Technologies for Sensors and Clinical Instrumentation

Group 10: Andrea Pezzillo | Serena Bassetto | Clelia Galluzzi | Mariangela Nappi |Francesca Fati | Giulia Camporeale

POLITECNICO MILANO 1863

Tool to follow a pathway with haptic feedback

Introduction

Our project is titled 'PoliPath' since its aim was to realize a tool able to follow a pathway driven by haptic command, using LDR sensors. In order to implement it we have used the microcontroller board *Arduino Micro*, for what concerns the realization of the hardware, the *Arduino Software* (IDE), for the firmware, and the software *Processing*, for what concerns the realization of the graphical interface with the user. In addition, *Photoshop* has been used to improve the quality of the graphics of the interfaces and to create the logo representative of the whole project.

The main idea consists in the development of a device, which intuitively works as a joystick and can be used to move a virtual object into a virtual environment. More in detail, we have created different virtual pathways that are shown to the user on a screen and which they have to 'follow' by moving a small virtual ball. To give the command for making the ball move, the user only needs to cover the proper sensor with a finger and, moreover, moving closer or further the finger with respect to the sensor, the user can regulate the velocity of the virtual ball.

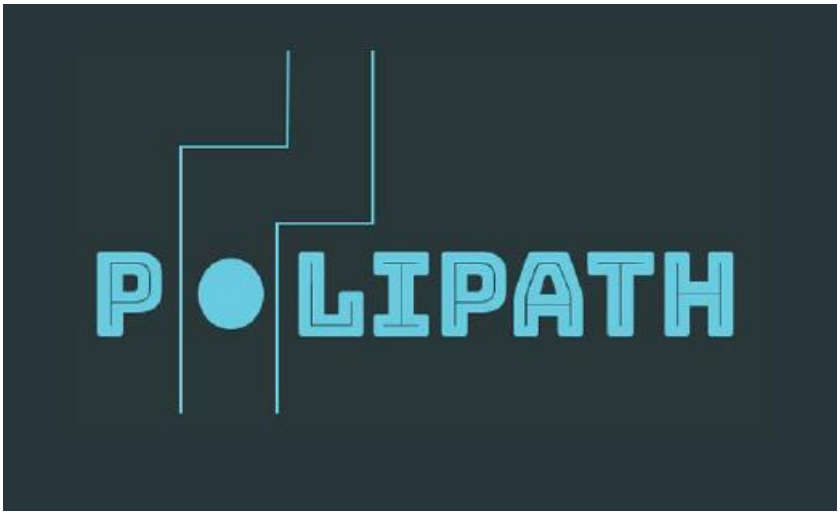
The device is composed of four different optical sensors (LDR) placed in such a way to indicate the four main directions - up, down, left and right. LDRs are sensors sensitive to the light to which they are exposed, and they change the values of their resistance according to the intensity of the light. Thus, it is possible to differently stimulate the sensors by simply covering them with a finger and therefore changing the intensity of light hitting them. The responses of the sensors, when covered, are the signals used to move the ball in the directions represented by the positions of the sensors themselves. Precisely, the different response of the sensors to different light and dark conditions, is not only used to give a binary command according which the ball moves or does not move, but is more deeply exploited allowing the user to vary the velocity of the ball by moving closer or further the finger from the sensor.

A possible application of this device, besides being an entertaining game, can be in the rehabilitation field for patients affected by stroke or with somatosensory impairments who suffer from alterations in the visuo-perceptive and visuo-motor skills. In particular, it can be used to assess the ability of the patient in applying a visually guided motor behaviour through the evaluation of the eye-hand

coordination and of the visuo-motor velocity, that is the velocity with which they are able to respond to a visual stimulus through a movement. The patient in fact has to command the virtual ball by touching one of the four sensors placed on the breadboard: since each of them allows the movement of the object in one of the main four directions, they should be able to move their fingers in a precise way in order to touch the correct one. Moreover, they also have to regulate the velocity of the object by moving the finger closer or further from the sensor. In this way it is possible to evaluate whether the patient is able to understand that, when approaching a variation of direction, they have to decrease the velocity in order to remain inside the borders of the pathway and whether they are able to connect this visual stimulus to the movement of the finger.

We have decided to implement three different levels of increasing difficulty to challenge and entertain more the user. The level of complexity is represented by a progressively narrower and more tortuous path, with a higher number of changes of direction.

Moreover, in order to have a quantitative measure, the time spent to complete each level is recorded and shown to the user, so that they can have feedback about their performance. This also should be an important clinical data for both patients and doctors and could be used to set goals which the patient has to reach during a rehabilitation program.



Our logo

Hardware

The hardware is composed at first by a solderless breadboard and the microcontroller board Arduino Micro. Together with these starting elements, the other components that we have used in order to realize the circuit are LDR sensors and resistances.

