POLITECNICO
MILANO 1863

Department of Electronics, Information and Bioengineering

Hall sensor based magnetic tachometer

Project of:
Lorenzo Locatelli,
Matteo Mentasti,
Giorgia Monzio Compagnoni,
Alberto Pettenella,
Letizia Santini

Advisor: Prof. Aliverti Andrea
Co-advisors: Bernasconi Sara

Year 2022 - 2023

Contents

1| Introduction

1.1. Contex

Tachometer is a device used to measure rotational speed, expressed in revolutions per minute (RPM), that is a measure of frequency of rotation, so it represents the number of full revolution around a fixed axis in one minute. Different classifications of tachometers can be done based on theirs characteristic.

They can be defined as contact tachometers if the contact with the rotating object is needed to perform measurement, while they are defined non-contact if it's not necessary to have any device attached to the target. In general contact ones are more accurate.

Another classification distinguishes analog from digital tachometers [1]. Analog ones can't store information, neither perform computations with data; speed is converted into voltage and it's read by analog voltmeter. Digital devices have more advantages, in fact they have circuitry with processor, memory, sensor and LCD or LED redout; since they can store measured values, they can perform calculations, and they provide speed in RPM on the display.

According to their working principle, tachometers can be divided in different categories: [2]

- **Mechanical**: based on the principle for which centrifugal force on a rotating mass depends on the speed of rotation. The force can be used to compress or stretch a mechanical spring. It's used for application that don't need high resolution, and it covers the range of 20-20000 rpm. [3]
- **Optical**: rotation is measured through an infrared light laser, that is reflected by refractive substance on the rotating target. A led light source is placed in a fixed position, and a photodiode biased in reverse bias is placed in the same area in a way that every time the refractive part of the target is hit by laser light, the photodiode is hit by the refracted light, which biases it in forward bias. The passage of the refractive part of the object can be detected by measuring a voltage drop on a resistor placed in series with photodiode. [4]
- **Stroboscopic**: in order to measure the rotational speed of the target, a reference mark is made on it, and its angular speed has to be measured. The stroboscope tachometer consists of a neon gas discharge lamp that illuminates directly the reference mark. The frequency of the flashing light is adjusted until the reference mark on the rotating object appears stationary, so that the frequency of flashlight and the one of the rotation of the target become equal. Rotational speed is obtained by the frequency of flashing light in this condition. [5]
- **Magnetic pickup**: it consists of a permanent magnet and a coil in which it's inserted. The sensor must be placed close enough to the rotating wheel in order to detect a ferromagnetic piece passing in its magnetic field. Ferrous object passing through the magnetic field changes the shape of the field itself, and this induces voltage in the coil. The rate of change of the speed of the rotating member, the strength of the magnet and the size of the air gap will affects the output amplitude. The frequency of the output signal is directly proportional to the rotational speed of the member. [6]

1.2. Hall effect

Another possible way to perform the measurement of RPM is using a Hall effect sensor. This type of sensor is based on a phenomenon called Hall effect after the American scientist Edwin Herbert Hall (1855-1938) who discovered it in 1879 during his PhD.

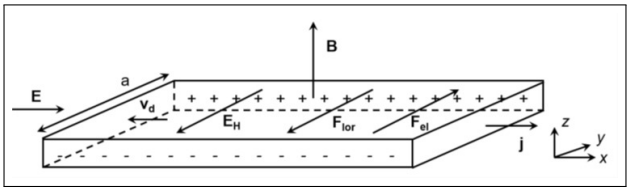

Figure 1.1: Experimental setup used to measure the Hall effect

A metallic plate in the shape of a parallelepiped is traversed by a current with density $\vec{J}$ (here directed along the x-axis) parallel

to the applied field $\overrightarrow{E}$. Since the carriers have a negative charge, the drift velocity $\overrightarrow{v_d}$ is opposite to $\overrightarrow{E}$. Subsequently, a uniform magnetic field $\overrightarrow{B}$ is turned on (for simplicity, here applied perpendicular to $\overrightarrow{E}$ and parallel to the z-axis), giving rise to a magnetic component of the Lorentz force:

$$\overrightarrow{F_{lor}} = -e \cdot \overrightarrow{v_d} \times \overrightarrow{B} \tag{1.1}$$

The Lorentz force (eq.1.1), opposite to the y-axis, is applied on the electrons. This force deflects the electrons from their average straight-line trajectory, creating an accumulation of electrons on one surface of the metal, leaving an equivalent excess of positive charge on the opposite surface. This charge distribution causes an electrostatic field $\overrightarrow{E_H}$ (called "Hall field"), transverse to the current flow and opposite to the y-axis, exerting a force:

$$\overrightarrow{F_{el}} = -e \cdot \overrightarrow{E} \tag{1.2}$$

Also the electrostatic force (eq.1.2), acts on the electrons on the y-axis, and it is opposite to the Lorentz force (eq. 1.1).

While $\overrightarrow{F_{lor}}$ stabilizes as soon as $\overrightarrow{B}$ reaches the desired value, the absolute value of $\overrightarrow{F_{el}}$ exponentially increases during the transient until it reaches a steady-state value where:

$$\overrightarrow{F_{el}} = \overrightarrow{F_{lor}} \tag{1.3}$$

The equilibrium condition of the forces (eq. 1.3) causes the electrons, subjected to zero force in the y-direction, to return to an average straight-line motion. At steady state:

$$E_H = v_d \cdot B \tag{1.4}$$

As long as the magnetic field $\overrightarrow{B}$ remains on, a transverse potential difference is generated:

$$\Delta V_H = E_H \cdot a \tag{1.5}$$

The potential difference shown in 1.5 can be measured, where a is the dimension of the parallelepiped in the y-direction. In conclusion a Hall effect sensor measures this potential difference ΔV_H proportional to the Hall field value E_H and detects changes in the applied magnetic field $\overrightarrow{B}$.

1.3. Aim of the project

Tachometers are used in different fields to measure and monitor rotating speed of engines; e.g., it can be used to detected if the speed exceeds a certain value, or to assess the performance of the engine. This project's aim is to realize a tachometer with a hall effect sensor.

This sensor is sensitive to magnetic field, so by placing a magnet on a rotating wheel and fixing the sensor close to the wheel, the sensor detects the passage of the magnet. The number of rotations can be obtained by counting the number of passages detected or by computing the periods between them.

In this project, this device has been realised in order to be applied to a bicycle and to measure its speed from the rpm of the back wheel. It could be applied also in rehabilitation, as it's able to record speed, so it's useful to monitor the performance of a subject; moreover, it could be integrated on a cycle-ergometer, a device consisting in a stationary bike used to perform tests of physical capabilities of subject as well as to evaluate their health condition.

