



Ministère de l'Enseignement Supérieur et de la Recherche Scientifique



Université des Sciences et de la Technologie Houari Boumediene

**Faculté d'Electronique et d'Informatique
Département Informatique**

Mémoire de Licence

Filière: Informatique

Spécialité: ISIL

Thème

Conception and realization of a motion control system based on the wrist:

A smart-watch function

Sujet Proposé par :

Mr BABA ALI Riadh & Mme MADI Sarah :

Soutenu le :.../.../2021

Présenté par : TOLBA Yasmine & TOUMACHE Doudja Rania

Devant le jury composé de:

M..... Président (e)

M..... Membre

Binôme n° : 33 / 2021

Special thanks & acknowledgement:

We would like to thank and show our sincere gratitude to everyone who helped us throughout this project, starting from our supervisors Pf. Baba Ali Riadh and Mme. Madi Sarah, for guiding and providing us with invaluable advice and everything we needed in order to do the best job we could, in both knowledge and resources. as well as giving us countless words of encouragement and support that were crucial to us.

And of course, our families that provided us with support in the entirety of our studies that pushed us to always go beyond our limits and always do our best in whatever we work on.

We would also like to thank our friends who showed an immense amount of support while working hard and always inspiring us to work harder and do better.

Table of contents :

0. Chapter 0: Introduction.....	3
0.1. Intro:	3
0.2. Problem:	3
0.3. Objective:	3
1. Chapter 1: State of the art.....	4
1.1. Machine Learning:.....	4
1.2. Supervised Learning:	4
1.2.1. Regression:	4
1.2.2. Classification:	5
1.2.3. Steps of Supervised learning :	5
1.3. The KNN Algorithm:	6
1.4. Weka:	7
1.5. Motion sensors:	9
1.5.1. Accelerometer:	10
1.5.2. Gyroscope:	10
1.6. MPU6050:	11
1.6.1. Features & components:	11
2. Chapter 2: Conception	13
2.1. Methodology:.....	13
2.2. Arduino with MPU6050:	14
2.3. Arduino Code:	15
2.4. Preprocessing the data:	17
2.5. Classification with python:.....	19
3. Chapter 3: Results	20
3.1. Arduino raw data results:	20
3.2. Gathered Dataset:	21
3.3. Preprocessed Dataset:.....	21
3.4. Result dataset in Weka:	22
3.5. Classification with Python results:.....	27
4. Conclusion:.....	28
Sources:	29
Bibliography :	31

0. **Chapter 0: Introduction**

0.1. **Intro:**

Motion tracking is an emerging topic in today's technologies, it is the analysis and interpretation of human motion, using mathematical algorithms.[1] The motion in question can be any body movement, but the two most commonly used ones, are the face and wrist.

Movement recognition is mainly paired with artificial intelligence and machine learning. This opens up a broad world for Human-Machine Interaction, without even having to make contact with the device itself, which will make easier, more intuitive use of technology possible.

In today's world, connected devices like smartwatches and bracelets, a small portable object that comfortably sits at our wrists. Their convenient placement is perfect for gesture recognition. They are equipped with motion sensors, which means that they can make a wide range of different calculations and measurements, all related to the motion of the wrist and its position, like the angle, the orientation, acceleration, and overall movement. By using these assets, with the help of machine learning, we are able to make a reliable, accurate, and more importantly, intelligent way of motion recognition. whether it is a simple movement like waving your arm, to a more complex one like a slight movement or a twist of the wrist.

All of this is possible thanks to all kinds of technologies, like accelerometers, gyroscopes, compasses, and a few other sensors. However, we need all of these devices to fit in a tiny space, that is why manufacturers have to maximize the efficiency of space usage in the smart-devices, and miniaturize the different chips, modules, and sensors that will be added to the small motherboard.

0.2. **Problem:**

Smartwatches' technology, as advanced as it seems, still has a lot of room for improvement. For example, sometimes when the user wants to see the time on their device, the watch won't turn on despite doing the correct movement, like bringing their arm upward and turning their wrist in the direction of their face, thus making the user repeat their movement multiple times, and other times, the watch will turn on when not needed, when scratching their head or extending their arm for example, which leads to a waste of energy and battery.

0.3. **Objective:**

The objective of this project is to use KNN algorithm to identify and recognize wrist movements to enhance the reliability, efficiency and accuracy of movement detection in smartwatches to save both energy and time.

Our system will classify the movements using KNN based on a gathered training set. For that, we used different motion sensors (gyroscope and accelerometer) to collect values of our data that will be used to train the model in the best way possible to get the most accurate, precise and reliable result in the "time checking" gesture of the wrist.

1. **Chapter 1: State of the art**

1.1. **Machine Learning:**

Machine learning is an application of artificial intelligence that provides the systems with the ability to automatically learn from data that has been fed into it and improve their accuracy over time without being explicitly programmed. Machine learning programs access data, such as examples that we feed into them, and are trained to analyze it, thus finding patterns in order to make decisions and predict correct outcomes for the new data. There are four steps to build a machine learning model :

1. Select and prepare the training dataset.
2. Choose an algorithm to run on the training dataset (such as KNN, decision trees, neural networks...etc.).
3. Train the algorithm to create the model.
4. Use and improve the model [2].

Machine learning examples are all around us nowadays, we see it in recommendations on websites (such as Amazon, Netflix, YouTube...etc.) which recommend products, ads, movies and songs based on what the user has bought, watched or listened to before. We can also find it in fraud detection in banking and self-driving cars. There are many machine learning methods and a popular one is supervised learning.

1.2. **Supervised Learning:**

Supervised learning is the most common sub-branch of machine learning and artificial intelligence. Supervised machine learning algorithms are designed to learn by examples, these examples are also known as labeled or classified datasets which are used to train the learning algorithm to classify unlabeled data and predict outcomes[3]. A supervised learning function can be as simple as : $F(X)=Y$, where X is our input and Y is the result of the prediction such as a class[4]. For example , we can use supervised learning for movement recognition , in our case , it will be to control a smartwatch using the motion of the wrist.

Supervised training uses a training set to teach models to yield the desired output. This training set includes inputs and correct outputs , which allow the model to learn over time. The algorithm measures its accuracy through a loss function, and it will adjust till the error has been minimized enough. Supervised learning can be split into two subcategories : regression and classification.

1.2.1. **Regression:**

Regression is used to understand and determine the important relationship between dependent and independent variables. The goal of a regression algorithm is to predict and forecast quantitative data (a continuous number) such as sales revenue for a certain business or test score[3]. For example we want to determine a student's grade based on how many hours they studied for the test.

1.2.2. Classification:

It uses an algorithm to classify unlabeled data into specific categories. It analyses and recognizes patterns and specific entities to determine how these entities should be labeled and defined. One of the most common examples of classification is determining if an email is spam or not with two classes to choose from (spam and not spam), this problem is called binary classification problem. There exist numerous and multiple algorithms to solve classification problems. The most popular and commonly used learning algorithms are : linear classifiers, logistic regression, decision trees, neural networks, support vector machines, k nearest neighbors (KNN)[3].

1.2.3. Steps of Supervised learning :

1. Gather data: In supervised learning, we will use a labeled dataset that we will divide into two categories : training dataset and testing dataset. The training dataset will teach our algorithm to predict the correct class, while the testing dataset will be used to test the accuracy of our algorithm but it is not used to train the model.
2. Pre-process the data: we will need a clean dataset to correctly predict the outcomes.
3. Choose an algorithm to run our dataset : In our project we will be using the KNN algorithm to solve a classification problem.
4. Train and test our algorithm.
5. Evaluate our model and improve on it : After training the model, we will evaluate the accuracy with the validation set, depending on the result we will make changes to our model and test once again[5][6].