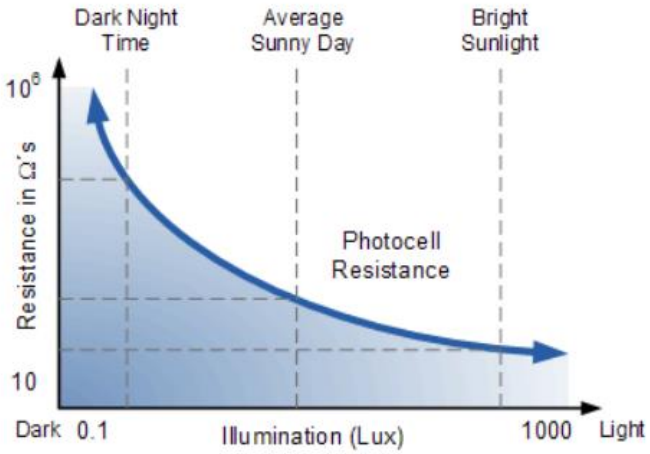
Light Dependent Resistors (LDR) are optical resistive sensors also called photoresistances. As the name suggests, they are variable resistances that work by changing their resistance depending on the light to which they are exposed. Depending on the material they are formed of, they work for wavelengths ranging from near UV (0.1 µm) to near IR (1 µm), thus the visible light is included in their working range (400-700 nm).

They are constituted by semiconductors, both intrinsic or doped: at ambient temperature these materials behave like insulators since the valence and conduction bands are not overlapped, as instead happens in conductors, but are divided by the energy gap whose value is specific for each material, but for semiconductors is about few eV. If some energy is provided in the form of heat or light some valence electrons can absorb an energy higher or equal to the one of the energy gap: in this way their covalent bond is broken and they can be promoted to the conduction band becoming 'free electrons'. In the valence band, each electron promoted leaves behind a hole, therefore there is the generation of electron-hole pairs: they increase the conductive properties of the material thus lower its resistance. This effect is even higher in doped semiconductors, in which the probability of having electrons in the conduction band is higher. The impurities added contribute to the conductive properties of the material by providing extra electrons immediately below the conduction band, that can be easily excited (donors), or extra holes immediately above the valence band, that can be easily filled by bound electrons (acceptors). Since the electrons do not have as far to jump, lower energy photons (which means longer wavelengths and lower frequencies) are sufficient to trigger the device.

This is the principle of the functioning of LDR: they are variable resistances able to respond to different levels of light by changing their electrical properties, a phenomenon called photoconductivity. Under dark conditions they show a very high resistance and therefore they can be considered as insulators. On the contrary, for increasing levels of light, they progressively decrease their resistances following a highly non-linear relationship:

$$R = A\,Ev^{-\alpha}$$

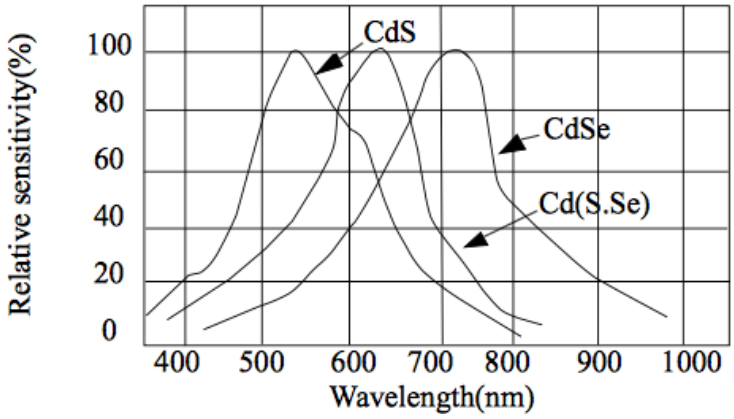
where:
R = resistance [Ohm]
Ev = illumination [Lux]
A, α are constants characteristic of the material
(0.7 < α < 0.9 for CdS)



Therefore, the greater the illumination, that is the incident power per unit of area, the higher the conductivity shown by the material. In the dark, a photoresistor can have a resistance as high as several megaohms (MΩ), while in the light, it can decrease up to a few hundred Ohms.

The resistance range and sensitivity of a photoresistor can substantially differ among different devices and also they may react substantially differently to photons within certain wavelength ranges.

The incident photon must have an energy at least equal to the energy gap and therefore a maximal wavelength equal to λ_{GAP} in order to generate the internal photonic effect. The spectral response of LDR shows a peak for $\lambda = \lambda_{GAP}$, and decreases toward zero for different wavelengths. For higher values of λ the energy of the photon is not enough to allow the overcoming of the energy gap and therefore, even if the electromagnetic wave can be absorbed by electrons, they immediately relax to the initial condition, releasing the energy absorbed. For lower values of λ another phenomenon has to be considered: the penetration depth of light is inversely related to frequency. Therefore, even before reaching values of energy so high that will cause the ionization of the material, these photons arrive only at a very superficial layer without penetrating the sample. Since the energy gap and so the peak of the response is material-specific, the type of LDR has to be chosen depending on the wavelength of the radiation that has to be detected in each particular application. For the visible range, that is the spectrum to which we are interested, Cd-based materials are typically used.