2| Hardware

2.1. Components

- Arduino Micro; Hall Sensor; Neodymium Magnet (N35).
- LM358AN OpAmp; CA9-MT-V10-10K Potentiometer; Capacitance
- HC-05 Bluetooth module
- Power Bank solar charger 5.5V, 300-500mA; Booster DC-DC Step-up 2/24V to 5/12/28V Micro USB; Rocker switch ON/OFF SPST

2.2. Schematic of hardware connection

The circuit in the fig.2.1 has been used to ascertain that the output of the Hall sensor was analogical; moreover, a capacitance has been added to further stabilize the voltage supply from the Arduino.

Given the sensor's analog response to variations in the magnetic field, a possible approach to implement a tachometer would require to identify a voltage threshold, whose crossing, caused by the proximity of the magnet, signals the passage of the magnet in front of the sensor, and so a unitary increase of the revolutions' counter. Within this framework, the choice was to manually set the threshold value via a potentiometer, whose output is connected to the negative pin of an OpAmp used as a comparator. Being the output of the Hall-effect sensor connected to the positive input, a positive pulse is sent to the Arduino each time the threshold is overcome, while the output voltage has digital state LOW otherwise. The advantage provided by the potentiometer is the possibility to vary the distance at which the sensor is capable of detecting the magnet, without the need to modify any firmware variables. Indeed, lower threshold values entail that the sensor is sensitive also at higher distances, while higher threshold values require the magnet to be close to the sensor.

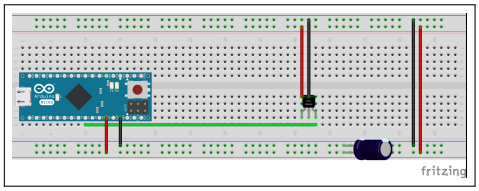

Figure 2.1: Circuit for the analogue reading of the sensor

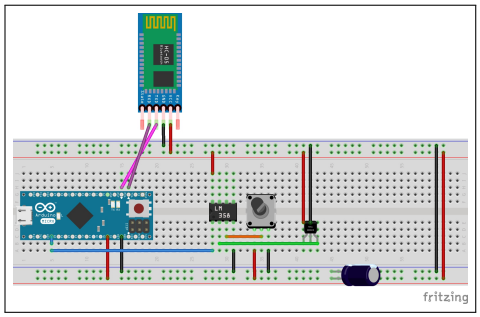

Figure 2.2: Circuit for the digital reading of the sensor and Bluetooth transmission of the data.

In order to transmit the data to a smartphone, the Bluetooth module HC-05 was connected to the RX, TX pins of the Arduino. A requirement for a device to be portable is to provide a mobile power supply. In this case, a solar charger with 5.5V output has been connected to a Boost voltage regulator to provide a stable input to the Arduino within the 7-12V range, as suggested by the manufacturer. A switch is interposed between the two elements, to turn on and off the device from outside the case.

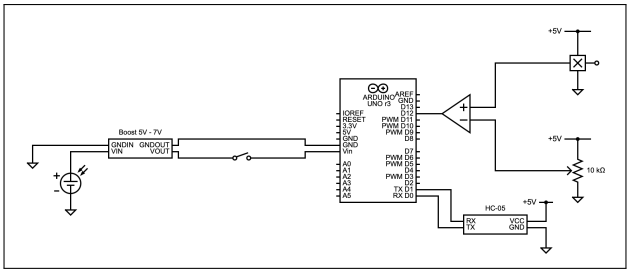

Figure 2.3: Digital Bluetooth configuration of the circuit.

2.3. Sensor conditioning

2.3.1. Linear measurements

To verify the distance at which the magnet is read better by the sensor, two magnets were tested. Firstly, it was chosen the family of the magnet; in particular, in this case it was used the N35. Then, two different magnets were tested: one with a uniform surface (*Cylinder magnet*), the other with a hole in the middle (*Hole magnet*).

As it can be seen in the fig.2.4, fig.2.5, the response is similar in both of them; an uniform surface could have been better to address properly the magnetic field, but the hole is more suitable for the application on the bike. In both cases, the response of the magnet was tested three times, varying linearly the distance. The magnet has been placed with 0° inclination with respect to the sensor, and the distances tested were in the range of [0mm ; 60mm] with resolution of 3mm.

Moreover, the magnet with the hole was properly positioned in order not to have the hole itself aligned with the sensor; this allowed to have a more uniform response, even though the lack of material.

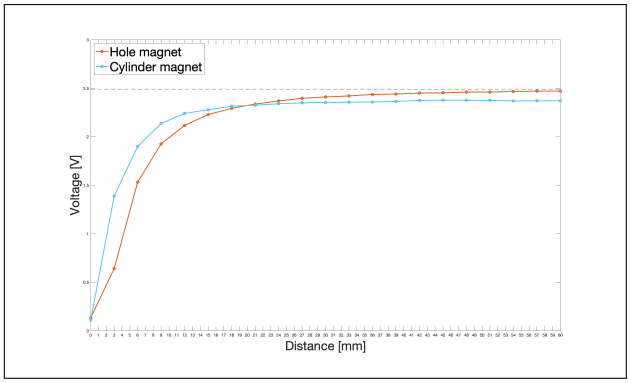

Figure 2.4: Comparison made with two different magnets, varying the distance, considering the north pole.

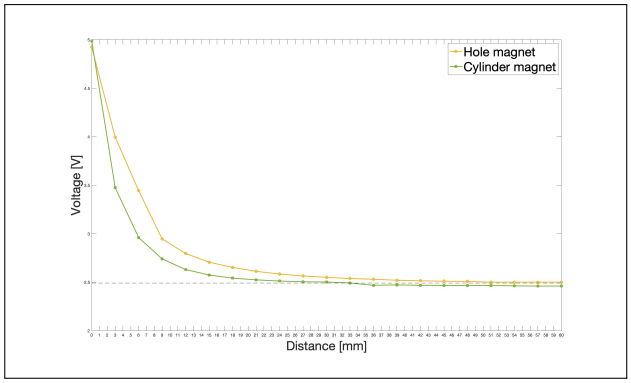

Figure 2.5: Comparison made with two different magnets, varying the distance, considering the south pole.

As reported in the fig.2.4, moving the north pole closer to the sensor the output signal decreases, while it returns to the baseline value (2,49V) with increasing distance of the magnet. The opposite behaviour is registered for south pole (fig.2.5), where the maximum output is obtained with minimum displacement from the sensor. Those magnets are read with distance up to 60mm, where the output stabilizes at offset value. In the following table 2.1, it is represent the values varying the distance, for respectively north **N** and south pole **S**.