We can see in the example down below , a flow chart describing the workflow of supervised learning:

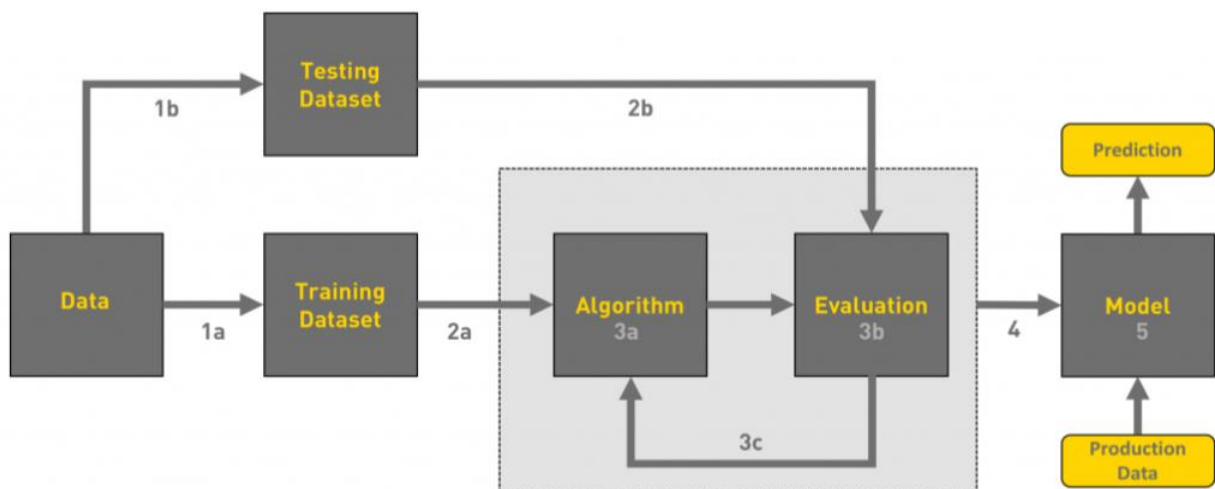


Figure 1 : Flowchart describing supervised learning

1.3. The KNN Algorithm:

if the knn algorithm was a function defined as $f(X,Y,K)=Z$, X being the example dataset, Y the query, and Z the prediction (the result).

Inside this $f(X,Y,K)=Z$ function, the algorithm takes in the X data and calculates the distance between each $X[i]$ and Y which is our query. it will then put all the calculated distances, as well as the index of each compared example in X in an ordered collection and sorts it in an ascending order. Then, we take the K first values in the ordered collection, and the returned result will be the class of the majority of the K taken values, in other words the K nearest neighbors to our query Y .

KNN can be explained as a majority vote like this example below:

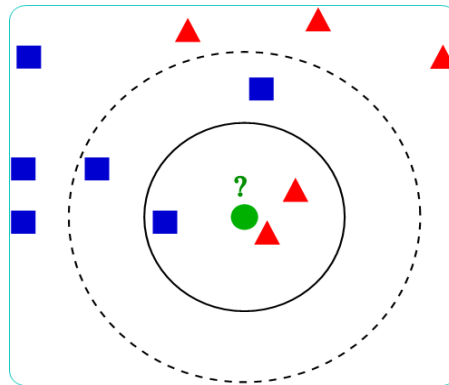


Figure 2: If $k=3$, the query (green spot) will be classified as a red triangle. If $k=5$, it will be classified as a blue square. These results are calculated by the mode (majority)

The predictions will depend heavily on the number K that we decide to assign. Too little of a K value will result in less insight by using less data, too large of a K value will result in errors, so our goal is to assign a suitable K value that will give us more accurate and stable predictions, since we are using classification for our project, an odd K number will be chosen. [7]

- **Steps of the KNN algorithm:**

1. Load the training data
2. Set a value for K
3. For each example in the data:
 - 3.1. Calculate the distance between the query and each row of the training data (Euclidean, Manhattan or Hamming distance)
 - 3.2. Add the calculated distance and the current example's index to an ordered collection

4. Sort the ordered collection in an ascending order (from smallest to largest) in ascending order
5. Pick the K first entries from the sorted ordered collection
6. In classification, return the mode (most frequent occurrence) of K labels.
7. END [7][8]

- **Pseudocode of the KNN algorithm:**

```

KNN(data, query, k)
START
    vector neighbors_DI = []
    float distance

    for each index, example in data:
        | distance = euclidean_distance(example, query)
        | neighbors_DI.append((distance, index))
    end for

    neighbors_DI.sort()
    vector K_nearest = neighbors_DI[:k]
    vector K_labels = []

    for each distance, i in K_nearest:
        | K_labels = data[i][-1]
    end for

    return K_labels.mostcommon()
END

```

In conclusion KNN is a simple and easy algorithm that is also pretty versatile, which makes it quite powerful and capable, however its main issue is speed, which means that the algorithm will get noticeably and significantly slower as we give it more training and example data. This algorithm is widely used in recommendation systems for products and entertainment content. [7]

1.4. Weka:

Weka is an open-source software developed by The University of Waikato in Hamilton, New Zealand. It provides tools for data mining and machine learning.

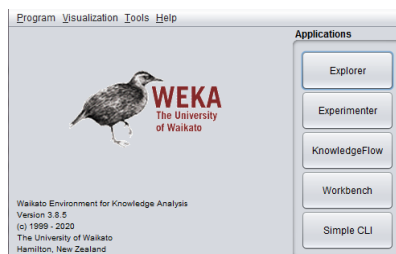


Figure : Weka's main

Weka offers many features, the main one is the Explorer menu. The Explorer gives the user the option to load a dataset and choose between multiple machine learning methods and algorithms to process it. The window shows us all the attributes of our dataset and how many instances there are, there are also multiple tabs for different methods of processing the data. In our study we will use the classification method. As we open this tab we will have to, first, choose which classification method we want to use, as shown in figures below, in our case we will be using the KNN or IBk method. By clicking our choice we get more advanced options for choosing our K which is set to 1 by default. After that, we can adjust the test options as we wish.

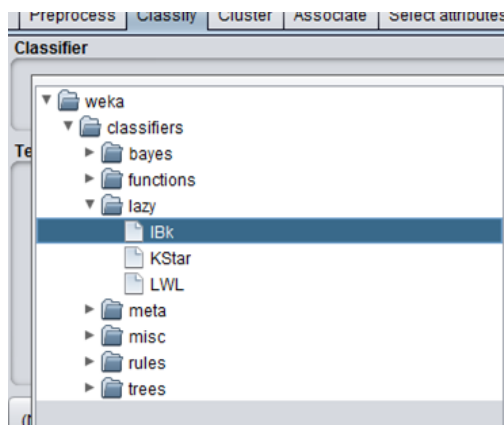


Figure : Classifiers options.

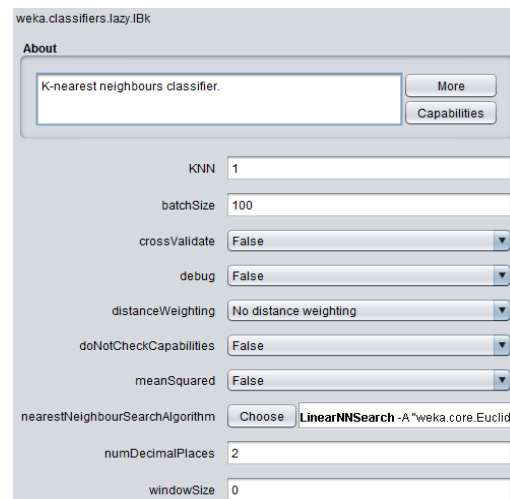


Figure : KNN settings.

- Test Options :

- Use Training set: we use the training set as our testing set.
- Supplied test set: we choose our own testing set.
- Cross-validation: divide the dataset into pieces, then hold out each piece for testing and use the remainder for training, this gives multiple evaluation results which will then be averaged.
- Percentage split: the chosen % will be used as the training set and the remainder for testing.

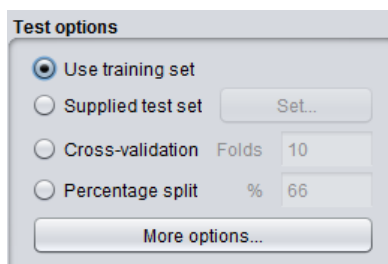


Figure : Test options.

By changing the test options and our K value , we get different results, so it's better to try different settings to choose the one that will suit our needs.

- **Confusion Matrix:**

A confusion matrix is what summarizes the performance of classification algorithms. It takes the number of correct predictions and incorrect predictions for each class then organizes them in a table. In this case , our matrix is a 2x2 matrix, since we have a binary classification problem, each row of the table represents a predicted class , and each column of the table represents an actual class[12].

Real Class	Predicted Class	
	C1	C2
C1	TP	FN
C2	FP	TN

Table 1: Confusion matrix for binary classification.

- C1 and C2 are our classes.
- The first row contains all the samples that are really C1, the seconds row contains all the samples that are really C2.
- The first column contains all the samples that the models classified as C1 and the second one contains all the samples the model classified as C2.
- TP (True Positives) are all the samples that are C1 and were classified as C1.
- TN (True Negatives) are all the samples that are C2 and were classified as C2.
- FN (False Negatives) are all the samples that are C1 that were classified as C2.
- FP (False Positives) are all the samples that are C2 and were classified as C1.

In conclusion , Weka is a strong tool for machine learning that will be used in our project. It includes many useful features that can easily be applied to our datasets to preprocess, manipulate and transform them.