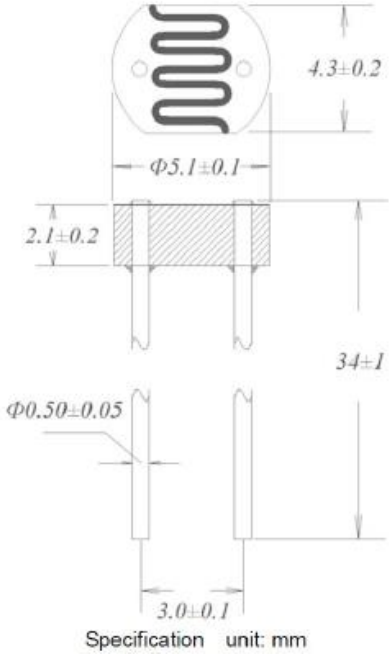
The structure of the considered sensor is very simple: the sensible element is a serpentine made of sensible material placed on a ceramic substrate and connected to two electrodes. In this way, when an external voltage is applied and a conditioning circuit with a fixed resistance is created, it is possible to measure the useful signal as the tension across the fixed resistance, which in turn depends on the value of the LDR variable resistance and so on the input light intensity.

Our sensor is a GL55 Series Photoresistor, precisely model GL5516.

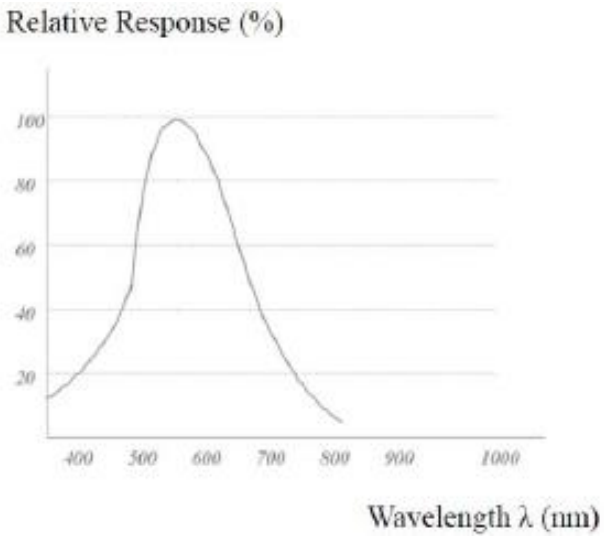
Tab. Specifications

Sensing Element	Cds
Max Voltage	150 V
Max Power	90 mV
Environmental Temperature	-30 ~ +70 °C
Spectrum Peak Value	540 nm
Light Resistance (10 Lux)	5-10 KΩ
Dark Resistance	0.5 MΩ
Response Time (increase)	30 ms
Response Time (decrease)	30 ms
γ = log(R10/R100)	0.5

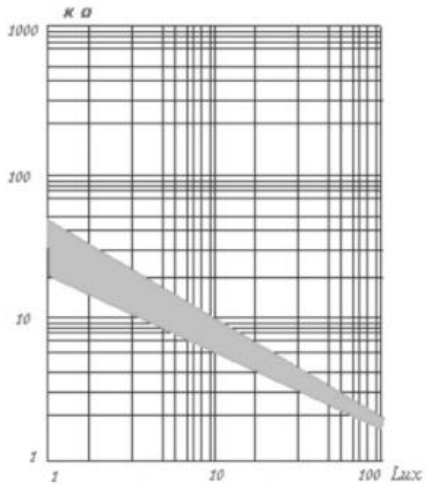
* *R10 resistance at 10 Lux, R100 resistance at 100 Lux*



Specification unit: mm

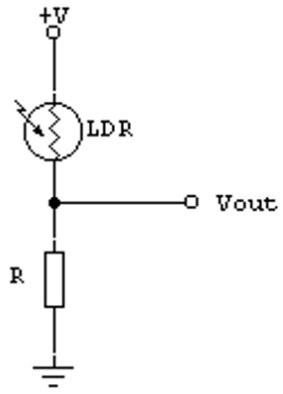


Spectrum Response Characteristic



Illuminance-Resistance Characteristic Curve

In our application we have used 4 LDRs that have been conditioned through a voltage divider circuit. Each photoresistor has been coupled with an appropriate resistance in order to control the tension across it and so avoid its rupture.



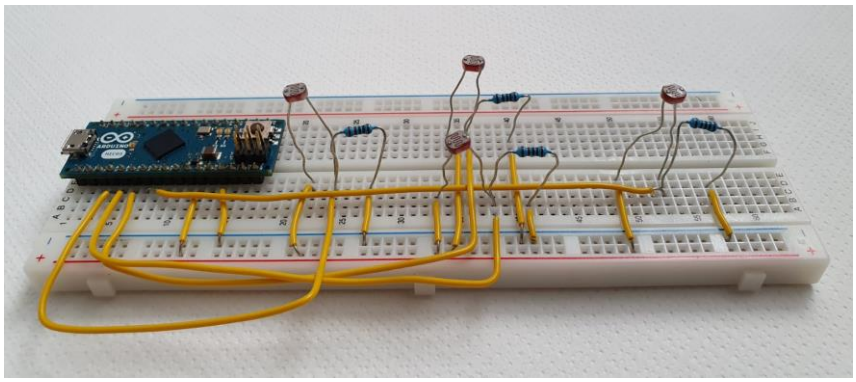
$$V_{out} = +V * [R / (LDR + R)]$$

where: +V is the voltage supply
LDR is the value of the photoresistance
R is the value of the conditioning resistance

The above-mentioned formula clearly explains the dependence of LDR resistance on light intensity: with decreasing levels of light, the conductivity of the sensor material decreases and therefore its resistance increases; as a consequence, the value of Vout decreases.

