POLITECNICO
DI MILANO

16 June 2019

FINAL REPORT: project 15

Group 15:
Anita Caracciolo, Dario Comini, Chiara Gilli
Tam Huynh, Rossella Indaco, Ada Nowakowska

Contents

Introduction

Purpose

Our project is named "Follow your path" and it is based on an LDR sensor. The main goal of the project is to develop a black path and to follow it with the sensor which should be able to detect any falling off the line.

Aim of the project

We have decided to create a system dedicated to the analysis of tremor and motion-impairing disorders through handwriting tasks.

Handwriting is a complex motor skill, which involves the interplay of arm and hand muscles, as well as precise hand-eye coordination. This is why handwriting difficulties may be one of the earliest symptoms present in patients with Parkinson's disease. [2]

Parkinson's disease is a neurodegenerative disorder characterized by the gradual onset of muscle rigidity, bradykinesia (i.e. slowness of movement) and, in some patients, tremor. Simple drawing tests, easily applicable in a clinical environment, since only a digitizer pen and tablet are needed to perform the measurements, can supplement the neurological examination and can also be useful for monitoring the effects of rehabilitation programs or other interventions. [1] [2]

LDR sensor

The Light Dependent Resistor generates an output signal indicating the intensity of light. These sensors are known as photoelectric devices or photo sensors. Photoelectric devices are divided into two main groups: those which generate electricity when

illuminated (photo-voltaic cells, photo-emissive cells) and those which change their electrical properties (photo-conductive cells).

The photoresistor changes its electrical resistance in response to changes in the light intensity. When the intensity of light increases, the resistance goes to its minimum value, ideally zero. When no light is falling on the LDR then its resistance goes to its maximum value, ideally infinite.

LDRs are made of semiconductor material. The most commonly used is Cadmium Sulphide (Cds) with spectral responses similar to that of the human eye. It has a peak sensitivity wavelength of 550 nm in the visible spectral range.

Since LDR are very cheap light sensors, there is a wide range of applications including both scientific researches and everyday residential applications (burglar alarms, garage door openers, security systems, etc.).

Test

In our project we have created four different paths of increasing difficulty that the patient has to follow with a particular pen to determine the tremor severity.

The pen has at the tip the LDR sensor. The path is a black line on a white surface and that allows to count the percentage of time spent on either color thanks to the high contrast between the two colours. Another aspect that we have taken into account is the time needed to perform the single test in order to assess bradykinesia and analyzing the slowness of movements.

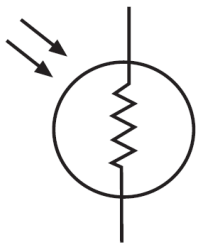
How to perform the test:

1. Select the desired path;
2. Adjust the screen's brightness level to the maximum;
3. Get in position with the pen on the starting point;
4. Press "ENTER" on the keyboard in order to begin the test and to start the chronometer;
5. Press "SPACE" on the keyboard to stop the chronometer and to obtain the results when end is reached.

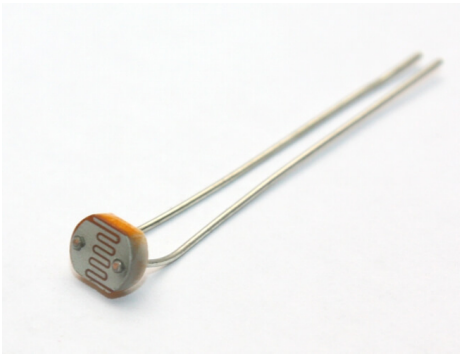
Hardware

The hardware of the project is composed by:

- LDR resistor;
- Arduino Uno board;
- breadboard;
- resistors;
- amplifiers;
- micro piezoelectric speaker.



(a) LDR symbol



(b) LDR component

As said before, the LDR's resistance decreases with the increase of illumination, because more and more electrons can reach the conductive band. Thanks to this feature LDR can be used to detect the path line, as more light is reflected from a white surface than a black one.