Distance	Cylinder magnet	Hole magnet	Distance	Cylinder magnet	Hole magnet	Distance	Cylinder magnet	Hole magnet
0	N 0.11 S 4.986	N 0.132 S 4.992	24	N 2.34 S 2.512	N 2.368 S 2.584	48	N 2.376 S 2.466	N 2.46 S 2.5
3	N 1.388 S 3.474	N 0.64 S 3.996	27	N 2.348 S 2.504	N 2.396 S 2.55	51	N 2.374 S 2.466	N 2.46 S 2.5
6	N 1.9 S 2.96	N 1.532 S 3.444	30	N 2.354 S 2.502	N 2.41 S 2.538	54	N 2.368 S 2.462	N 2.466 S 2.5
9	N 2.136 S 2.74	N 1.926 S 2.946	33	N 2.374 S 2.492	N 2.45 S 2.53	57	N 2.37 S 2.46	N 2.47 S 2.5
12	N 2.24 S 2.63	N 2.116 S 2.796	36	N 2.358 S 2.468	N 2.436 S 2.52	60	N 2.37 S 2.46	N 2.37 S 2.5
15	N 2.278 S 2.574	N 2.228 S 2.706	39	N 2.364 S 2.472	N 2.44 S 2.514	∞	N 2.47 S 2.47	N 2.49 S 2.49
18	N 2.314 S 2.542	N 2.292 S 2.652	42	N 2.374 S 2.468	N 2.45 S 2.51	-	-	-
21	N 2.324 S 2.524	N 2.336 S 2.612	45	N 2.376 S 2.466	N 2.452 S 2.508	-	-	-

Table 2.1: Linear measurements for the cylinder magnet and the hole magnet, for the both poles.

It has been seen that the measurement are similar, so it has been chosen the hole magnet, since it is the most suitable for this project application. Hence, the following analysis has been conducted only on the hole magnet.

2.3.2. Angular measurements

Another consideration that can be taken into account are the angular measurements, where the magnet was moved along a circular trajectory with radius 8.5cm; the output of the sensor was measured with resolution of 3° in the range of [-24°; +24°], since for greater angles the magnet was not sensed and the output of the sensor was stable at the offset voltage (2.49V).

This test has been performed by placing the magnet at different linear distances from the sensor, in particular [1.5cm; 2cm; 2.5cm]. Also in this case, both the north and the south pole have been evaluated, and the measurements have been repeated 3 times for every condition. In the graphs are represented the values obtained by averaging the repeated measurements in every setup in order to have a more robust outcome.

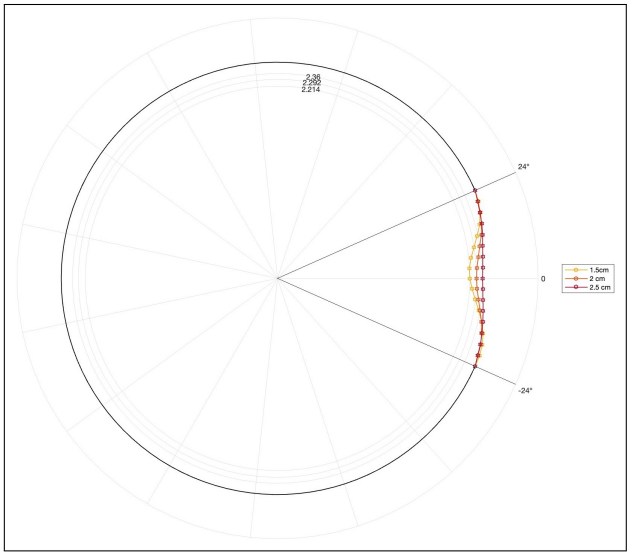

Figure 2.6: Plot of voltages of the magnet, varying the angle θ and maintaining the distance at three distances fixed - North pole.

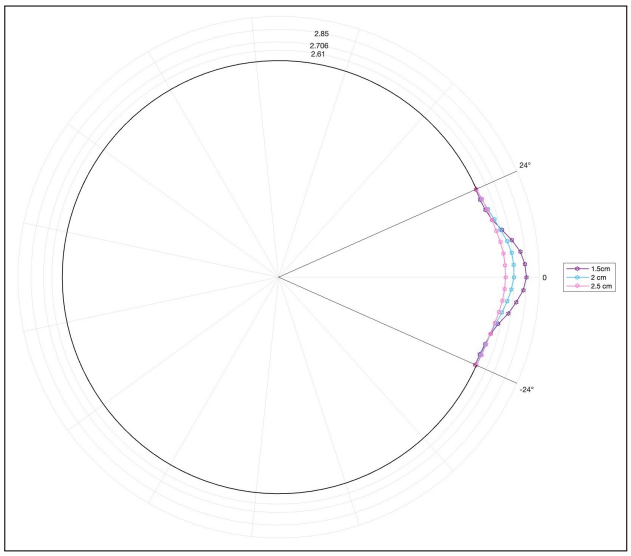

Figure 2.7: Plot of voltages of the magnet, varying the angle θ and maintaining the distance at three distances fixed - South pole.

θ	1.5cm	2cm	2.5cm	θ	1.5cm	2cm	2.5cm	θ	1.5cm	2cm	2.5cm
-24°	N 2.49 S 2.476	N 2.49 S 2.49	N 2.49 S 2.49	-6°	N 2.288 S 2.748	N 2.326 S 2.644	N 2.378 S 2.588	12°	N 2.348 S 2.624	N 2.398 S 2.61	N 2.414 S 2.56
-21°	N 2.49 S 2.476	N 2.5 S 2.472	N 2.49 S 2.468	-3°	N 2.242 S 2.818	N 2.3 S 2.68	N 2.366 S 2.602	15°	N 2.41 S 2.544	N 2.43 S 2.569	N 2.434 S 2.54
-18°	N 2.478 S 2.494	N 2.46 S 2.5	N 2.456 S 2.51	0°	N 2.214 S 2.85	N 2.292 S 2.706	N 2.36 S 2.61	18°	N 2.45 S 2.5	N 2.456 S 2.53	N 2.45 S 2.52
-15°	N 2.45 S 2.524	N 2.43 S 2.52	N 2.402 S 2.52	3°	N 2.212 S 2.836	N 2.298 S 2.708	N 2.366 S 2.608	21°	N 2.478 S 2.482	N 2.47 S 2.5	N 2.468 S 2.506
-12°	N 2.402 S 2.582	N 2.4 S 2.554	N 2.416 S 2.54	6°	N 2.24 S 2.796	N 2.324 S 2.694	N 2.376 S 2.6	24°	N 2.49 S 2.482	N 2.49 S 2.49	N 2.49 S 2.49
-9°	N 2.344 S 2.674	N 2.36 S 2.6	N 2.396 S 2.566	9°	N 2.29 S 2.716	N 2.356 S 2.66	N 2.39 S 2.582	-	-	-	-

Table 2.2: Angular measurements of the hole magnet, for both poles, varying θ at three set distances.

At the end of this phase, the south pole has been selected for the project. This choice is due to the fact that a slightly higher range of variation from the baseline has been recorded from the sensor at 2cm distance, which has been chosen based on the dimension of the bike.

2.4. Case

The case needs to contain: the breadboard with all the electronics, the Bluetooth module, the on/off button, a solar powerbank used to supply voltage and the booster. The space between the ray wheel and the seat stay was very limited, for this reason the case needed to be as compact as possible to fit all the components inside without interfering with the wheel. The design is such that the distance between the sensor and the magnet is $2cm$.

