly. Frequency drop do not have effect on efficiency after first halved, but 35 Hz seemed a little bit bare for it.

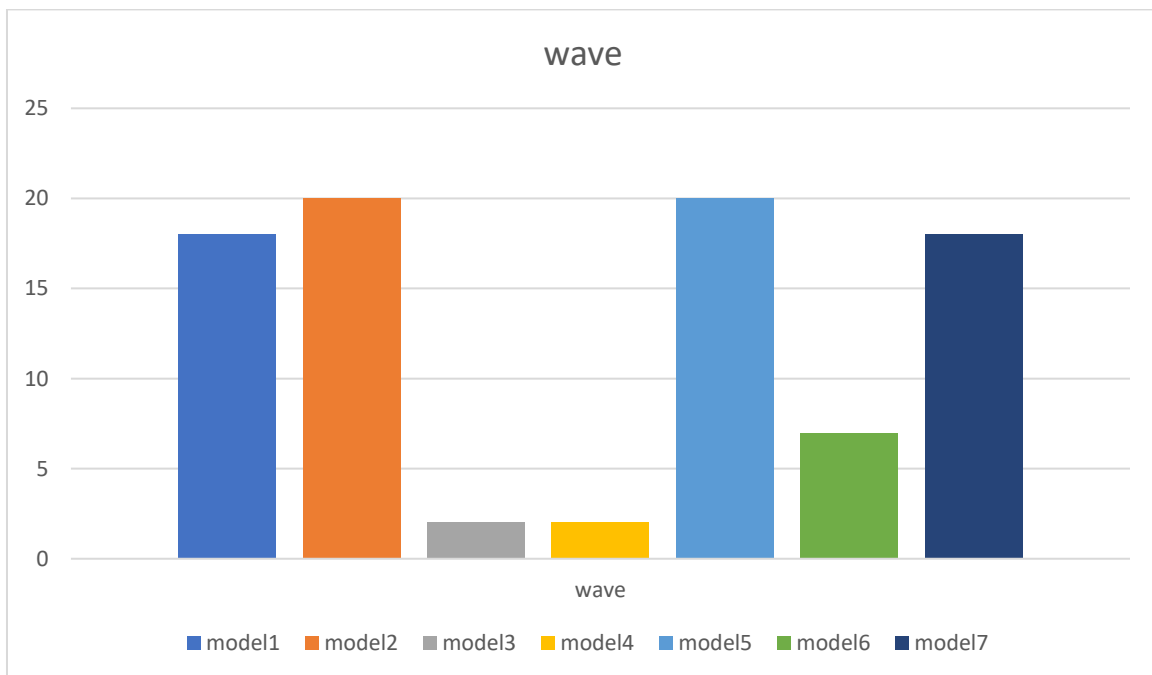


Figure 105 wave accuracy on models

It was the most complex pattern, and it caused the biggest surprises. As the sample time interval decreased, its recognizability dropped to around 0 very abruptly, and then when I increased the strength of the model, it went back. In the case of model5 and model7, it can be noticed that the complexity of the model is more important than the content of the samples.

9 Conclusion

In today's world, we use larger and larger models for deep learning, but the memory of microcontrollers is finite, but that does not mean microcontrollers cannot take their share of artificial intelligence. Although one direction is to imitate human thinking better and better by using larger and larger models, the other direction is to shrink the models enough to run it on a microcontroller environment. They are coming up with better and better solutions in this area as well. For example ARM and Google work together on tinyML, a fast growing field of machine learning technologies and applications, that enable ultra-low power ML at the edge

In our case, we wanted to upload a 238 MB ANN to a 512 KB flash memory and 128 KB RAM. At first it seemed like an unsolvable problem, but a lot of measurements proved that it could be done with relatively good accuracy. We had quite a bit of memory, but we were still able to make a lot of improvements with minimal changes. You can see in figure46 that there are microcontrollers that have twice the amount of memory. They can be loaded data with higher sampling rate, they can use more complex models, and even if this amount of memory is not enough, we can use external memory as a supplement.

MUCs are already quite suitable for running simple neural networks. The memory of microcontrollers is getting bigger and bigger, we can compress the models smaller and smaller in size. Perhaps in the future we will be able to apply increasingly complex, increasingly sophisticated deep learning solutions on microcontrollers, thus makes smarter the objects around us.

It seems like a beautiful future, isn't it?

Bibliography

1. Áron, H. M. (2020). *Önálló laboratórium beszámoló*. Budapest.
2. Crisp, J. (1998). *Introduction to Microprocessors and Microcontrollers*.
3. Fisher, D. M. (2016). *ARM Cortex M4 cookbook : over 50 hands-on recipes that will help you develop amazing real-time applications using GPIO, RS232, ADC, DAC, timers, audio codecs, graphics LCD, and a touch screen*.
4. Frenzel, L. (2015). *Handbook of Serial Communications Interfaces: A Comprehensive Compendium of Serial Digital Input/Output I/O Standards*. Newnes.
5. Horvath, M. A. (dátum nélkül.). *Témalaboratórium beszámoló*.
6. Ivan Vasilev, D. S. (2019). *Python Deep Learning: Exploring deep learning techniques, neural network architectures and GANs with PyTorch, Keras and TensorFlow*.
7. John Paul Mueller, L. M. (2018). *Artificial Intelligence for Dummies*.
8. Kimmo Karvinen, T. K. (2014). *Getting Started with Sensors: Measure the World with Electronics, Arduino, and Raspberry Pi*. Maker Media, Inc.
9. Moolayil, J. (2019). *Learn Keras for Deep Neural Networks*.
10. Moolayil, J. J. (2019). *Learn Keras for Deep Neural Networks: A Fast-Track Approach to Modern Deep Learning with Python*. Apress.
11. Norris, D. (2018). *Programming with STM32: Getting Started with the Nucleo Board and C/C++*.