However, when the sensor is put in a bright environment or under sunlight, the measurements can be very different, and this must be considered when calibrating the sensor. To overcome this problem, we put the LDR inside a pen, covering it from external light, and used screen emitted light instead of reflected light.

Transduction stage

To realize the circuit, we used a Wheatstone bridge configuration followed by an amplification stage. The Wheatstone bridge has the purpose of lowering the offset that would otherwise be present from a simple voltage divider circuit.

For the calibration of the circuit components, we first used a multimeter to measure the resistance given by the sensor when exposed to the maximum light (White, W) and to the maximum darkness (Black, B).

$$LDR_{BLACK} = B = 10k\Omega$$

$$LDR_{WHITE} = W = 4.5k\Omega$$

The resulting output of the Wheatstone bridge, given enough input impedance of the amplification stage and 5V supply voltage, is then (see schematic, Figure 2):

$$V_o^{Bridge}(LDR) = 5\left(\frac{LDR}{LDR + R_S} - \frac{R_4}{R_1 + R_4}\right)$$

To maximize the range between the two thresholds we must analyse the range function:

$$\Delta V_o^{Bridge} = 5\left(\frac{B}{B + R_S} - \frac{R_4}{R_1 + R_4}\right) - 5\left(\frac{W}{W + R_S} - \frac{R_4}{R_1 + R_4}\right) = 5\left(\frac{B}{B + R_S} - \frac{W}{W + R_S}\right)$$

We can infer from the equation above that range depends only on the choice of R_S.

$$\lim_{R_S \to 0} \Delta V_o^{Bridge} = 0$$

$$\lim_{R_S \to \infty} \Delta V_o^{Bridge} = 0$$

$$\frac{d}{dR_S}\left(\Delta V_o^{Bridge}\right) = \frac{5W}{(W + R_S)^2} - \frac{5B}{(B + R_S)^2}$$

Hence, we can find the value of R_S which ensures the maximum range:

$$\frac{d}{dR_S}\left(\Delta V_o^{Bridge}\right) \geq 0$$

$$\longrightarrow \frac{5W}{(W+R_S)^2} - \frac{5B}{(B+R_S)^2} \geq 0$$
$$\longrightarrow W(B+R_S)^2 - B(W+R_S)^2 \geq 0$$
$$\longrightarrow W(B^2+2BR_S+R_S^2) - B(W^2+2WR_s+R_S^2) \geq 0$$
$$\longrightarrow R_S^2(W-B) + 2R_S(WB-BW) + WB^2 - BW^2 \geq 0$$
$$\longrightarrow R_S^2 \leq \frac{WB^2-BW^2}{B-W} = \frac{WB(B-W)}{B-W}$$
$$\longrightarrow 0 \leq R_S \leq \sqrt{WB}$$

As the derivative is first positive and then negative at the critical point $\sqrt{WB}$, this is a global maximum:

$$\longrightarrow R_S = \sqrt{WB} \approx 6.7k\Omega$$

Therefore, the full range is:

$$\Delta V_o^{Bridge} = 5\left(\frac{B}{B+R_S} - \frac{W}{W+R_S}\right) \approx 0.99V$$

From the equation of V_o^{Bridge} we can see that the second part $\frac{R_4}{R_1+R_4}$ could be used to shift the signal, reducing the offset. Ideally, we want voltage output on "white" to be 0:

$$V_o^{Bridge}(W) = 5\left(\frac{W}{W+R_S} - \frac{R_4}{R_1+R_4}\right) = 0$$

From the previous equations we can substitute $R_S = \sqrt{WB}$

$$\frac{R_4}{R_1+R_4} = \frac{W}{W+\sqrt{WB}}$$
$$R_4 = R_1 + R_4\frac{W}{W+\sqrt{WB}}$$
$$R_4\left(1 - \frac{W}{W+\sqrt{WB}}\right) = R_1\frac{W}{W+\sqrt{WB}}$$
$$R_4\left(\frac{\sqrt{WB}}{W+\sqrt{WB}}\right) = R_1\frac{W}{W+\sqrt{WB}}$$