2.4.1. Magnet shell

For the magnet case, a shell is used to protect the magnet and to attach it to the ray of the back wheel. Since the chosen magnet is a disk with a through hole, it was used this characteristic to fix the magnet on the support with a nylon bolt and a nut.

The ray of the wheel is meant to pass in a groove of the case, and it was fixed between the case and another plastic plate in fig.2.8. In this way, it's possible to squeeze the ray between the structures, to better stabilize the magnet and the case on it, hence avoid undesired rotations of the magnet around the ray.

2.4.2. Sensor case

The case (fig. 2.10) consists in a box with a rectangular base of $155 \times 70mm$, and a height of $61mm$. All the walls have a $4mm$ width to have a stable structure.

One of the walls on the long side can slide along the top and bottom guides. This movable panel has a window covered with a small plexiglass layer, which protects the sensor and lets it perceive better the magnetic field. The opposite face has also a plexiglass covered window: it's used to expose the powerbank's solar panel to sunlight in order to recharge the powerbank while biking. Also one of the walls on the short side has an aperture, to accommodate the on/off button. On the floor of the box, eight holes of $5mm$ of diameter have been made, they are used to fix the case with its support and set both of them on the seat stay.

2.4.3. Support

A support (fig.2.9) has been designed to fix the case on the set stay of the bicycle. It consists in a U-shaped piece, sized in order to be blocked around the seat stay. On top of this structure there is a platform with eight holes that coincide with the ones of the case.

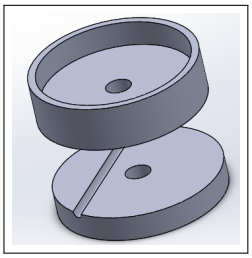

Figure 2.8: Magnet shell.

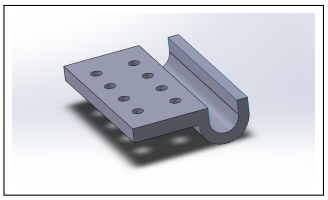

Figure 2.9: Support.

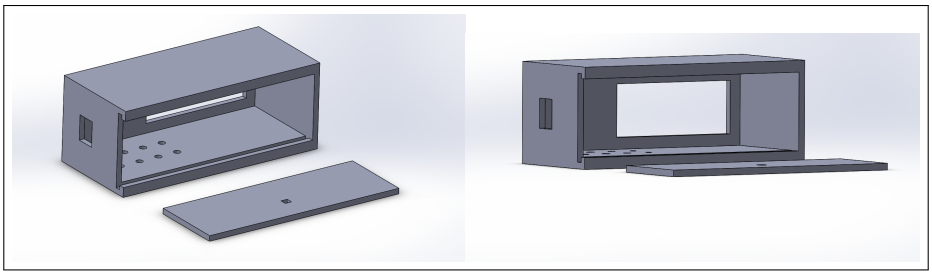

Figure 2.10: Sensor case.

3| Firmware

3.1. Algorithm

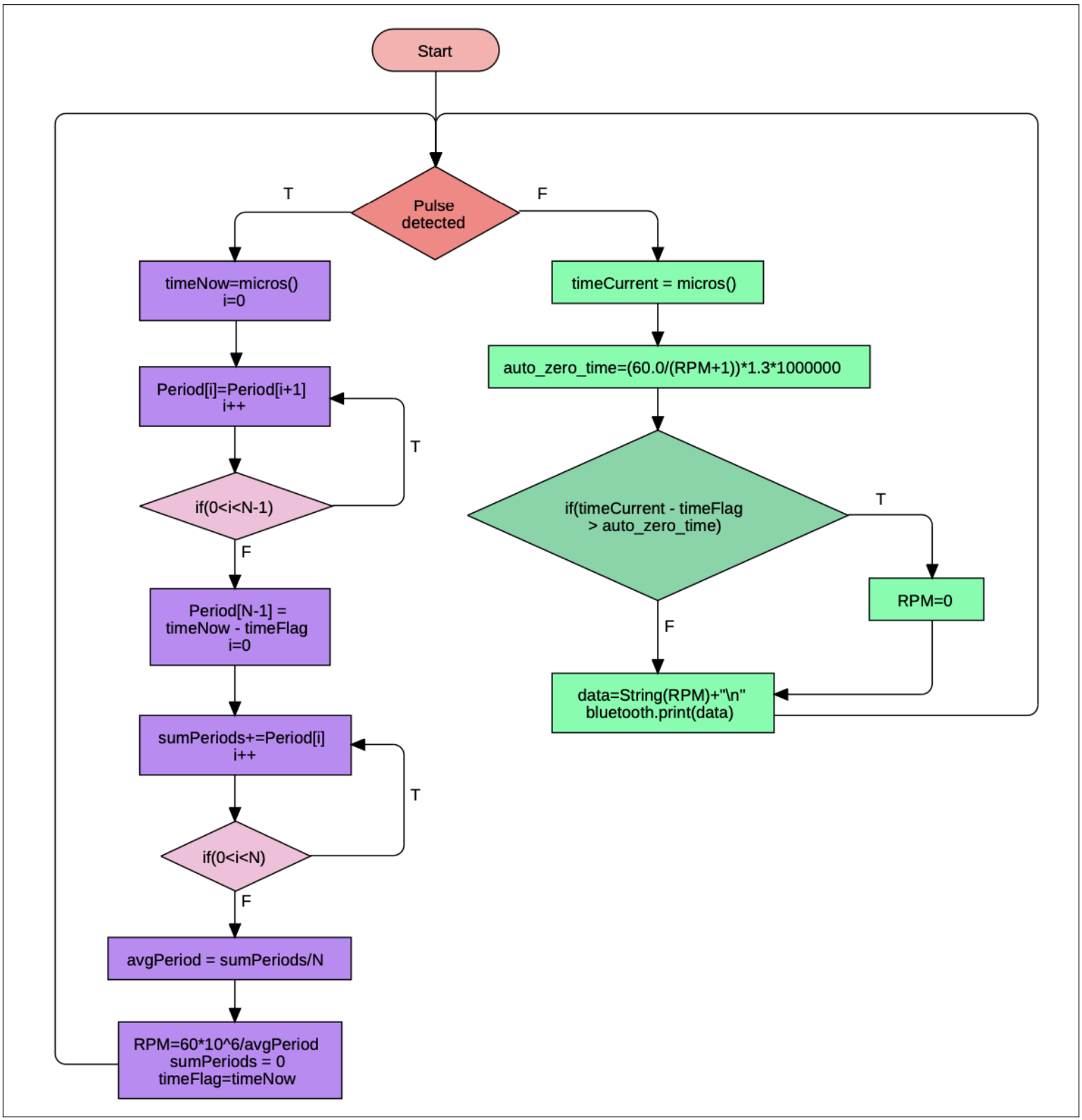

Figure 3.1: Flowchart that represents the firmware that has been used into this project to realize the tachometer.