1.5. Motion sensors:

Accelerometers in mobile phones are used to detect the orientation of the phone. The gyroscope, or gyro for short, adds an additional dimension to the information supplied by the accelerometer by tracking rotation or twist. [13]

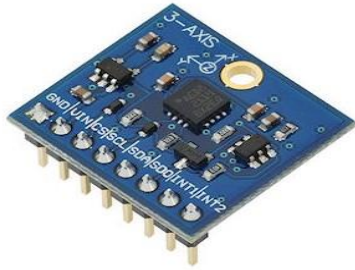


Figure 3: gyroscope module (3 axis)

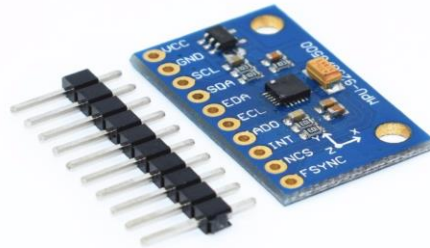


Figure 4: accelerometer sensor and gyroscope module (9 axis)

1.5.1. Accelerometer:

It is a small sensor that is always found in mobile devices such as smartphones and smart watches, and it plays an important role measuring motion inflicted to the device that equips it, and as its name suggests it will calculate the acceleration, as well as the force exerted on it in 3 axes, and the acceleration is quantified in the SI unit meters per second per second (m/s^2) in terms of standard gravity. It will be able to determine the device's position in space and monitor its movement. [17]

All of this is measured using vibration, and of course, motion of the device, by using the Micro-Electro-Mechanical Systems technology, shortened to MEMS. This type is one of the most common types of accelerometers since it is miniscule, and works on the principle of a mass on a spring and fixed electrode plates, measuring the change in electrical capacitance, and is thus called a capacitive accelerometer. [18] [19]

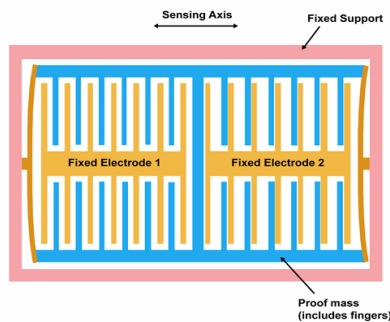


Figure 5: MEMS Capacitive Accelerometer diagram.

1.5.2. Gyroscope:

The gyroscope is a device that helps detect the orientation or deviation of an object, it is composed of a rotating wheel mounted onto a spinning axis in the center[14]. It uses Earth's

gravity to measure the angular velocity along 3 axes which is change of angular position of a rotating body or measurement of speed of rotation. The unit of angular velocity is degrees per seconds ($^{\circ}/s$)[15].

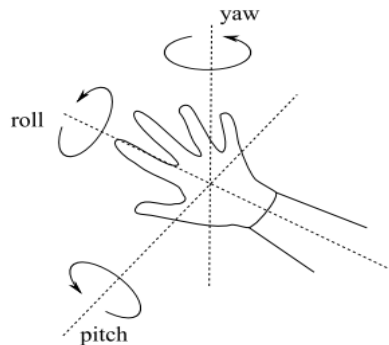


Figure 6: the yaw, pitch and roll coordinate system used by gyroscope to track wrist motion.

An accelerometer measures linear acceleration of movement, while a gyro on the other hand measures the angular rotational velocity. Both sensors measure rate of change, but for different purposes. [13]

In practice, that means that an accelerometer will measure the directional movement of a device but will not be able to resolve its lateral orientation or tilt during that movement accurately unless a gyro is there to fill in that info. [13]

With an accelerometer you can either get a really "noisy" info output that is responsive, or you can get a "clean" output that's slow. But when you combine the 3-axis accelerometer with a 3-axis gyro, you get an output that is both clean and responsive at the same time. [13]

The 3 axes known as roll, pitch and yaw are used to describe the rotation, yaw means left or right, pitch means up or down and roll means the rotation of our object, these three will define the rotational motion of the wrist in our project as shown in figure 6[16].

1.6. MPU6050:

The MPU6050 is a Micro Electro-Mechanical Systems (MEMS)[20] and is the world's first integrated 6-axis Motion Tracking device that combines a 3-axis gyroscope, 3-axis accelerometer, and a Digital Motion Processor™ (DMP) all in a small 4x4x0.9mm package. This device has been adopted by smartphone and tablet manufacturers, since its technology allows the conversion of a normal device to a powerful 3D intelligent one, at the cost of a small package size, low power consumption, high accuracy and repeatability, high shock tolerance, and application specific performance programmability – all at a low consumer price point.[21]

1.6.1. Features & components:

This device is equipped with multiple features as follows:

- MEMS 3-axis accelerometer and 3-axis gyroscope values combined
- Power Supply: 3-5V
- Regulating VDD supply voltage range of 2.375V-3.46V
- Communication through the I²C protocol, which means communication with all registers of the device is performed using I²C at 400kHz
- Built-in DMP (Digital Motion Processor) provides high computational power, 9-Axis Motion Fusion by the on-chip DMP
- Configurable I²C (Inter-Integrated Circuit) Address [20][21]

- Module Pinout:

Pin Number	Pin Name	Description
1	Vcc	Provides power for the module, can be +3V to +5V. Typically +5V is used
2	Ground	Connected to Ground of system
3	Serial Clock (SCL)	Used for providing clock pulse for I2C Communication
4	Serial Data (SDA)	Used for transferring Data through I2C communication
8	Interrupt (INT)	Interrupt pin to indicate that data is available for MCU to read.

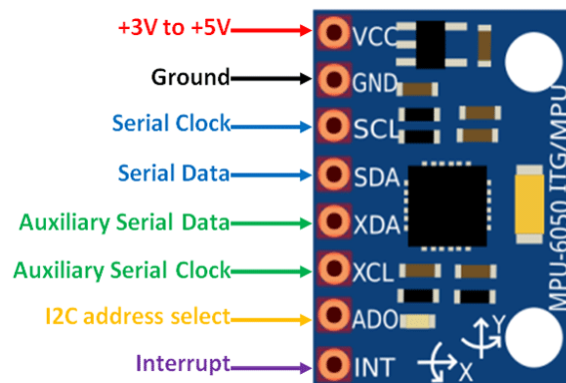


Figure 7: MPU6050 Module Pinout

By pairing the MPU6050 module with an Arduino, we can obtain the Roll, Pitch and Yaw. These are the three values that we shall use in our project. [20]

2. **Chapter 2: Conception**

2.1. **Methodology:**

In order to make a better function of the smartwatch: turning on and off the lock screen to show time, we have equipped ourselves with an Arduino, a breadboard, and an mpu6050.

The first step we went through was attaching the breadboard containing the mpu, and the Arduino to our wrist and correctly wire them. The Arduino is then plugged to our computer using a USB cable, where this will enable the Arduino to receive power, to transfer data, and show it on the serial monitor. Using an Arduino code, we are able to get the values of roll, pitch, and yaw from the mpu6050 in real time on the Arduino app's serial monitor, at a rate of 19200 bauds.

The next step is gathering the data. With the mpu attached to our wrists the same way a smartwatch would, we can start making movements of 2 kinds: correct movements when we want the screen to turn on, and wrong movements when we want the screen to stay off. The wrong movements will allow the AI to differentiate movements that are close in a better way.

After getting the data we need, each single movement will be stored in a txt file, making sure to distinct between the right and wrong movement. Using a python script, we were able to extract the attributes needed which are min, max, average, range, and asymmetry of each of the roll, pitch and yaw of every single movement. The result we get is a CSV file containing all our movements with their respective attributes and class that we will use as our dataset.

The steps that follow are the classification and tests. We will be using two methods: Weka and our own python script. First using weka, the tool explained in chapter 1, we will open the CSV dataset and choose to classify with KNN to do our tests by dividing our dataset into two categories : training and testing. The second method, using our own python script, we will read the dataset and use all of the movements as a training set. We will then plug our mpu with the Arduino to our computer and run the script, we can then proceed to make hand movements we want and the system will classify them in real time, after the tests we can make the necessary changes to the algorithm.