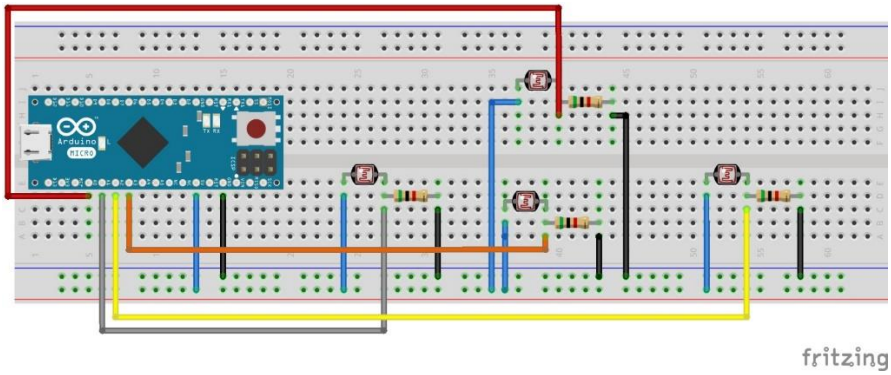
The value of the resistances we chose for conditioning the sensor is of 5 KΩ. For each sensor we have realized a simple circuit on the breadboard. One terminal of each LDR is connected to the power supply of 5 V of the Arduino. The other one is connected to an analog input pin of the Arduino, since the LDR gives out an analog voltage, and, in parallel, to a resistor. The other terminal of the resistor is grounded. The Arduino, with its built-in ADC (analog-to-digital converter), then converts the analog voltage (from 0-5V) into a digital value in the range of (0-1023).

The value of the conditioning resistance has been chosen in order to maximize the difference of the output voltage measured under light and dark conditions and therefore to exploit as much as possible the interval 0-1023 in which these values are converted. In particular, we have considered conditions of light ranging from 100 Lux, when the sensor is exposed, to 10 Lux, when the sensor is shadowed. From the characteristics of the LDR showed in its datasheet we have derived the value of resistances under these conditions of illuminance: 2 kΩ at 100 Lux and 10 kΩ at 10 Lux. By considering these values we have computed the output voltage: respectively they are 3.75 V with light and 1.67 V without light, resulting in ΔV=1.90 V. The same has been done with different values of the fixed resistances (1 KΩ, 2 KΩ, 10 KΩ), but 1.90 V was the maximum voltage difference achievable. The same circuit has been realized for the remaining 3 sensors in order to have the output from each of them on a different Arduino pin.



For the realization of all the connections we have used jumper rigid wires, so that the breadboard would appear more organized and especially none of the sensors would be partially shadowed by the wires. In addition, we have created a custom-made black case with four holes in correspondence of the LDRs, for hiding the breadboard and making the device more suitable for a possible user.

The final circuit realized through the software *Freezing* is shown below.



In our application, at the beginning all the sensors are exposed to the same light intensity thus having a very similar output response: this is the baseline condition. Then the resistance of each sensor increases when the light intensity hitting it decreases. Therefore it is expected to observe a significant reduction in the output voltage of the sensor of interest when it is covered with a finger so that lower or no light can reach it.

Arduino boards contain a multichannel, 10-bit analog to digital converter (2^10 possible levels). This means that an input voltage from 0 to the operating voltage (5V in our case) is mapped into integer values between 0 and 1023. Arduino has 5 built-in analog to digital converter channels therefore only these pins can be used to measure analog signals. If more channels have to be used it is necessary to interface an external ADC with Arduino. The resolution (magnitude of each voltage level or step) can be computed as:

$$Discrete\ Step = \frac{V_{ref}}{Total\ Steps} = \frac{5\ V}{1024} = 0.0048828 = 4.88\ mV$$

The *analogRead()* function is used to program and address analog pins on the Arduino board and it returns the analog input reading by converting it into an integer between 0 to 1023.
The function takes 100 microseconds to read an analog input, leading to a theoretical sampling rate of 9600 Hz (9600 samples each second). It is high enough to avoid the problem of having a delay between the moment in which the command is sent to the input pin and the moment in which the pin is read, which could cause errors in identifying the active sensor. For this reason it is not necessary to use a multiplexer.

We reach the conclusion that the amplifier is not needed in this specific situation. This is because the function *AnalogRead()* of the Arduino code maps input voltages between 0 and the operating voltage 5V into integer values between 0 and 1023, as we already mentioned. Under those circumstances and considering a 5 KΩ conditioning resistance, *AnalogRead()* returns integer values within the range of 0-600. Using a 2X amplifier, the signal is amplified and the values obtained are 0-1200. However, the maximum value that *AnalogRead()* returns is 1023. Thus, values greater than this threshold are returned as 1023 and the signal is always saturated. Moreover, the signal generated by our sensors is high enough and so *AnalogRead()* returns an optimal range of values. As a result, the usage of the amplifier has been unnecessary since it would have amplified too much the output values, losing therefore part of the signal.