The algorithm uses an interrupt on the rising edge of the digital signal arriving from the comparator, to compute the time between the last and the current pulses, which is the time measurement of the period of the last cycle.

The interrupt code implements a **moving average window** of size N. When a new pulse is detected, the values in the vector $Period[N]$ are shifted to the left of one position, and the newly computed period is saved in position N-1: the average of those periods is computed and converted in RPM.

The main code updates the current time at every iteration via the **micros()** function, and computes the time passed from the last measurement: if this time overcomes the auto zeroing time, then RPM is set to 0. Since our firmware updates the RPM value only when the magnet passes in front of the Hall sensor, so only when a pulse is detected, 0RPM have been set manually; otherwise, when the bike stops, the device would still show the last updated value of RPM. This auto zeroing time is computed with the formula:

$$\frac{60}{RPM+1} \cdot 1.3 \cdot 1000000 \tag{3.1}$$

The auto zeroing time (eq.3.1) is inversely proportional to the current RPM value, and was tuned via empiric measurements.

Then, at every iteration of the main code the RPM variable is passed as a string and followed by the newline character "\n" via Bluetooth to the laptop or to the smartphone. This formulation is needed to reconstruct the information with Processing after receiving it.

By trying with different values of N, the increasing of N makes the device slower in following the variations of the speed of the bicycle and doesn't help in stabilizing the measurement at high RPM rates, since they are already stable enough. So, in the final implementation N is set to 1, which means that instead of converting in RPM the mean value of the last N periods, we just convert the period of the last cycle.

4| Processing Application results

An application for Android devices has been designed using Processing software, to visualize the data collected from the sensor. This way, by having the application on their mobile phone, users can easily monitor various parameters during the activity. The app is named **"G.L.A.M. - Magnetic Tachometer"** with the first part that comes from the initials of the project participants' names, while the second part refers to the use of a magnetic sensor to create a tachometer.

4.1. Data collection

To enable the Arduino Micro firmware to send the data detected by the sensor to the application on the phone, a Bluetooth transmission has been implemented. The firmware utilizes the **"bluetooth.print()"** function of Arduino to send each data as a string consisting of an integer, representing the detected RPM followed by a newline character "\n": this is necessary because sending the detected data one after another without any separation, would make it impossible to distinguish and display them separately. Then, the code created with Processing (Fig.4.1) retrieves the original data by taking the string, removing the newline character and finally converting the data from a string to an integer for further use.

```
// transmission data by bluetooth
void onBluetoothDataEvent(String who, byte[] data)
{
  try
  {
    for (byte b : data)
    {
      if (b == '\n')
      {
        receivedString = stringBuilder.toString();
        // get RPM
        rpm = Integer.parseInt(receivedString.trim());
        // get KM/H
        kmh = rpm*2*PI*bikeRadius*0.06;
        // get M/S
        speedms = kmh/3.6;
        // get KM
        new_timestamp = millis();
        interval = new_timestamp - old_timestamp;
        old_timestamp = new_timestamp;
        km += speedms*interval/1000000;
        // save RPM MAX for LOG
        if((rpm > rpmMax) && (rpm <= 520)) rpmMax = rpm;
        // save KM/H MAX for LOG
        if((kmh > kmhMax) && (kmh <= 65)) kmhMax = kmh;
        // mean RPM and KMH
        if(rpm != 0)
        {
          sumrpm += rpm;
          countrpm ++;
          meanrpm = sumrpm/countrpm;
          sumkmh += kmh;
          countkmh ++;
          meankmh = sumkmh/countkmh;
        }
        //reset StringBuilder
        stringBuilder.setLength(0);
      }
      else
        stringBuilder.append((char) b);
    }
  }
  catch (NumberFormatException e)
  {
    rpm = 0;
  }
}
```

Figure 4.1: Fragment of the code in which it is defined the way through which data are sent from Arduino to the device.

4.2. Homepage

In the HOME section (fig.4.2), all the options offered by the application are listed: RPM detection (sect. 4.3), Speed expressed in km/h (sect. 4.4), Distance travelled (sect. 4.5), Destination to reach (sect. 4.6), Compass (sect. 4.7) and Data storage (sect. 4.8).

The various buttons have been implemented by creating a "Button" class (fig.4.3) in which parameters are defined to determine the position along the x and y axes, width, height and label of each button. Using the **"rect"** and **"smooth"** functions, a rectangle with rounded corners is created and the assigned name for that button is applied at the centre. Then, some conditions are imposed, by using Processing parameters **"mouseX"**, **"mouseY"**, and **"mousePressed"**, to identify the position of the user's touch on the phone, compare it with the button's position and manage the corresponding operation accordingly.

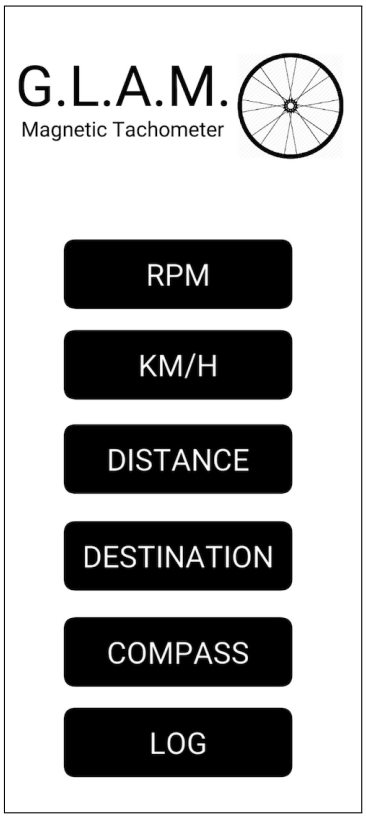

Figure 4.2: Home page of the app.

```
// Button class
class Button{
  float xpos, ypos, wid, hei;
  String label;
  boolean over = false, pressed = false;

  Button(float tx, float ty, float tw, float th, String tlabel)
  {
    xpos = tx;
    ypos = ty;
    wid = tw;
    hei = th;
    label = tlabel;
  }

  void update()
  {
    // draw rectangle
    smooth();
    fill(0);
    strokeWeight(4);
    rect(xpos, ypos, wid, hei, 30);
    // center text
    fill(255);
    textSize(80);
    text(label, xpos+wid/2-(textWidth(label)/2), ypos+hei/2+(textAscent()/2));
    // check if mouse pressed
    if(mouseX>xpos && mouseY>ypos && mouseX<xpos+wid && mouseY<ypos+hei)
    {
      over = true;
      if(mousePressed)
        pressed = true;
      else
        pressed = false;
    }
    else
      over = false;
  }
}
```

Figure 4.3: Fragment of the code in which we define the button class.

4.3. RPM

By pressing the first button, it is possible to visualize the RPM values detected by the sensor in both analogue and digital formats (Fig. 4.4). To draw the tachometer, a new "Arc" class (Fig. 4.5) has been defined consisting of parameters indicating the position along the x and y axes, width, height, start angle and end angle of the arc.