The steps explained above can be summarized in the following block diagram:

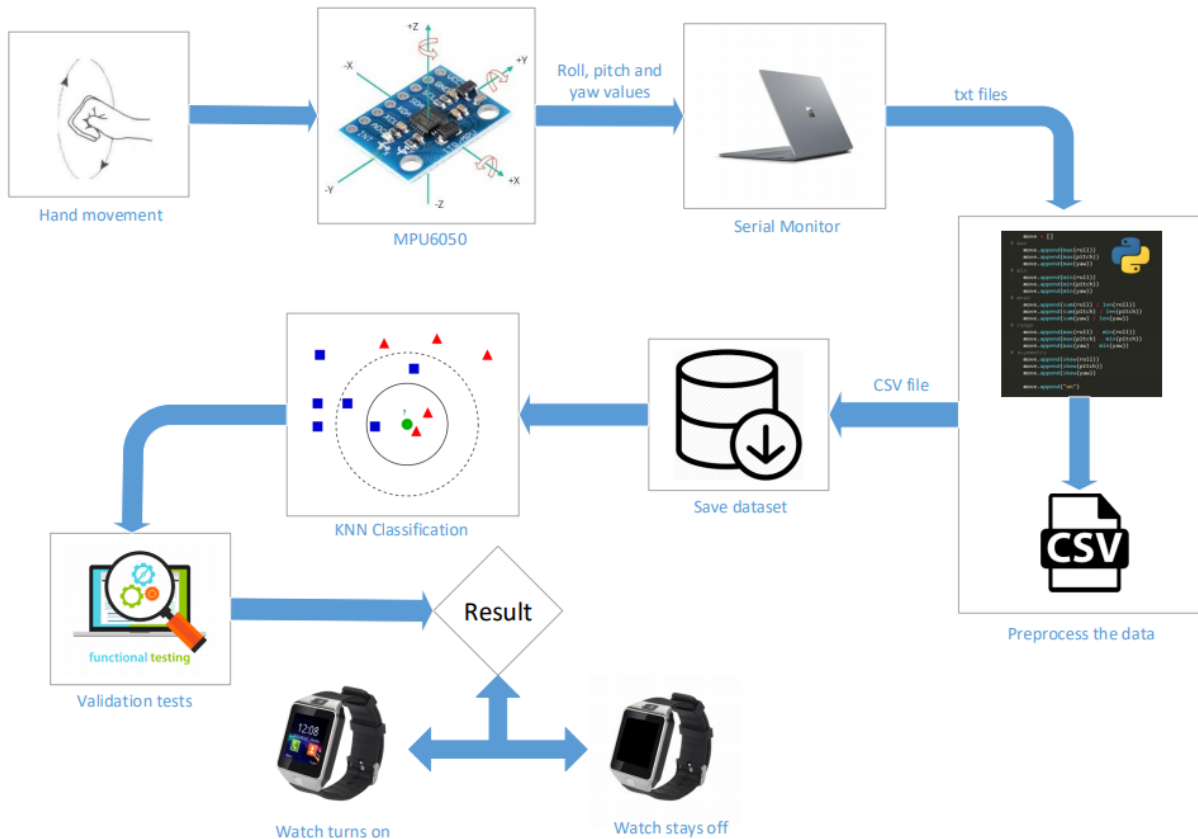
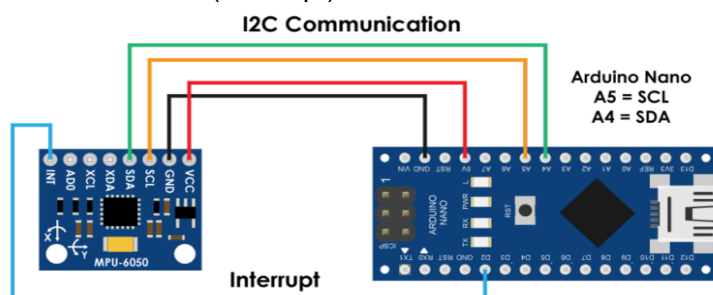


Figure : Block diagram of the system.

2.2. Arduino with MPU6050:

To use our MPU 6050, we will have to pair it with an Arduino, using the I²C protocol. We shall wire the two modules as follows:

- MPU - Arduino Nano
- VCC - 5V (power supply)
- GND - GND
- SCL - A5 (clock)
- SDA - A4 (data)
- INT - D2 (interrupt)



2.3. Arduino Code:

The code used on the Arduino will calculate the values provided by the accelerometer and gyroscope of the MPU 6050 module, and print them on the serial monitor on our screens.[22]

The first step is to set it all up, by declaring all variables and start communicating with the Arduino's registers that we will use, by initializing them. This step also includes calculating the error margin, we shall explain in detail later.

Now in the main, we start off by reading the data from the accelerometer, it will provide us with data from 3 axes (each value stored in 2 registers). The roll and pitch will be calculated using the 3 values on X Y Z axes, and using a mathematical equation, we will have a piece of the roll and pitch as a result.

```
accAngleX = (atan(AccY / sqrt(pow(AccX, 2) + pow(AccZ, 2))) * 180 / PI) - accErrX  
accAngleY = (atan(-1 * AccX / sqrt(pow(AccY, 2) + pow(AccZ, 2))) * 180 / PI) - accErrY
```

atan: arc tangent

sqrt: square root

pow: power

AccX, AccY, AccZ: raw accelerometer values

accErrX, accErrY: accelerometer error margin

For the gyroscope data, we shall use the concept of time in milliseconds, the data is read just like the accelerometer, on 3 axes (X Y Z) and each value stored in 2 registers. We have to fix the units of the values provided by the gyroscope before continuing with the calculations. The Z axis of the gyroscope will provide us with the final yaw value.

```
GyroX = GyroX - GyroErrorX
```

```
GyroY = GyroY - GyroErrorY
```

```
GyroZ = GyroZ - GyroErrorZ
```

```
gyroAngleX = gyroAngleX + GyroX * elapsedTime
```

```
gyroAngleY = gyroAngleY + GyroY * elapsedTime
```

```
yaw = yaw + GyroZ * elapsedTime
```

GyroErrorX, GyroErrorY, GyroErrorZ: gyroscope error margin

GyroX, GyroY, GyroZ: gyroscope raw values

elapsedTime: actual time read - previous time stored before actual time read

The main issue with the gyroscope and accelerometer is every external force working on the object, such as gravity and magnetism, can affect and disturb our measurements, resulting in errors in our readings. We can solve this problem by adding a complementary filter which will combine the gyroscope and accelerometer data and help us minimize the errors and get more precise values.

A complementary filter will be used to raise the accuracy of the roll and pitch values and reduce errors resulting from exterior forces, and eliminate the gyroscope drift error. The way it works is simply by combining the data of the accelerometer with the gyroscope's data (roll and pitch only), in our code we used 96% of gyroscope data and 4% of the accelerometer data.

The general equation to implement a complementary filter is :

$$\text{angle} = (1-\alpha) * (\text{angle} + \text{gyro} * \text{dt}) + (\alpha) * (\text{acc})$$

Where:

$$\alpha = (\tau) / (\tau + \text{dt})$$

dt = 1/fs where fs is your sampling frequency.

gyro is the gyroscope data and acc is the accelerometer data.[23]

So for our code we used the following equations:

$$\text{roll} = 0.96 * \text{gyroAngleX} + 0.04 * \text{accAngleX}$$

$$\text{pitch} = 0.96 * \text{gyroAngleY} + 0.04 * \text{accAngleY}$$

The error margin calculating function is here to help us calibrate the MPU, the accelerometer and gyroscope values for better accuracy, since it is very sensitive to slight changes of environment (gravity, magnetization, etc...). These values will be hardcoded and modified directly in the code's calculations and equations, we calculate it using the 200 first inputs of the MPU, and average them.

For the accelerometer:

(loop 200 times)

$$\text{AccErrorX} = \text{AccErrorX} + ((\text{atan}((\text{AccY}) / \sqrt{(\text{AccX})^2 + (\text{AccZ})^2})) * 180 / \text{PI}))$$

$$\text{AccErrorY} = \text{AccErrorY} + ((\text{atan}(-1 * (\text{AccX}) / \sqrt{(\text{AccY})^2 + (\text{AccZ})^2})) * 180 / \text{PI}))$$

$$\text{AccErrorX} = \text{AccErrorX} / 200;$$

$$\text{AccErrorY} = \text{AccErrorY} / 200;$$

For the gyroscope:

(loop 200 times)

$$\text{GyroErrorX} = \text{GyroErrorX} + (\text{GyroX} / 131.0)$$

$$\text{GyroErrorY} = \text{GyroErrorY} + (\text{GyroY} / 131.0)$$

$$\text{GyroErrorZ} = \text{GyroErrorZ} + (\text{GyroZ} / 131.0)$$

$$\text{GyroErrorX} = \text{GyroErrorX} / 200;$$

$$\text{GyroErrorY} = \text{GyroErrorY} / 200;$$

$$\text{GyroErrorZ} = \text{GyroErrorZ} / 200;$$

[22]

2.4. Preprocessing the data:

In order to use the data that we got from the Arduino, we must preprocess it first, since the raw data doesn't provide us with the necessary information needed for training our knn. To do that, we coded a python script, whose job is to go through all the files, separate the roll, pitch and yaw, calculate the min, max, range, average and asymmetry of each, then save it in a csv file with their respective class attached to them. Each line in the csv file represents one move and one .txt file containing the raw data from the Arduino.