$$R_4 = \sqrt{\frac{W}{B}}R_1 \approx 0.67R_1$$

So, we have decided to use:

$$R_4 \approx 6.7k\Omega$$

$$R_1 \approx 10k\Omega$$

Amplification stage

To exploit the full dynamic range of the built-in ADC of the Arduino Board, from 0 to 5V, we used an amplification stage. We decided to use one operational amplifier in a differential configuration. The output is then:

$$V_o^{Amplifier} = V_o^{Bridge} \frac{R_2^{Amplifier}}{R_1^{Amplifier}}$$

As said, input impedance must be high to overcome the problem of measuring distortion. As the full range of the bridge ΔV_o^{Bridge} is about 1V, we chose a gain of 5:

$$R_2^{Amplifier} \approx 1M\Omega$$

$$R_1^{Amplifier} \approx 200k\Omega$$

The output is then connected to Arduino's analog input.

Feedback

To have a feedback of being correctly following the path with the sensor, other than a visual graph during the session, we decided to use a mini speaker. It announces the starting and ending of a session and warns the user of being out of the path with an intermittent beep. The speaker is simply connected to ground and to a digital output of Arduino.

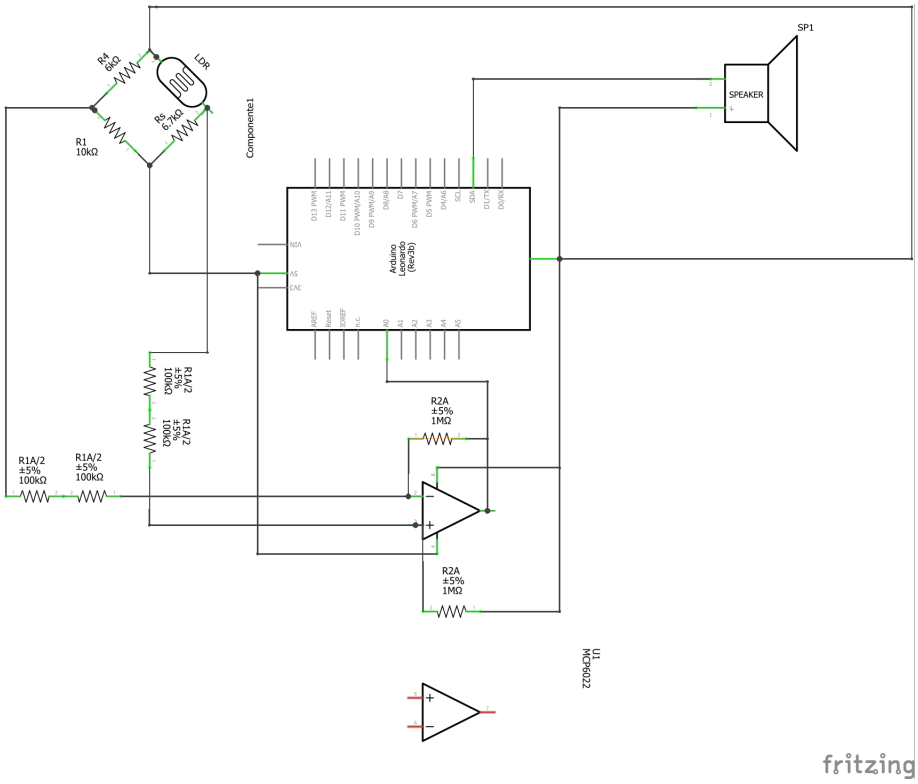


Figure 2: Schematic of the circuit

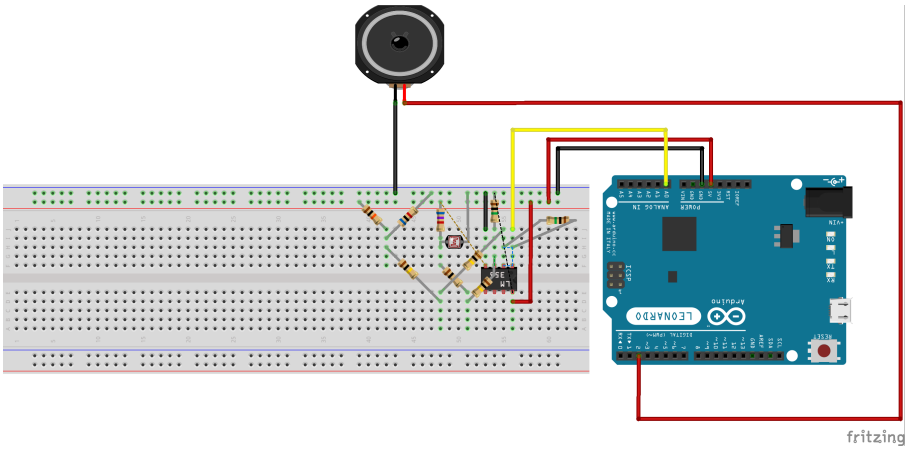


Figure 3: Positioning of all the components

Firmware

After the initialization of pins and the declaration of global variables, we read and converted the output of LDR in the main part of the Arduino code with the function `analogRead()`, which transforms the analog voltage to a digital signal with value range [0-1023]. `sensorOut = analogRead(sensorPin)`; We have implemented a threshold that is able to discriminate when the LDR is passing over a dark region or a white one. After several tests, we set the threshold to 500.

```
if (sensorOut > 500) {
    sensorOut = 1023; //sensor is on the black path
    delay(100); //used to synchronize Arduino serial output