The range of possible values selected for the tachometer is 0-520 RPM, which is consistent with the application for which this project has been developed. In addition, specific sub-intervals have been chosen: 0-100 RPM is considered "low" speed and the arc is coloured green, 100-300 RPM at "medium" speed with an orange arc, finally, 300-520 RPM at "high" speed with a red arc. At the bottom of the screen, there is also an icon of a house that, when pressed, allows the user to return to the HOME page with all the available options to select.

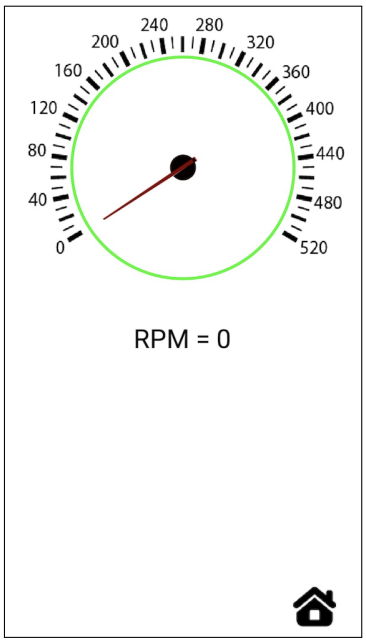

Figure 4.4: RPM visualized through the app.

```
// Arc class
class Arc{
  float xpos, ypos, wid, hei, angleOn, angleOff;

  Arc(float tx, float ty, float tw, float th, float ang1, float ang2)
  {
    xpos = tx;
    ypos = ty;
    wid = tw;
    hei = th;
    angleOn = ang1;
    angleOff = ang2;
  }

  void update()
  {
    // draw arc
    noFill();
    strokeWeight(8);
    arc(xpos, ypos, wid, hei, angleOn, angleOff);
    if(rpm <= 100)
      stroke(#03FF00);
    else if(rpm <= 300)
          stroke(#FF9100);
        else
          stroke(#FF0000);
  }
}
```

Figure 4.5: Fragment of the code in which it is define the arc class.

4.4. KM/H

The second button displays the same design as the first button with an analogue section represented by a speedometer (Fig.4.6) and a digital section that shows the value of the speed in km/h at which the bike is moving. To derive these values a conversion has been performed, based on the RPM values detected by the sensor, using the following formula:

$$km/h = RPM \cdot 2 \cdot \pi \cdot r \cdot 0.06 \tag{4.1}$$

In eq.4.4, r represents the length of the bike's radius. Consequently, a new range of displayable values ranging from 0 to 65 km/h is obtained. Once again, at the bottom of the screen, there is a button to return to the HOME page.

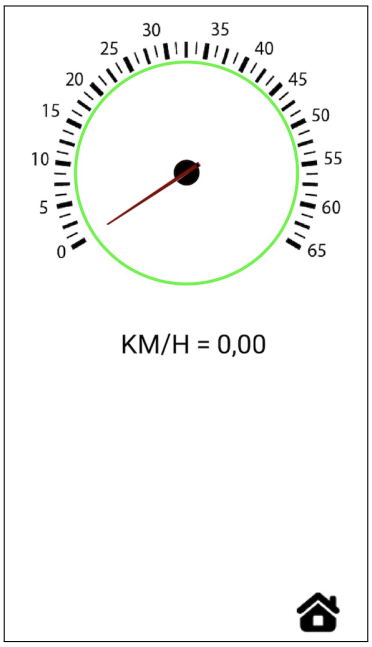

Figure 4.6: km/h visualized through the app.

4.5. Distance

The third button features a section that allows to display different parameters (Fig.4.7):

- **Speed in m/s**: obtained by dividing the km/h value by $3,6$.
- **Distance in km**: it tracks the distance covered by the bike. To obtain this data, the **"millis()"** function is used to retrieve the milliseconds elapsed since the start of the activity. Each time data is received, the timestamp is saved and compared with the previous one to calculate the time interval between the two acquisitions. By multiplying the time interval by the detected speed, the distance covered in that specific moment can be obtained. To determine the total distance travelled, a repeated sum of the calculated values is performed.
- **Activity duration**: it indicates the time elapsed since the application has been opened.
- **Clock**: it displays the current time.

In addition to the button HOME, this section also includes an additional RESET button at the bottom of the screen. This button allows the user to reset all the data collected in this section, namely the distance and the activity duration.

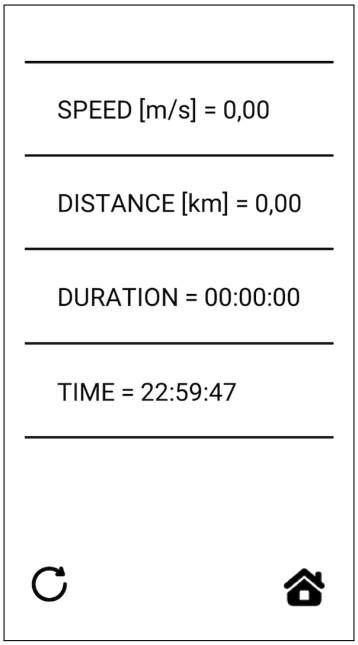

Figure 4.7: Distance and related values visualized through the app.

4.6. Destination

In this section, a useful tool has been included for the user to reach a preselect destination that in this case is the "Politecnico di Milano". By using the GPS coordinates of the phone, a pointer towards the set destination and the distance from the current location to it are displayed (Fig.4.8).

To make the pointer following the fixed location, the angle to rotate it is calculated by subtracting the device bearing toward the destination by using the **"location.getLocation().bearingTo()"** method from the device orientation toward north stored in "compass" by the **"onOrientationEvent()"** function. By using **"distanceTo()"** method the code compares the device's location via **"location.getLocation()"** with the fixed location and with the **"round()"** method it calculates the closest integer number to the floating point value returned by **"distanceTo()"** (Fig.4.9). At the bottom of the screen there is also a button to return to the HOME page.