Firmware

After having defined the variables needed, we have proceeded with the *setup*. In this part of the code the calibration of the sensors is implemented. This is done for the first 5 seconds after the turning on of the device. During these seconds, all the values of the signals generated by LDRs are read through the *AnalogRead()* function and recorded by Arduino. This is done because, even under fixed light conditions, the signal is not completely stable and the values oscillate within a small range. For this reason, the maximum and the minimum signal of each sensor are taken by the Arduino. Then, two variables are created in order to compare two by two the maximum voltage values. The first variable returns the maximum value of the two values analysed. The same procedure is applied to the second variable. Then, a third variable is created in order to compare the results just obtained and finally the absolute maximum value is associated with a fourth variable called *sensorMax*. The absolute minimum value is measured using the same method and it is given to the variable *sensorMin*. Thereafter, the average between the absolute maximum value and the absolute minimum value is calculated in order to set the optimal threshold. In particular, the threshold is computed as the 80% of the average response of the sensors acquired during the calibration period.

```
while (millis()<5000) {
   sensorValue1=analogRead(sensorPin1);
   sensorValue2=analogRead(sensorPin2);
   sensorValue3=analogRead(sensorPin3);
   sensorValue4=analogRead(sensorPin4);

   // record the maximum sensor value

 if (sensorValue1>sensorMax || sensorValue2>sensorMax || sensorValue3>sensorMax || sensorValue4>sensorMax ) {

     sensorX1 = max(sensorValue1, sensorValue2);
     sensorX2 = max(sensorValue3, sensorValue4);
     sensorX3 = max(sensorX1, sensorX2);
     sensorMax = sensorX3;
   }

   // record the minimum sensor value

 if (sensorValue1<sensorMin || sensorValue2<sensorMin || sensorValue3<sensorMin || sensorValue4<sensorMin) {

     sensorY1 = min(sensorValue1, sensorValue2);
     sensorY2 = min(sensorValue3, sensorValue4);
     sensorY3 = min(sensorY1, sensorY2);
     sensorMin=sensorY3;
  }
  }


//Define the threshold
float mean= (sensorMin + sensorMax)/ 2;
threshold = 0.80 * threshold;
```

Actually, the calibration of the sensors through Arduino is useful only if the device is used without the graphical interface described in the next pages, otherwise the calibration process is managed through Processing.

After the calibration, the *loop* starts. Through the *AnalogRead()* of Arduino, we read the 4 pins relative to the different sensors.

The value read on each of them has to be compared with the threshold already defined in order to derive which sensor has been covered by the shadow of the finger. In particular, the useful signal is given by the fact that the sensor's signal is below the threshold, meaning that its voltage is reduced because of the increased resistance in dark conditions.

Therefore, with an if-cycle, we have compared the output from each sensor with the threshold. When the voltage of one of them is below, a message on the Serial is printed containing the instruction of the direction in which the ball will have to move ('up'-'down'-'left'-'right') and the numeric value of the output of the relative sensor.

At the end, a delay is inserted in order to read the values every 100 ms.

```
 if (sensorValue1 > threshold && sensorValue2 > threshold && sensorValue3 > threshold && sensorValue4 > threshold)
  {
    Serial.println("Stop");
  }
  else if (sensorValue1 < threshold)
  {
    Serial.println("Left");
    Serial.println(sensorValue1);
  }
    else if (sensorValue2 < threshold)
  {
    Serial.println("Down");
     Serial.println(sensorValue2);
  }
    else if (sensorValue4 < threshold)
  {
   Serial.println("Right");
   Serial.println(sensorValue4);
  }
   else if (sensorValue3 < threshold)
 {
   Serial.println("Up");
   Serial.println(sensorValue3);
  }


  delay (100);
}
```

By looking at the values showed on the Serial we had the possibility to set our experimental conditions, better understanding the behaviour of the sensors and the range of values within their responses varied under different light conditions, such as natural light -baseline condition-, artificial lamp light of higher intensity and total darkness.

Software

In order to allow the communication between Arduino and Processing we have installed the library Firmata in Arduino IDE and the library "Arduino" in Processing. In this way we have created in Processing the Arduino object, thanks to which it is possible to apply the methods defined for that class of objects. In the first part of the code we have also defined the variables that would be used later on. In the *setup()* we have configured the Arduino by connecting it to the serial port of the computer. In order to know to which port it has been connected we have scanned the serial ports of the PC and we have selected the port identified by the index 0.