}
```

When the sensor goes on the white background it starts an acoustic alarm.

```
else if (flag == 1) {
//if a test session has started: alarm can be played
    sensorOut = 0; //sensor is on the white background
    tone(buzzerpin, SOL/2, 50);
    delay(50);
    tone(buzzerpin, 0, 50);
    delay(50);
} else {
    sensorOut = 0;
    delay(100);
  }
```

Moreover we have created two different melodies that indicate both start and end of the test. They are implemented firstly with the declaration of the music notes, for example **float** `SI = LA*pow(semitone,2)`, and then with the alternation of the two functions `tone()` and `delay()`.

We then pass the binarized data `sensorOut` to processing with the function `Serial.write(sensorOut)`.

Software

Main page

The user interface has been programmed with Processing, in order to provide simple and intuitive use. The main screen's title is "Choose your path":

```
textSize(100);
textAlign(CENTER);
fill(0,204,0);
for (int x = -1; x < 8; x++) {
    text("CHOOSE YOUR PATH", width/2+x, height/2-300+x);
}
fill(0, 153, 0);
text("CHOOSE YOUR PATH", width/2, height/2-300);
```

Below, the user can see below the four modes: easy, medium, hard, extreme, with the relative thumbnails showing a preview of the paths. When the mouse passes on an icon, a description text appears, the thumbnail enlarges and changes its opacity. The PNG thumbnails are loaded in PNG with the function `loadImage(.PNG)`.

```
for (int i=0;i<4;i++){
    if (i==(rectOver-1)){
        textSize(30);
        textAlign(CENTER);
        fill(0);
        text(thumbLabels[i], thumbX-600+400*i, thumbY+200);
        tint(255, alpha);
        image(paths[i], thumbX-600+400*i, thumbY, thumbLength
        +20, thumbLength+20);
    } else {
        tint(255, beta);
        image(paths[i],thumbX-600+400*i,thumbY,thumbLength,
```

```
        thumbLength);
    }
}
```