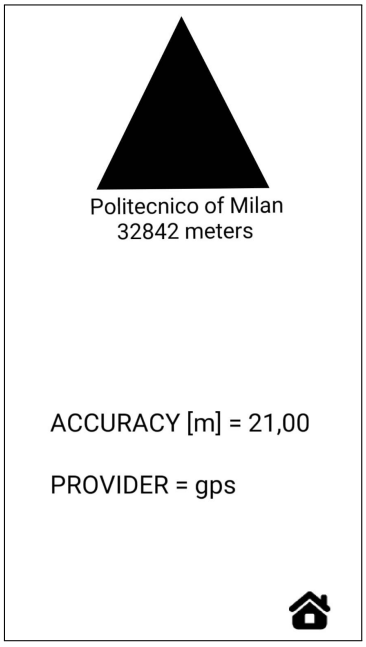

Figure 4.8: Destination visualized through the app.

```
// update triangle angle COMPASS
void speedoDESTINATION()
{
  pushMatrix();
  translate(width/2, height/3);
  // rotate the triangle toward the destination
  // the angle is calculated by subtracting the device bearing toward the destination from the device
  //orientation toward north stored in compass
  rotate(radians(bearingTo) - radians(compass));
  fill(0);
  triangle(-width/4, 0, width/4, 0, 0, -width/2);
  textSize(70);
  text("Politecnico of Milan", -width/3.66, height/31.26);
  text((int)distance + " meters", -width/5.14, height/15.63);
  popMatrix();
}

// display destination data
void drawDestinationData()
{
  if (location == null  || location.getProvider() == "none")
  {
    textSize(80);
    fill(0);
    text("LOCATION DATA IS \n     UNAVAILABLE", width/6, height/4);
    text("PLEASE CHECK YOUR \nLOCATION  SETTINGS", width/7, height/2);
  }
  else
  {
    // direction of destination pointer
    bearingTo = location.getLocation().bearingTo(destination);
    // distance betweeen actual position (device's location) and destination (Politecnico of Milan)
    distance = round(location.getLocation().distanceTo(destination));

    speedoDESTINATION();

    textSize(80);
    fill(0);
    text("ACCURACY [m] = " + nf(accuracy,0,2), width/9, height/1.56);
    // getProvider() returns "gps" if GPS is available otherwise "network" (cell network) or "passive" (WiFi MACID)
    text("PROVIDER = " + location.getProvider(), width/9, height/1.39);
  }
}
```

Figure 4.9: Fragment of the code in which it is defined the destination.

4.7. Compass

The fifth button features a compass (Fig.4.10) with some location information useful to the user such as: latitude, longitude, altitude, altitude accuracy and the cardinal point to which the phone is pointing.

The parameters listed above can be obtained through the **"onLocationEvent()"** function, while to display the cardinal point the previously described "compass" variable is used once again (Fig.4.11). It can take values from 0 to 359 degrees, where: 0° corresponds to North, 90° to East, 180° to South and 270° to West. By dividing the 360° arc into 8 equal parts, each of them can be assigned to one of the cardinal points: N, NE, E, SE, S, SW, W and NW.

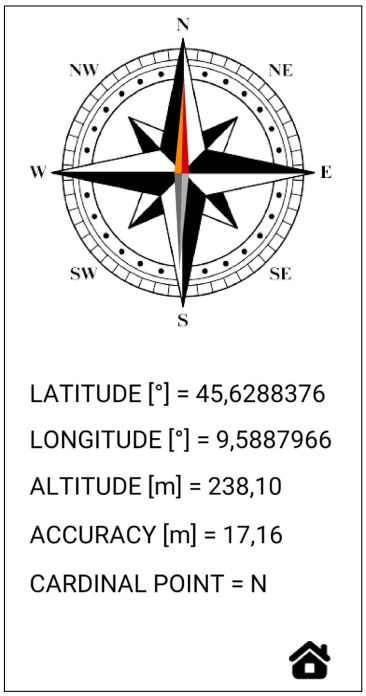

Figure 4.10: Compass tool included into the app.

```
// get location info
void onLocationEvent(Location loc)
{
  latitude = loc.getLatitude();
  longitude = loc.getLongitude();
  altitude = loc.getAltitude();
  accuracy = loc.getAccuracy();
}

// display compass data
void drawCompassData()
{
  if (location == null  || location.getProvider() == "none")
  {
    textSize(80);
    fill(0);
    text("LOCATION DATA IS \n     UNAVAILABLE", width/6, height/4);
    text("PLEASE CHECK YOUR \nLOCATION  SETTINGS", width/7, height/2);
  }
  else
  {
    drawCOMPASS();
    speedoCOMPASS();
    textSize(78);
    fill(0);
    text("LATITUDE [\u00B0] = " + nf((float)latitude,0,7), width/19.64, height/1.85);
    text("LONGITUDE [\u00B0] = " + nf((float)longitude,0,7), width/19.64, height/1.67);
    text("ALTITUDE [m] = " + nf((float)altitude,0,2), width/19.64, height/1.52);
    text("ACCURACY [m] = " + nf(accuracy,0,2), width/19.64, height/1.39);
   if((compass < 67.5) && (compass > 22.5))
         cardinalpoint = "NE";
      else if((compass <= 112.5) && (compass >= 67.5))
            cardinalpoint = "E";
         else if((compass < 157.5) && (compass > 112.5))
               cardinalpoint = "SE";
            else if((compass <= 202.5) && (compass >= 157.5))
                  cardinalpoint = "S";
               else if((compass < 247.5) && (compass > 202.5))
                     cardinalpoint = "SW";
                  else if((compass <= 292.5) && (compass >= 247.5))
                        cardinalpoint = "W";
                     else if((compass < 337.5) && (compass > 292.5))
                           cardinalpoint = "NW";
                        else
                           cardinalpoint = "N";
    text("CARDINAL POINT = " + cardinalpoint, width/19.64, height/1.28);
  }
```

Figure 4.11: Fragment of the code in which it is defined the compass.

4.8. Data storage

In this last section some user-relevant data has been collected (Fig.4.12), including: the maximum values reached for RPM and km/h, as well as the average values of RPM and km/h during the activity when RPM is different from 0. At the bottom of the screen, the RESET and HOME buttons are once again present.

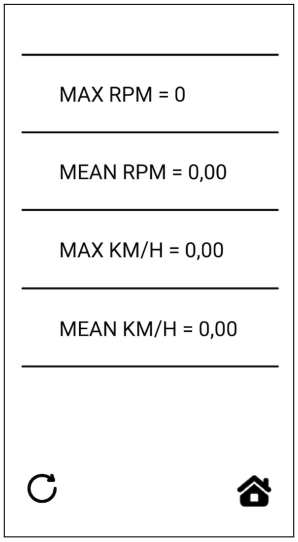

Figure 4.12: MAX/MIN visualized through the app.

5| Results

5.1. Calibration with stepper motor

In order to calibrate our device and to evaluate its accuracy we decided to create an experimental apparatus that we could use to simulate the rotation of the bike's wheel. We used a step motor **D42HS3418-24B22** and a stepper driver module **DRV8825** to control the rotation speed of a disk, on which we glued and balanced the chosen magnet. With a power supply of 12V, the stepper motor was able to reach the maximum speed of 520 RPM. We varied the speed of the disk from 10RPM to 520RPM and we measured it with our device, the result of those measurements are showed in the tab.5.1. The measurements have been repeated 5 times, each time the same values have been obtained. The weight of the balanced disk put in rotation is of 59g.