Arduino = new Arduino(this, Arduino.list()[0], 57600);

At this point the object Arduino corresponds to our Arduino Micro.
Also, in the *setup()*, we have defined the dimension of the window and created the object representing the timer. Again inside the *setup()* we have implemented the calibration of the sensors. It takes 5 seconds (5000 milliseconds) and during this period of time a message is shown to the user on the screen in order to inform them that thevcalibration is ongoing and they do not have to touch or cover the sensors. In this moment in fact it is important that the sensors are exposed to the baseline light conditions of the environment in which they are located. The calibration, similarly to what has been done on Arduino IDE, is implemented by reading the values on the 4 analogue pins of the Arduino for 5 seconds and comparing them in order to obtain the maximum and minimum value recorded during the calibration period. The average between these two values is computed and stored in the variable *average*. The threshold is calculated as the 80% of this value, a percentage that we have defined experimentally.
Calibration of the sensors is fundamental in order to manage different settings and working conditions that can affect the behaviour of the LDRs. Since they are photoresistors, their functioning is light-dependent but it is very difficult to work every time in the same environmental light conditions. From many trials we have noted that, depending on the light source hitting them, they were more or less able to respond to the stimulus and therefore the movement of the sphere was affected by this uncertainty. In order to tackle this problem and to have a coherent behaviour of the system independently of the external factors we use the calibration time to measure the baseline conditions every time the application has to run and then, based on the values obtained, the threshold is computed. A sensor in fact is considered stimulated, i.e. obscured, if its output is lower than the threshold. By setting a proper value of the threshold we have also solved the problem of the shadows on the other sensors while stimulating one of them. Indeed, the user has to approach the finger to the breadboard in order to cover the correct LDR but, since they are quite close each other, it is frequent the problem of having multi-activation of more than one sensor, resulting in a wrong movement of the ball with respect to the position of the finger. By considering a threshold lower than 100% of the average this problem has been solved.

In the *draw()* we have loaded the menu page of the application where the user has the possibility to choose between starting the game or performing a test in order to try and get familiar with the device.

The virtual object that will be moved by covering the sensors is a circle and the coordinates of its center (stored in the variables *mx, my*) are adjourned every time the scanning of the sensors is performed, that is every 100 ms in our case. In order to read the values from the four Arduino's pins we have used the function *analogRead()* and we have stored each value in a different variable. With

an if-cycle, the value on each pin is compared to the threshold previously defined and this information is used to apply the movements to the virtual ball. When one signal is below the threshold, it means that the relative sensor has been covered by the finger and therefore the ball has to move in the direction represented by the position of the sensor. In order to apply the movements to the object, the output measured on the relative pin is added or subtracted to the x or y coordinate of the center of the circumference.

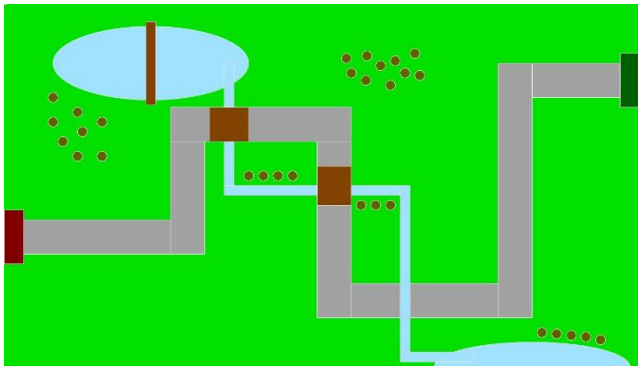
In order to have fluid movements, the increment that is added or subtracted to the coordinates, that is the output of a sensor, is mapped from an integer value ranging (0:1023) to a float value ranging (-30:0) through the Processing function *Map.* Our goal was to have a modulation of the velocity based on the distance between the finger and the sensor. In particular we would like that the closer the finger the faster the movement since it is more intuitive for the patient: they can associate it to the increasing effect obtained in general when a button is pressed more intensely like in the case of a car accelerator, where the strength of the pressing force modulates the velocity of the car itself. In order to obtain this effect we have mapped the output signal from the sensors into negative values. When the finger approaches the LDR, the level of light hitting it decreases, generating an increase of its resistance. This is inversely related to the output voltage that in turns decreases. By mapping these values on a negative range (-30:0), the lower the signal read on the sensor the bigger in absolute value the quantity added or subtracted to the coordinates of the ball and therefore the faster the movement. The range (-30:0) has been chosen in order to have a value of the increment small enough to perceive a continuous movement of the ball. To further increase the velocity modulation we have decided to map only the output values from 0 to *threshold* instead of the whole range (0:1023) since we are interested in the modulation of the velocity only when the sensor is active, i.e. when its output is lower than the threshold. By shrinking the range of the input values and enlarging the range of output values provided to the function *Map* the increment corresponding to a variation of the sensor's signal is smaller and so the resolution is increased. In this way we have built a non-linear relationship between the signal read on the sensor and the variation of the velocity so that, instead of having discrete movements consisting in "jumps" from one point to another, we have obtained continuous movement of the ball.

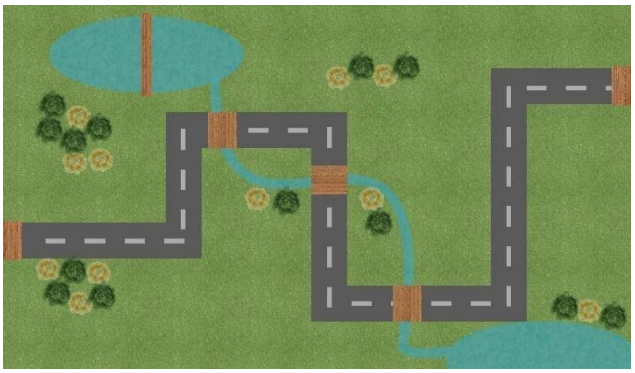
Considering the negative sign of the quantity that is added or subtracted and remembering that the coordinate system of Processing has the origin in the top-left angle so that the direction of the y axis is reversed with respect to the conventional one, the movement of the ball has been implemented as follow:

```
if (up < threshold) {  //it must move forward
  up = (int)map(avanti, 0, threshold, -30, 0);
  my = my + up;
 } else if (left < threshold) {  //it must move left
  left = (int)map(sinistra, 0, threshold, -30, 0);
  mx = mx + left;
 } else if (right < threshold) {  //it must move right
  right = (int)map(right, 0, threshold, -30, 0);
  mx= mx - right;
 } else if (down < threshold) { //it must move backward
  down = (int)map(down, 0, threshold, -30, 0);
  my = my - down;
 }
```