On the bottom of the screen, the "STATISTICS" button can be used to jump to the relative page. With a similar concept, the button lights up when the mouse passes over it:

```
if (statOver) {
    fill(0, 204, 0);
    rect(thumbX, thumbY+400, menuW*4, menuH*2,5);
    textSize(70);
    textAlign(CENTER, CENTER);
    fill(255);
    text("STATISTICS", thumbX, thumbY+390);
}
```

Statistics page

The statistics page is accessed by clicking on the "STATISTICS" button. This page shows a graph that indicates on 'x-axis the different runs, on the left axis of the orders the percentage of time spent on black (accuracy) and on the right axis of the orders the time spent to end the path. This is implemented loading a .csv file from an external folder and creating several arrays in which the values of the columns are saved. The arrays' change in length is based on the number of rows in the table `table.getRowCount ()`. With a for loop that cycles all the rows, the various values corresponding to the relative index are saved in the arrays and then these values are added to the coordinate array that will be used to draw the graph.

```
table = loadTable("statistics.csv","header");
stats = new GPointsArray();
int[] ID = new int[table.getRowCount()];
int[] percentage = new int[table.getRowCount()];
String[] totalTime = new String[table.getRowCount()];
//int[] percorsi = new int[table.getRowCount()];
int i=0;

for (TableRow row : table.rows()) {
    ID[i] = row.getInt("ID_run");
```

```
    percentage[i] = row.getInt("pBlack");
    totalTime [i]= row.getString("Time");
    //percorsi [i] = row.getInt("Path");
    stats.add(ID[i],percentage[i]);
    i++;
}
```

Path page

When a mouse button is pressed on a specific thumbnail, it opens another page, that is handled by the function `void path ()`. Inside this screen we have differentiated the four paths with a switch. A description of the drawing functions for every path will be addressed later. The user can return to the main screen by pressing the "BACK" button at the right bottom of the screen:

```
if (overRect(backX, backY, menuW, menuH)) {
    backOver=true;
} else backOver=false;

if (backOver) {
    fill(0, 204, 0);
    rect(backX, backY, menuW, menuH, 5);
    textSize(30);
    textAlign(CENTER, CENTER);
    fill(255);
    text("BACK", backX, backY-3);
}
```

The user must press the "ENTER" or "RETURN" button on his keyboard to start the test, and the "SPACE" button to end it:

```
if (keyPressed && openResults==false) {
    if ((key==ENTER || key==RETURN) && !trackStarted) {
        setup();
        trackStarted=true;
        start=millis();
        myPort.write(65);
        points = new GPointsArray();
    } else if (key==32 && trackStarted) {
```

```
        trackStarted=false;
        myPort.write(66);
        save = 1;
        openResults=true;
        setup();
    }
}
```

In the upper right corner of the screen there is a chronometer, which starts to count when "ENTER" is pressed.

```
for (int x = -1; x < 8; x++) {
    text("Time: "+stime+"s", 0.75*width+140+x, 175+x);
}
text("Time: "+stime+"s", 0.75*width+140, 175);
```

The variable `time` is counted during the test: `time=(millis()-start)/1000`, and then transformed in a string: `stime=nf(time, 0, 2)`.

Below, the percentage of the time spent on the path and on the background is shown:

```
text("Time on black:  "+pBlack+"%", 0.75*width+140, 300);
text("Time on white:  "+pWhite+"%", 0.75*width+140, 400);
```

This is where the variables `pBlack` and `pWhite` are updated during the test:

```
if (sensorValue==0)
        countWhite++;
else
    countBlack++;

sumColor=(countWhite+countBlack);
pBlack = round((countBlack/sumColor)*100);
pWhite = round((countWhite/sumColor)*100);
```