RPM set with the stepper	RPM read by the sensor	RPM set with the stepper	RPM read by the sensor	RPM set with the stepper	RPM read by the sensor	RPM set with the stepper	RPM read by the sensor
0	0	**140**	138	**280**	276	**420**	413
10	9	**150**	148	**290**	286	**430**	423
20	19	**160**	158	**300**	295	**440**	433
30	29	**170**	168	**310**	305	**450**	443
40	39	**180**	178	**320**	316	**460**	453
50	49	**190**	187	**330**	325	**470**	462
60	59	**200**	197	**340**	335	**480**	472
70	69	**210**	207	**350**	345	**490**	482
80	79	**220**	217	**360**	355	**500**	492
90	89	**230**	227	**370**	364	**510**	502
100	99	**240**	236	**380**	374	**520**	512
110	109	**250**	246	**390**	384	-	-
120	118	**260**	257	**400**	393	-	-
130	128	**270**	266	**410**	404	-	-

Table 5.1: Difference between the RPM set with a stepper motor and the RPM actually read by the sensor, using a moving window with N equal to 1 and the Bluetooth, with a boost converter.

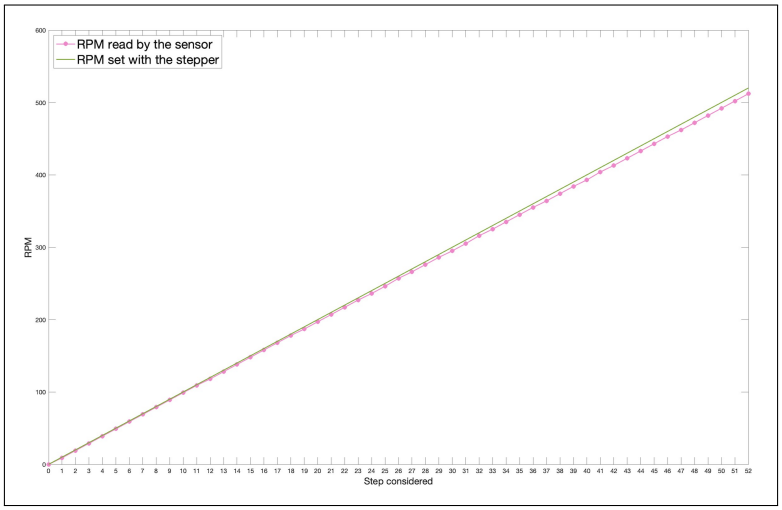

Figure 5.1: Difference between the RPM set by the stepper motor and the RPM read by the sensor, with respect to the step chosen. Mean inaccuracy: ±1.77%; max inaccuracy: ±10%; input range: 0 - 520 RPM; Full scale output: 0 - 512 RPM; resolution of the sensor: 1 RPM

5.2. Comparison with commerical device

As last step of the current project, a comparison between the developed device (*GLAM tachometer*) and a commercial bike tachometer based on a Hall effect sensor (*Cateye™ Velo 8*), has been made, installed both on the same bike wheel.

Several measurements have been taken during the simulation of 6 acceleration-decelerations sequences. Since the GLAM tachometer was updating the RPM value more frequently than the commercial one, measurements have been taken at every update of the latter (both devices were placed at the same position, so the values considered are the last corresponding update of both). The fig.5.2 shows that the measurements of the devices are similar both in the acceleration phase and in the deceleration phase; in particular, there have been trials in which the GLAM tachometer was measuring slightly higher values than the commercial one, and vice versa.

It is possible to see in the fig.5.3 that there is a correlation between the measurements of both devices. The maximum difference between them ($GLAM - commercial$) is 3.4 km/h (27RPM), while the mean difference is -1.1Km/h (-9 RPM).

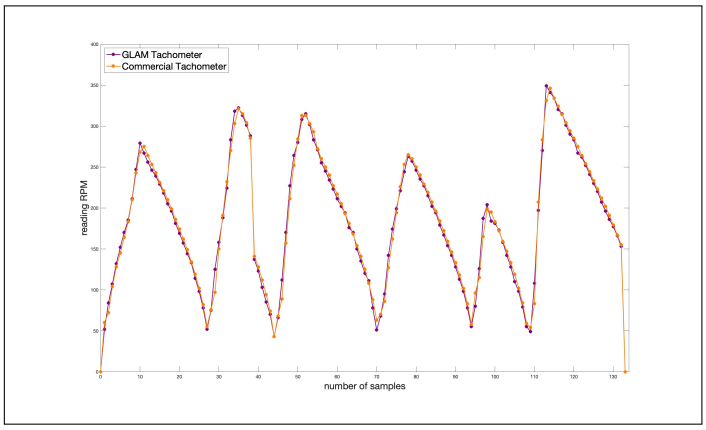

Figure 5.2: Comparison between GLAM tachometer commercial tachometer.

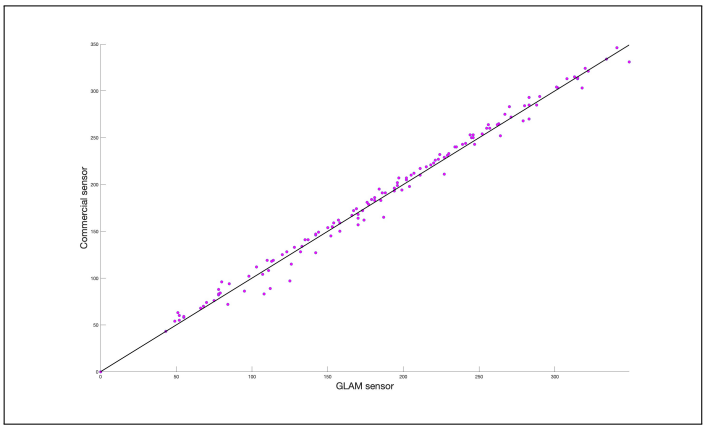

Figure 5.3: Scatter plot obtained comparing values of GLAM sensor and the commercial sensor.

5.3. Future development and current limitations

Since the current device prototype isn't sufficiently compact, future developments would include designing a PCB to make the device smaller and more portable on the bike.

Moreover, it would be possible to implement the storing of the information of past sessions, as well as some statistics and analytics based on them, that could be showed with graphs.

Also, by external insertion of data from the user via the application it would be possible to compute the calories expenditure, starting from the weight, sex and age of the biker. This feature could also be used to change the considered radius of the bike's wheel, to calibrate the device on different bikes.

Finally, the possibility to set a different destination from the "Politecnico di Milano" would make the destination feature more flexible.

Bibliography

[1] https://www.dubai-sensor.com/blog/tachometer-working-principle-types-applications/

[2] https://www.electronicsforu.com/tech-zone/test-measurement-electronics/rpm-measurement-sensors-techniques

[3] https://www.electronicsforu.com/tech-zone/test-measurement-electronics/rpm-measurement-sensors-techniques

[4] https://automationforum.co/tachometer-rotation-measurement-techniques-magnetic-optical-method/

[5] https://www.electricalvolt.com/2023/03/stroboscope-tachometer-and-its-working-principle/

[6] https://automationforum.co/tachometer-rotation-measurement-techniques-magnetic-optical-method/

List of Figures

List of Tables