These files will be used to train the knn algorithm in order to classify the movement instances as accurately as possible. Here is the pseudo code of the script in question:

```
START
Create CSV file
For each right file
    Vector roll = []
    Vector pitch = []
    Vector yaw = []
    For each line
        roll.append roll value
        pitch.append pitch value
        yaw.append yaw value
    Close the file
    Vector Move= []
    Move.append max of the roll, pitch and yaw
    Move.append min of the roll, pitch and yaw
    Move.append range of the roll, pitch and yaw
    Move.append mean of the roll, pitch and yaw
    Move.append assymetry of the roll, pitch and yaw
    Move.append (« on »)
    String temp
    For each element in Move
        temp = temp + str(element) + « , »
    Delete the last « , »
    Write temp in CSV file
For each wrong file
    Vector roll = []
    Vector pitch = []
    Vector yaw = []
    For each line
        roll.append roll value
        pitch.append pitch value
        yaw.append yaw value
    Close the file
    Vector Move= []
    Move.append max of the roll, pitch and yaw
    Move.append min of the roll, pitch and yaw
    Move.append range of the roll, pitch and yaw
    Move.append mean of the roll, pitch and yaw
    Move.append assymetry of the roll, pitch and yaw
    Move.append (« off »)
    String temp
    For each element in Move
        temp = temp + str(element) + « , »
    Delete the last « , »
    Write temp in CSV file
Close CSV file
END
```

To make it work, we used modules, glob and scipy. Glob is used in this context:

```
START
Create CSV file
For each right file
    Vector roll = []
    Vector pitch = []
    Vector yaw = []
    For each line
        roll.append roll value
        pitch.append pitch value
        yaw.append yaw value
    Close the file
```

This piece of code loops through the folder, and at every file, it collects all of the values of the roll, pitch, and yaw by splitting each line at the “/” character and puts it all in their respective array. The glob module is used to loop through all the files whose names start with “MPUREAD 1 move right” then with the wild card *, it selects any character or string in the stead of “*”, then the file must end with “.txt”.

The same loop resumes as follows:

```
Vector Move= []
Move.append max of the roll, pitch and yaw
Move.append min of the roll, pitch and yaw
Move.append range of the roll, pitch and yaw
Move.append mean of the roll, pitch and yaw
Move.append assymetry of the roll, pitch and yaw
Move.append (« on »)
```

What this block of code does is that it will calculate the max, min, mean, range and asymmetry of the arrays roll, pitch and yaw, and append the result to an array called move. This move array will represent one single move. The module scipy has been used to calculate the asymmetry with the function skew. Since this loop is dedicated to only the right moves, we will append the string “on” at the end of each move.

Now the last step is to save these values into a file, with each line representing a single move, we made that with this block of code:

```
String temp
For each element in Move
    temp = temp + str(element) + « , »
Delete the last « , »
Write temp in CSV file
```

In this piece of code, we make a temporary string that will contain each element of the array “move”, by appending the string with the element that we need to cast into a string, then concatenate a comma “,” at the end to welcome the next element. Once all the elements have been concatenated onto the string, we remove the last character from it, which is a comma “,” as there is no more element to append anymore. Once our string is ready all we have to do is to write it onto the file we had created at the start of the script with this code block:

```
weka = open("moves for weka.csv", "a")
weka.write("maxroll,maxpitch,maxyaw,minroll,minpitch,minyaw,avgroll,avgpitch,avgyaw,ranroll,ranpitch,ranyaw,asyroll,asypitch,asyyaw,class\n")
```

This piece of code creates a csv file, this file will contain all of the preprocessed data that we will use as training data for the knn algorithm. As it is a csv file, the first line is dedicated to naming the columns, so we write that first line and make sure that our elements follow the same order as these column names.

The code is the same for the wrong movements, the only difference is going to be in the variable file_name, so all we have to do is to redefine it as "MPUREAD 1 move wrong*.txt" and set the last element of the array move, the class, as "off".

We of course make sure to close the file at the end of the script to save it with the code line: weka.close().

2.5. Classification with python:

Now that we have gathered enough data as a training set for knn, we need a program that will take the data from the Arduino and tell us if the screen should turn on or stay off.

In order for that to happen, we need to go through the following steps:

1. Get the raw data from the Arduino (roll/pitch/yaw) if there is no movement, then the device is idle and there is no further data gathered.
2. Preprocess the gathered data into the 15 values that we need (max, min, mean, range, asymmetry of each of the roll, pitch and yaw)
3. Calculate the Euclidean distance between our instance (from the Arduino) and all of the instances in the training set. And sort them in an ascendant order.
4. Take the K nearest neighbors from the sorted neighbors, and classify the instance into the correct class.

Here is the pseudo code explaining how the program works:

```

START
  Read dataset
  Print dataset.details()
  Connect to arduino
  for every 4 seconds:
    vector roll=[]
    vector pitch=[]
    vector yaw=[]
    vector move=[]
    if position=idle
      print("idle")
    else
      gather arduino data
      roll.append roll values
      pitch.append pitch values
      yaw.append yaw values
      calculate attributes and append to Move
      Move.append(' ')
      testdataset=KNN.classify(move,dataset,3)
      print (testdataset)
      empty the roll, pitch and yaw vectors
    end IF
  end FOR
END

```

3. Chapter 3: Results

3.1. Arduino raw data results:

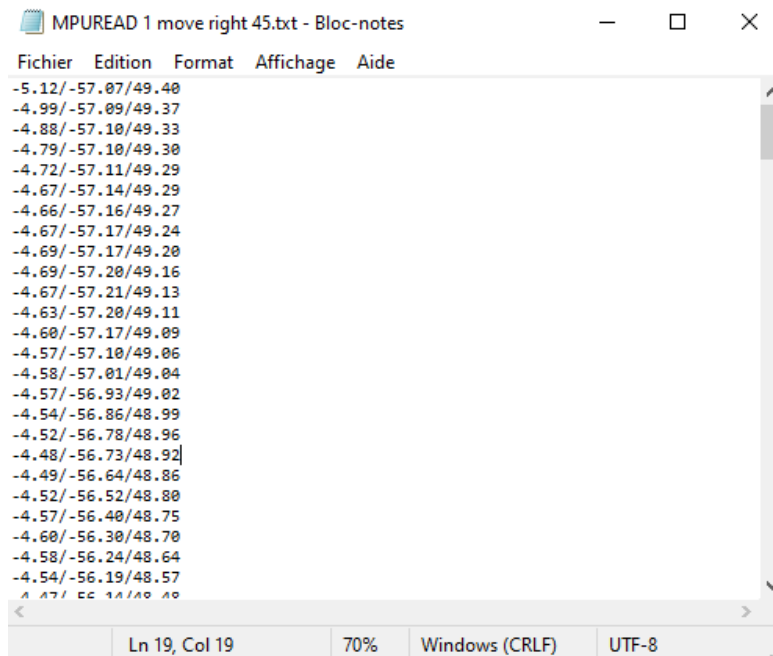
Using the aforementioned code for the Arduino, we were able to get the values of the roll, pitch, and yaw needed for our project. It presents itself on the serial monitor of the Arduino app in an infinite loop like so:



The screenshot shows the Arduino Serial Monitor window titled "COM3". The output displays a series of roll/pitch/yaw values in a loop. The values are: -0.40/0.19/0.12, -0.41/0.18/0.12, -0.42/0.17/0.12, -0.42/0.20/0.13, -0.42/0.20/0.13, -0.41/0.20/0.13, -0.41/0.19/0.13, -0.41/0.22/0.13, -0.42/0.20/0.13, -0.43/0.20/0.13, -0.46/0.20/0.13, -0.43/0.20/0.13, and -0.45/0.21/0.13. The window includes a text input field with an "Envoyer" button, and checkboxes for "Défilement automatique" (checked) and "Afficher l'horodatage". The baud rate is set to 19200.

```
COM3
-0.40/0.19/0.12
-0.41/0.18/0.12
-0.42/0.17/0.12
-0.42/0.20/0.13
-0.42/0.20/0.13
-0.41/0.20/0.13
-0.41/0.19/0.13
-0.41/0.22/0.13
-0.42/0.20/0.13
-0.43/0.20/0.13
-0.46/0.20/0.13
-0.43/0.20/0.13
-0.45/0.21/0.13
```

To get each move we initialize the Arduino on a default position that is on top of the table, then we positioned our hand into the starting point, make a move, and stop there. We then take all of the data from the starting point to the end point, and copy and paste each move into a txt file like so:



The screenshot shows a text editor window titled "MPUREAD 1 move right 45.txt - Bloc-notes". The editor contains a list of roll/pitch/yaw values, each on a new line. The values are: -5.12/-57.07/49.40, -4.99/-57.09/49.37, -4.88/-57.10/49.33, -4.79/-57.10/49.30, -4.72/-57.11/49.29, -4.67/-57.14/49.29, -4.66/-57.16/49.27, -4.67/-57.17/49.24, -4.69/-57.17/49.20, -4.69/-57.20/49.16, -4.67/-57.21/49.13, -4.63/-57.20/49.11, -4.60/-57.17/49.09, -4.57/-57.10/49.06, -4.58/-57.01/49.04, -4.57/-56.93/49.02, -4.54/-56.86/48.99, -4.52/-56.78/48.96, -4.48/-56.73/48.92, -4.49/-56.64/48.86, -4.52/-56.52/48.80, -4.57/-56.40/48.75, -4.60/-56.30/48.70, -4.58/-56.24/48.64, -4.54/-56.19/48.57, and -4.47/-56.14/48.50. The editor has a menu bar with "Fichier", "Edition", "Format", "Affichage", and "Aide". The status bar at the bottom shows "Ln 19, Col 19", "70%", "Windows (CRLF)", and "UTF-8".

```
MPUREAD 1 move right 45.txt - Bloc-notes
Fichier Edition Format Affichage Aide
-5.12/-57.07/49.40
-4.99/-57.09/49.37
-4.88/-57.10/49.33
-4.79/-57.10/49.30
-4.72/-57.11/49.29
-4.67/-57.14/49.29
-4.66/-57.16/49.27
-4.67/-57.17/49.24
-4.69/-57.17/49.20
-4.69/-57.20/49.16
-4.67/-57.21/49.13
-4.63/-57.20/49.11
-4.60/-57.17/49.09
-4.57/-57.10/49.06
-4.58/-57.01/49.04
-4.57/-56.93/49.02
-4.54/-56.86/48.99
-4.52/-56.78/48.96
-4.48/-56.73/48.92
-4.49/-56.64/48.86
-4.52/-56.52/48.80
-4.57/-56.40/48.75
-4.60/-56.30/48.70
-4.58/-56.24/48.64
-4.54/-56.19/48.57
-4.47/-56.14/48.50
```

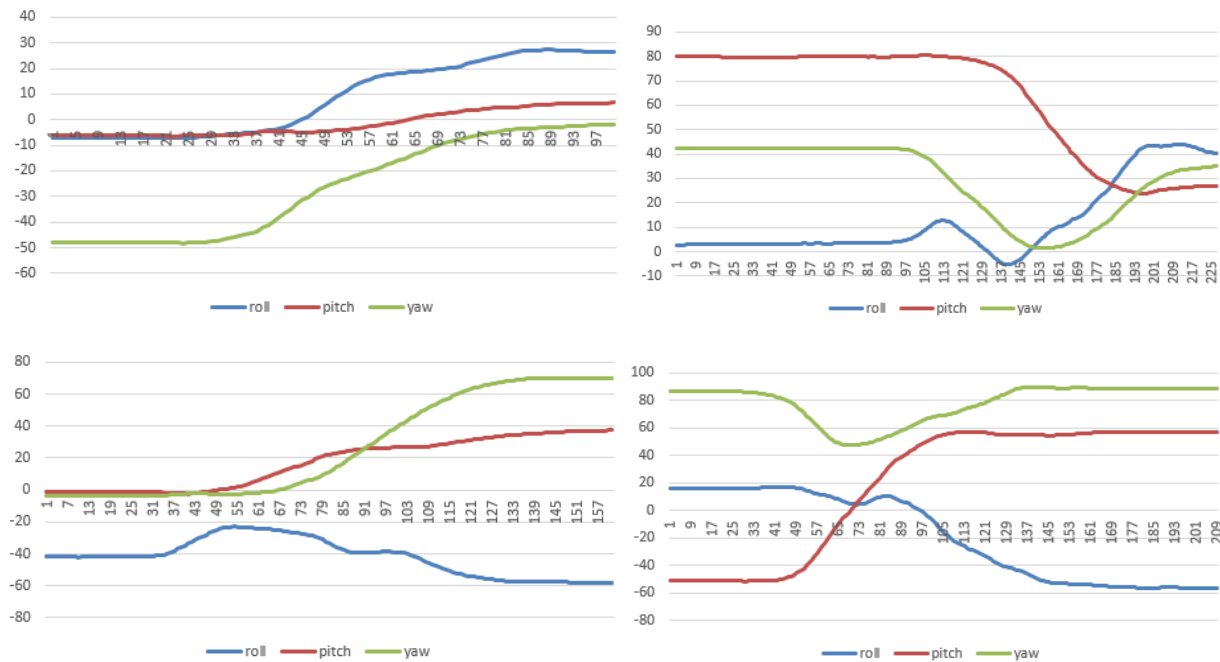
The average number of lines for each move is around 200 lines. And takes between 3 and 5 seconds to make.

3.2. Gathered Dataset:

We have gathered 2 types of moves: on and off. We gathered this data from the serial monitor as explained before, and named them accordingly, each one and every single file represents one single move. We gathered 140 moves in total to create our dataset.

In order to observe the results we got more closely; we represented a few moves as graphs. We chose 2 ON moves and 2 OFF moves.

We can observe that the ON moves end with roll pitch and yaw values that are closer to each other, while the OFF moves finish with values further from each other. This allows us to visualize the numerical results and see the difference between the two.



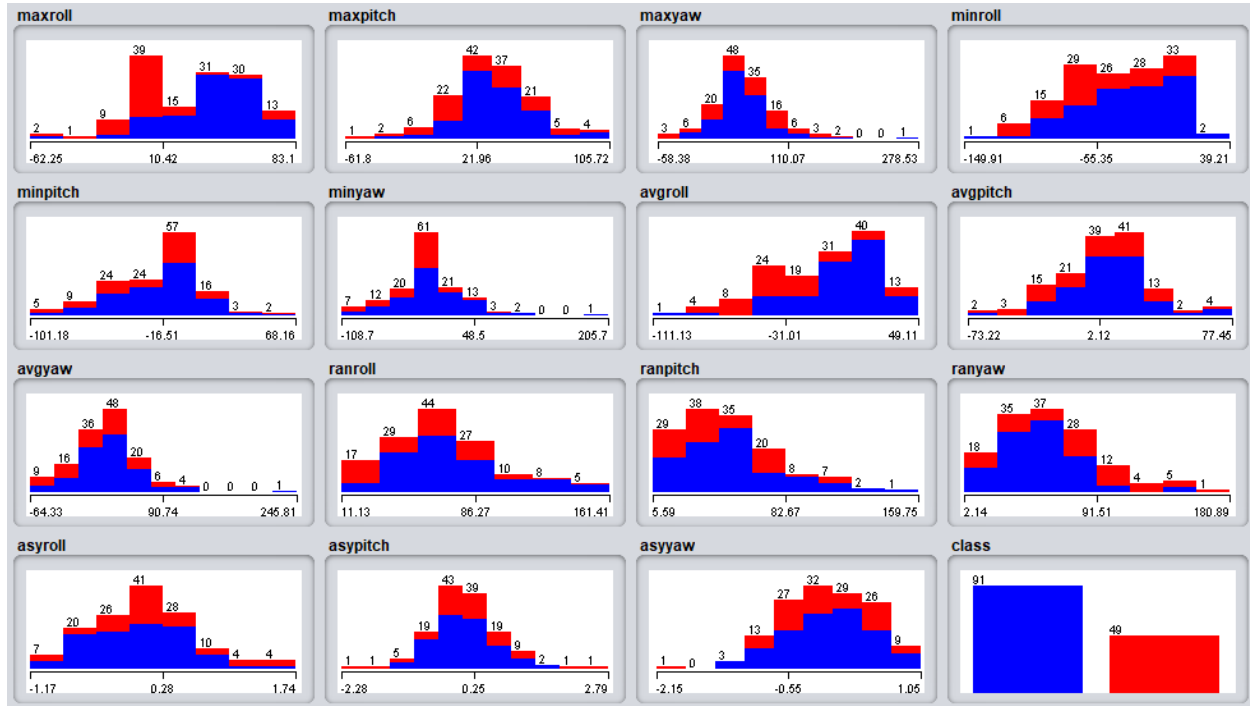
Top: ON, Bottom: OFF.

3.3. Preprocessed Dataset:

After getting enough moves and data, it is still in its raw state, so we must preprocess it before feeding it into the knn. It is at this stage that we are going to calculate the 15 values and save it all into a csv file.

The file in question will be the one we will be using in knn. It is a .csv file containing text, with its first line being the name of the column, each separated by a comma “,”. So there will be 16 columns, the 15 first ones are our calculated values, and the last one will represent its class, which is either “on”, or “off”.