In particular, if the upper sensor is active, its output is added to the y-coordinate since the ball has to move in the negative direction of the y-axis. Therefore a negative quantity is added resulting in a subtraction of values from the y-coordinate and so in a translation upward.
Vice versa, the output is subtracted when the active sensor is the one at the bottom since the ball has to move in the positive direction of the y-axis. By subtracting a negative value we obtain an increment of the y-coordinate and so a movement downward.
The same reasoning can be applied for the remaining two sensors: when the one at left is active the output (that is a negative quantity) is added to to the x-coordinate so that the ball moves in the negative direction of the x-axis (i.e. toward left) and when the sensor at the right is active the output is subtracted to the x-coordinate so that the ball moves in the positive direction of the x-axis (i.e. toward right).

The graphical pathways to be followed have been created in Processing through rectangles of different dimensions connected one to the other through a common edge. We have carefully set them to have the same width along the main dimension in order to have a precise wideness of the path characterizing each level and therefore increase the level of difficulty by shrinking it. Also the other components of the image (trees, lakes, bridges), that have only the aim of creating an enjoyable and more realistic interface for the user, have been created through geometrical shapes. Later on, in order to further increase the quality of the interface, the image created on Processing has been elaborated with Photoshop.



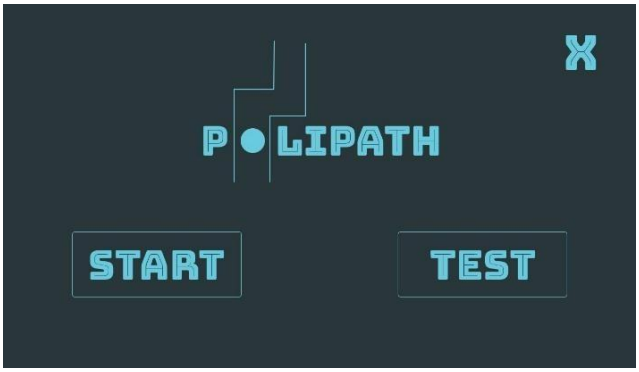


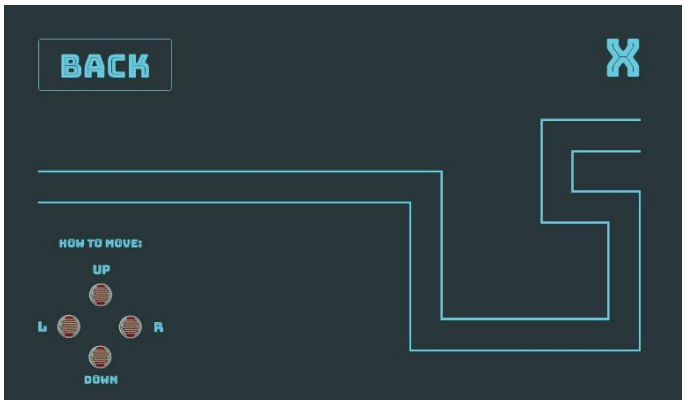
Again with Photoshop we have realized the pages to be shown at the beginning and end of the game and in between the different levels, in which two or three buttons are shown in order to allow the user to interactively select different options.

Since our application consists of three different levels, we have organized the code in different functions, one for each level. In addition, we have created a function called *mouseClicked()* in order to implement the interactive graphical interface. A detailed description of each of them is reported below.

void mouseClicked():

This function is used to control what the patient selects when an interactive screen is shown. It happens for the first time at the beginning of the game, where they have the possibility of starting directly to play or of performing a simple test of the functioning of the device.





The test consists of a very simple and short pathway in which the user can try the device to get more familiar with before starting the real game. Always in this page, the user can easily understand how the velocity changes depending on the distance between the finger and the sensor and can visualize which sensor is activated by the touching thanks to the highlighting of the relative sensor on the image shown on the screen.

At the end of each level, the user is shown an end page where they can visualize the time they took to accomplish the level and some options which they can select. The user needs to decide whether they want to retry the past level, perhaps with the aim of improving the time, or to proceed with the next level or to leave the game clicking on the cross.

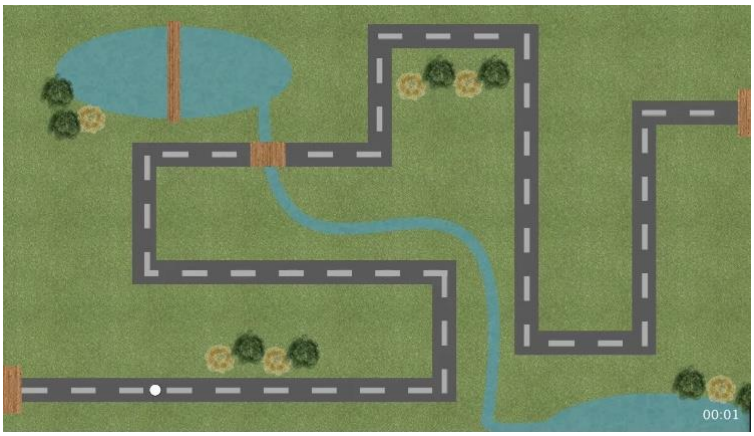
At the end of level 2 and level 3, the user has a third option to restart the game from level 1.

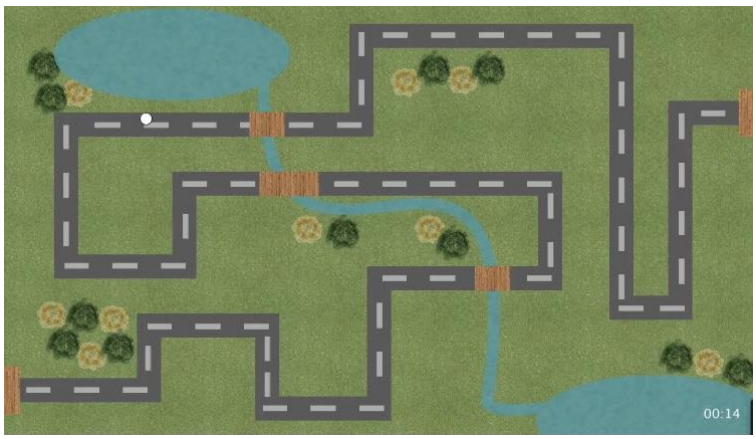
The control on the selection made by the user is done through the definition of boolean variables that are set true or false depending on the coordinates of the mouse when the selection is done, i.e. in the moment in which the mouse is clicked.

void Level(i)():

This function is called every time a new level has to start. At first, the image representing the path relative to each level is charged and set as background. Then, the virtual object is created and the commands are applied to its coordinates so that it can move according to the signals received by our sensors. Inside each level a further function is created in order to manage the measure of the time spent to carry on that level. This is the function *time()* and it allows to visualize a running timer in the right bottom corner for the entire duration of the game which is stopped when the end of the path is reached, thus showing the time passed. In order to create the timer we have defined the class *Timer* and an object belonging to it. Inside the class we have defined the methods to start, stop and

reset the timer, the ones to get the elapsed time and to return it in terms of minutes and seconds. This has been done by exploiting the Processing function *millis(),* which returns the number of milliseconds since starting the program, thanks to the communication of the program with the timer of the computer. In this part of the code we have also implemented the constraints in order to avoid the exit of the virtual sphere from the borders of the path. It has been done by firstly defining a boolean variable that is true when the ball is inside the path and so it is allowed to move, while becomes false when the ball exits the borders. The value of the boolean is set through a series of if-cycles which compare the coordinates of the ball with the numerical values of the limits of the road. These values are known since the realization of the pathway has been done by defining different rectangles in Processing. Therefore, when the boolean is true, the movements are applied to the ball as previously described while when it is false no movement is applied and the coordinates of the sphere are adjusted so that they coincide with the border itself. In this way the effect we have obtained is having virtual invisible "walls" that limit the movement of the ball inside the path, avoiding its exit. Finally, when the game is completed by the user, the end page appears giving the possibility to retry the current level, restart the game from the beginning, go to the next level or close the application.





Conclusion

There exist a few devices in the rehabilitation field which involve a game displayed on a screen and requires the patient to do simple movements in order to achieve some goals. They are mainly realized through virtual reality, but we think that our device could be a valuable alternative: its functioning is very simple and intuitive, the dimension is small and also the cost is limited, since LDR sensors are very cheap. However it is very effective since the patient can train comfortably at home whenever they want, without the need of going to dedicated structures where more complex technologies and clinicians are present. Data about the number of trials and the time required to fulfil each level can be easily stored and shared between the user and the provider but also among different users. The creation of a virtual community where patients can meet each other and be involved in challenges would make the application more engaging, stimulating the patient to often use it and to improve the results in order to climb the rankings.

A possible improvement of the application could be the realization of other levels, inside which some objects suddenly appear and which the patient has to reach or to avoid. Another interesting information could be the measurement of the errors done by the patient, not only in terms of number of times they hit the limits of the pathway but, more interesting, in terms of time spent covering the same sensor when the borders are reached. This could help the provider to understand how much time is needed for the patient to understand that they are giving the wrong command and that they need to change the direction: thus this data could be used to derive the entity of the impairment, whether it is physical -such as tremor- or somatosensorial -such as visuo-perceptual alteration.

Link for the video:

https://polimi365-my.sharepoint.com/:v:/g/personal/10790892_polimi_it/EQaKiA120KdKrin1IL2m8sYBh3JGsE_NtwT2sHn07kX2Vw?e=gVjALO