And here are the statistics for the gathered and preprocessed dataset:



3.4. Result dataset in Weka:

After gathering a number of data from a range of different movements , we used a simple python script to translate the results obtained from the MPU and Arduino in a csv file readable by weka. The data we gathered consists of 140 movements in total, 91 right moves and 49 wrong moves.

maxroll	maxpitch	maxyaw	minroll	minpitch	minyaw	avgroll	avgpitch	avgyaw	ranroll	ranpitch	ranyaw	asyroll	asypitch	asyyaw
43,85	80,49	42,61	-5,39	23,93	1,33	12,31982	62,94295	29,66727	49,24	56,56	41,28	1,203729	-0,8	-0,74863
72,96	18,22	70,69	-0,63	-5,84	-2,04	25,03091	4,634355	26,04183	73,59	24,06	72,73	0,595637	0,525465	0,450947
17,27	55,41	41,88	-41,23	-66,6	-36,21	-9,92929	-2,04924	2,437727	58,5	122,01	78,09	-0,13586	0,124499	0,039518
68,65	23,47	75,21	-5,32	-3,91	-0,04	30,158	9,358486	38,37303	73,97	27,38	75,25	0,113072	0,00773	-0,10353
51,33	56,31	37,56	-44,67	-52,19	-32,02	5,681686	1,191744	-2,34802	96	108,5	69,58	-0,04563	0,089485	0,468553
32,65	23,08	50,89	-22,99	-43,95	-4,66	3,260245	-10,1234	17,43368	55,64	67,03	55,55	0,167501	-0,01285	0,468964
40,57	43,74	31,93	-40,15	-13,19	-21,88	-2,8624	8,614589	-2,68445	80,72	56,93	53,81	0,421293	0,561504	0,595499
20,82	17,72	72,59	-71,18	6,38	40,97	-24,809	14,34638	53,8683	92	11,34	31,62	-0,04111	-1,20032	0,531258
23,35	36,11	56,68	-25,76	-28,5	-25,63	0,455811	3,387027	15,82162	49,11	64,61	82,31	-0,0381	0,029049	-0,0195
7,11	46,07	83,53	-71,31	1,98	31	-42,7521	28,0212	66,03265	78,42	44,09	52,53	0,552662	-0,46749	-0,33172

Figure : Dataset details

After choosing the test option and K value the summary of the results we get will look like this:

```

=== Summary ===

Correctly Classified Instances      125          89.2857 %
Incorrectly Classified Instances    15          10.7143 %
Kappa statistic                    0.7517
Mean absolute error                0.128
Root mean squared error            0.2566
Relative absolute error            28.0844 %
Root relative squared error        53.8005 %
Total Number of Instances         140

=== Detailed Accuracy By Class ===

              TP Rate  FP Rate  Precision  Recall  F-Measure  MCC      ROC Area  PRC Area  Class
              0,978   0,265   0,873    0,978   0,922     0,764    0,975    0,980    on
              0,735   0,022   0,947    0,735   0,828     0,764    0,975    0,931    off
Weighted Avg.   0,893   0,180   0,899    0,893   0,889     0,764    0,975    0,963

=== Confusion Matrix ===

  a  b  <-- classified as
89  2  |  a = on
13 36  |  b = off

```

Figure : Detailed results.

Each result shown has its own meaning:

- The correctly and incorrectly classified instances show the percentage of test instances that were correctly and incorrectly classified.
- The Kappa statistic (or value) is a metric that compares an Observed Accuracy with an Expected Accuracy (random chance). It is used to evaluate classifiers. Not only can this kappa statistic show us how well the classifier performed, but we can also compare it to the kappa statistic of any other model used for the same classification. A value greater than 0 means that your classifier is doing better than chance[9].
- Errors : we calculate the error rate using $1 - (TP + TN) / N$, N being the number of instances classified.
- TP Rate: rate of true positives (instances correctly classified as a given class)
- FP Rate: rate of false positives (instances falsely classified as a given class)
- Precision: proportion of instances that are truly of a class divided by the total instances classified as that class ($TP / (TP + FN)$), while the general precision of our classifier will be $(TP + TN) / N$, N being the total number of instances classified.
- Recall: is a proportion of instances classified as a given class divided by the actual total in that class (equivalent to TP rate), $Recall = TP / (TP + FN)$.
- F-Measure: A combined measure for precision and recall calculated as $2 * Precision * Recall / (Precision + Recall)$.
- ROC area: The purpose of ROC (Receiver Operating Characteristic) Curves is to examine the performance of a binary classifier, by creating a graph of the True Positives vs. False Positives for every classification threshold. It shows us a better description of the accuracy of our model. An optimal classifier will have ROC area values approaching 1, with 0.5 being comparable to "random guessing" (similar to a Kappa statistic of 0)[10].

- PRC area: the precision recall curve, similar to the ROC , is another way to tell the efficiency of our classifier; it's a graph with Precision values on the y-axis and Recall values on the x-axis. The greater the area under the curve , the better our algorithm is performing[11].

For these results, we will mainly focus on the accuracy and the confusion matrices. We used different test options with different K values to obtain the following results:

Test options	Classification Accuracy		
	k=1	k=3	k=5
Use training set	100%	89.28%	87.14%
Cross-Validation	82.14%	84.28%	80.71%
Percentage Split (66%)	85.41%	85.41%	87.5%

1. For K=1 we got the following results:

- Use training set:

Real Class	Predicted Class	
	ON	OFF
ON	91	0
OFF	0	49

Precision : 100%

91 ON instances classified as ON (91/91).

49 OFF instances classified as OFF (49/49).

- Cross-Validation :

Real Class	Predicted Class	
	ON	OFF
ON	84	7
OFF	18	31

Precision : 82.1429%

84 ON instances classified as ON (84/91).

31 OFF instances classified as OFF (31/49).

7 ON instances classified as OFF (7/91)

18 OFF instances classified as ON (18/49).

- Percentage Split (48 instances to classify) :

Real Class	Predicted Class	
	ON	OFF
ON	28	3
OFF	4	13

Precision: 85.4167%

28 ON instances classified as ON (28/31)

13 OFF instances classified as OFF (13/17)

3 ON instances classified as OFF (3/28)

4 OFF instances classified as ON (4/17)

2. For K=3 we got the following results :

- Use training set:

Real Class	Predicted Class	
	ON	OFF
ON	89	2
OFF	13	36

Precision : 89.2857%

89 ON instances classified as ON (89/91).

36 OFF instances classified as OFF (36/49).

2 ON instances classified as OFF (2/91)

13 OFF instances classified as ON (13/49)

- Cross-Validation :

Real Class	Predicted Class	
	ON	OFF
ON	85	6
OFF	16	33

Precision : 84.2857%

85 ON instances classified as ON (55/91).

33 OFF instances classified as OFF (33/49)

6 ON instances classified as OFF (6/91)

16 OFF instances classified as ON (16/49).

- Percentage Split (48 instances to classify) :

Real Class	Predicted Class	
	ON	OFF
ON	29	2
OFF	5	12

Precision: 85.4167%

29 ON instances classified as ON (29/31)

12 OFF instances classified as OFF (12/17)

2 ON instances classified as OFF (2/31)

5 OFF instances classified as ON (5/17)

3. For K=5 we got the following results :

- Use training set:

Real Class	Predicted Class	
	ON	OFF
ON	87	4
OFF	14	35

Precision : 87.1429%

87 ON instances classified as ON (87/91).

35 OFF instances classified as OFF (35/49).

4 ON instances classified as OFF (4/91).

14 OFF instances classified as ON (14/49).

- Cross-Validation :

Real Class	Predicted Class	
	ON	OFF
ON	84	7
OFF	20	29

Precision : 80.7143%

84 ON instances classified as ON (84/91).

29 OFF instances classified as OFF (29/49)

7 ON instances classified as OFF (7/91)

20 OFF instances classified as ON (20/49).

- Percentage Split (48 instances to classify) :

Real Class	Predicted Class	
	ON	OFF
ON	30	1
OFF	5	12

Precision: 87.5%

30 ON instances classified as ON (30/31)

12 OFF instances classified as OFF (12/17)

1 ON instances classified as OFF (1/31)

5 OFF instances classified as ON (5/17)

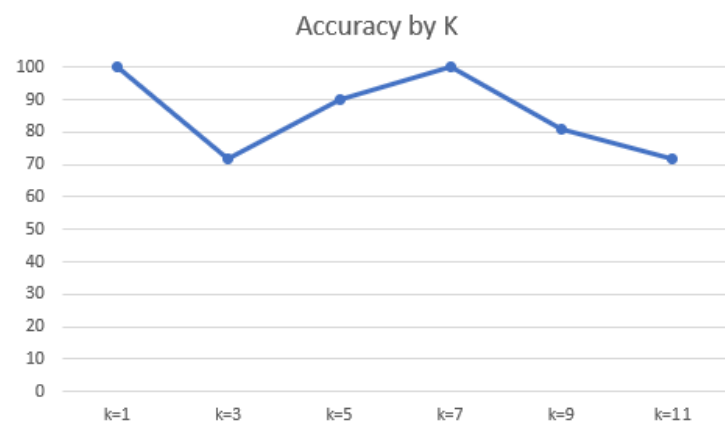
We notice that our model classifies the ON instances more accurately than the OFF instances, that is due to the higher number of the ON instances, which helped train on models to classify them better.

3.5. Classification with Python results:

Using our python script, we classified a total of 11 moves for different values of K, 5 of those moves were right (movements that turn on the watch) and 6 of them were wrong (movement that do not turn on the watch). The training dataset we used for these results is the one that consists of 140 movements, which we mentioned previously.

Here's a table that summarizes the results we got with each K value:

K values	k=1	k=3	k=5	k=7	k=9	k=11
Accuracy %	100%	72%	90%	100%	81%	72%



With the results we got, the average accuracy of our system is 85%.

When it comes to the K value, according to our research the ideal K value is the squared root of the number of instances.[24] In our case, the recommended value of K would be 11, however, that value proved itself too unreliable and provided inaccurate classification results.

4. **Conclusion:**

Throughout this project, we have gone through multiple steps, from research to conception to getting final results. We believe that we are satisfied with the results we have gotten, as they lived up to our expectations.

We worked towards an AI application that can recognize more precisely a person's wrist motion, in the goal of creating a system of motion recognition that can adapt to a person's movement to decide whether or not turn the watch on/off to see time. We have used KNN for this project, where we gathered our own dataset using an Arduino and a MPU6050, preprocessed it, then started classification with the help of the Weka program, we were able to validate our dataset. We simulated multiple and different scenarios and arm movements; this diversity has helped the algorithm recognize more accurately movements.

Weka results were encouraging and promising, so we designed and coded our knn program using python, that is able to test and recognize our target wrist movement in real time.

The python script was focus around real time testing and classification using a time window between each move. With a dataset of 140 instances, the classification was incredibly fast which proves very reliable as we got an average accuracy of 85% going up to 100%. This means that an even larger dataset will give even better and more promising results without having to sacrifice the performances.

Our system can be improved upon, and one way to enhance the results would be to implement incremental learning into it. So, every time our system would make a mistake, we would add the movement into the dataset to further enhance our accuracy.

Nonetheless, wrist movement detection is a very interesting subject that can be used in a variety of fields such as medicine, sports, and of course, leisure as well. This approach to motion detection shows great potential to expand towards responding to the needs of countless other fields. In our project we have worked with two outcomes: ON or OFF, but we can make more classes to define a wider set of possible results.

This approach is also quite affordable, as we were able to make a working prototype that has proven quite efficient, as mentioned previously. All of this has been done using very small and poor resources, which proves that with more budget, resources, a larger training dataset and time, great things can be achieved.

Sources:

- [1] (Hand gesture recognition using machine learning algorithms, November 2020, Computer Science and Information Technologies, Abhishek B. & Al)
https://www.researchgate.net/publication/345142103_Hand_gesture_recognition_using_machine_learning_algorithms
- [2] (IBM Cloud Education,in What is Machine Learning?, 15 July 2020)
<https://www.ibm.com/cloud/learn/machine-learning>
- [3] (IBM Cloud Education,in What is Supervised Learning? 19 August 2020) ,<https://www.ibm.com/cloud/learn/supervised-learning>
- [4] (Aidan Wilson, in A brief Introduction to Supervised Learning, 29 September 2019) <https://towardsdatascience.com/a-brief-introduction-to-supervised-learning-54a3e3932590>
- [5] & (Figure 1) (Ayush Pant, in Workflow of a Machine Learning Project, 11 january 2019) <https://towardsdatascience.com/workflow-of-a-machine-learning-project-ec1dba419b94>
- [6] (Mukund Billa, in Testers Guide For Testing Machine Learning Models, 12 October 2019) <https://medium.com/analytics-vidhya/testers-guide-for-testing-machine-learning-models-e7e5cea81264>
- [7] & (Figure 2) (Onel Harrison Sep 10, 2018):
<https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761>
- [8] (unknown):
https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with_python_knn_algorithm_finding_nearest_neighbors.htm
- (Figure 3): <https://www.generationrobots.com/fr/401504-module-gyroscope-l3q4200d.html>
- (Figure 4): <https://www.circuitspecialists.com/ACCELEROMETER.html>
- [9] (Dr. dale E.Parson)
[http://faculty.kutztown.edu/parson/fall2019/Fall2019Kappa.html#:~:text=%E2%80%9CThe%20Kappa%20statistic%20\(or%20value,to%20evaluate%20classifiers%20amongst%20themselves.](http://faculty.kutztown.edu/parson/fall2019/Fall2019Kappa.html#:~:text=%E2%80%9CThe%20Kappa%20statistic%20(or%20value,to%20evaluate%20classifiers%20amongst%20themselves.)
- [10] (experienceweka in ROC and AUC,May 24, 2016)
<https://experienceweka.wordpress.com/2016/05/24/roc-auc/>
- [11] (Precision recall curve, 19 July 2019)
<https://www.geeksforgeeks.org/precision-recall-curve-ml/>
- [12] (jason Brownlee in What is a Confusion Matrix in Machine Learning, 18 November 2016) <https://machinelearningmastery.com/confusion-matrix-machine-learning/>
- [13] (unknown)
<https://www.gsmarena.com/glossary.php3?term=sensors#:~:text=Accelerometers%20in%20mobile%20phones%20are,by%20tracking%20rotation>

[%20or%20twist.&text=Accelerometers%20are%20also%20used%20to,a%20vendors%20%27health%27%20application.](#)

- [14] (Ryan Goodrich, in Accelerometer vs Gyroscope: What is The Difference?, 31 May 2018) <https://www.livescience.com/40103-accelerometer-vs-gyroscope.html>
- [15] (unknown) <https://learn.sparkfun.com/tutorials/gyroscope/all>
- [16] & (Figure 6) (Yujie Dong, in Tracking Wrist Motion To Detect and Measure Intake Of Free-Living Humans, May 2012) <https://cecas.clemson.edu/~ahoover/theses/dong-diss.pdf>
- [17] (Adrian Fernandez, Dung Dang, in Getting Started with the MSP430 Launchpad, 2013) <https://www.sciencedirect.com/topics/engineering/accelerometer-sensor>
- [18] & (Figure 5): (unknown) <https://www.siliconsensing.com/technology/mems-accelerometers/#:~:text=All%20accelerometers%20work%20on%20the,corresponds%20to%20the%20applied%20acceleration.>
- [19](Danny Jost | Jul 11, 2019 4:00pm) <https://www.fiercееlectronics.com/sensors/what-accelerometer>
- [20] & (Figure 7) (MPU6050 - Accelerometer and Gyroscope Module, 17 March 2018) <https://components101.com/sensors/mpu6050-module#:~:text=The%20MPU6050%20is%20a%20Micro,of%20a%20system%20or%20object.>
- [21]& (MPU-6000/MPU-6050 Product Specification, Document Number: PS-MPU-6000A-00, Revision: 3.4, Release Date: 08/19/2013) https://components101.com/sites/default/files/component_datasheet/MPU6050-DataSheet.pdf
- [22] <https://howtomechatronics.com/tutorials/arduino/arduino-and-mpu6050-accelerometer-and-gyroscope-tutorial/>
- [23] (Complementary Filter) <https://sites.google.com/site/myimuestimationexperience/filters/complementary-filter>
- [24] S. Zhang, X. Li, M. Zong, X. Zhu and R. Wang, "Efficient kNN Classification With Different Numbers of Nearest Neighbors," in IEEE Transactions on Neural Networks and Learning Systems, vol. 29, no. 5, pp. 1774-1785, May 2018, doi: 10.1109/TNNLS.2017.2673241.

Bibliography :

- 1- Alberto Artasanchez & Prateek Joshi - Artificial Intelligence with Python, Your complete guide to building intelligent apps using Python 3.x and TensorFlow 2-Packt Publishing (2020).
- 2- DÉVELOPPEMENT D'UN SYSTÈME DE CLASSIFICATION AUTOMATIQUE DE L'ACTIVITÉ PHYSIQUE, Tariq Abdessettar, Février 2018.
- 3- Hand gesture recognition using machine learning algorithms, November 2020, Computer Science and Information Technologies, Abhishek B. & Al.
- 4- S. Zhang, X. Li, M. Zong, X. Zhu and R. Wang, "Efficient kNN Classification With Different Numbers of Nearest Neighbors," in IEEE Transactions on Neural Networks and Learning Systems, vol. 29, no. 5, pp. 1774-1785, May 2018, doi: 10.1109/TNNLS.2017.2673241.