It is also displayed in real time the graph of the performances, with time on the x-axis and with two values on the y-axis: black (path) and white (background). This has been realized with different library functions:

```
plot1.beginDraw();
plot1.drawBackground();
plot1.drawBox();
plot1.drawXAxis();
```

```
plot1.drawYAxis();
plot1.drawTitle();
plot1.drawGridLines(GPlot.BOTH);
plot1.drawPoints();
plot1.drawLines();
plot1.endDraw();
```

Results page

After pressing "SPACE" `openResults`=**true** the flag becomes true and the `results()` function is called. This function shows the complete trend of the performance of the last test session, displayed in a final graph, the percentages of time spent on the path and on the background and an emoticon giving feedback on the results. The four emoticons used are shown below:

(a) Bad

(b) Good

(c) Excellent

(d) Perfect

These four different range of results have been distinguished by analyzing `pBlack` value:

```
if (pBlack>=95){
    image(perfect, width/2+450, height/2-300, emojiDimension,
    emojiDimension);
} else if (pBlack < 95 && pBlack >= 80){
    image(excellent, width/2+450, height/2-300, emojiDimension,
    emojiDimension);
} else if (pBlack < 80 && pBlack >= 70){
    image(good, width/2+450, height/2-300, emojiDimension,
    emojiDimension);
} else {
    image(bad, width/2+450, height/2-300, emojiDimension,
    emojiDimension);
```

}

In addition, two buttons are shown below the results's page: "MENU", to return to the main screen and "AGAIN", to restart the test. At the end of the test, the "save" flag is set to 1 (true) and the results are saved in a .csv file in order to compare them. The code below allows to load the pre-existing .csv file as a table and add a new line with the results of the last run, using the `setInt(), sentString()` functions and then save the new file with saveTable.

```
if (save==1){
    table = loadTable("statistics.csv","header");
    TableRow newRow = table.addRow();
    newRow.setInt("ID_run", table.lastRowIndex()+1);
    newRow.setInt("pBlack", pBlack);
    newRow.setString("Time", stime);
    //newRow.setInt("Path",status);
    saveTable(table, "statistics.csv");
    save=0;
}
```

Drawing of the paths

Now we will explain the implementations of the four different paths.

In the easy mode the path corresponds to a circle with a starting/ending point highlighted in a different color. The function `void circle()` draws the basic figure with `circle ()` and sets color and dimension of the stroke with `stroke()` and `strokeWeight()` functions.

In the medium mode the path to follow consists of multiple lines with different directions intersecting in their median point. The function `void flake()` draws each basic element with `line()`.

In the hard mode the path chosen is a spiral. In the function `void spiral()` we used `beginShape()` and `endShape()` to create this complex form.
`beginShape()` with no mode parameter specified begins recording vertices for a shape and `endShape()` stops recording. The spiral is implemented in a recursive way with a for cicle that runs 90 times the `curveVertex()` function which specifies vertex coordinates for the curve. `curveVertex()` with two parameters specifies a position in 2D and the shape will be outlined with the stroke color and dimension chosen with `stroke()` and `strokeWeight()`.

In the extreme mode the path consists in a sequence of lines that create a "zig-zag" pattern. The function `void zigzag()` uses `line()` and `circle()` to create the drawing.

Results

The user interfaces with the menu screen where he can choose from four different options the level of difficulty of the test.

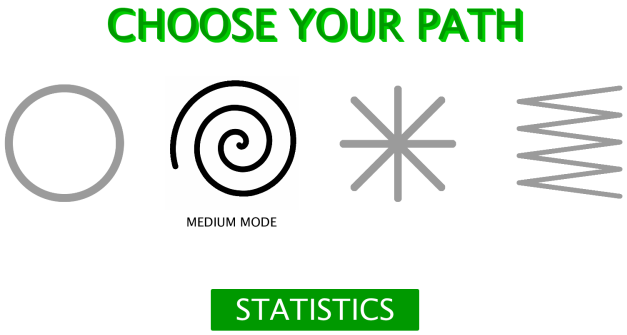


Figure 5: Screenshot of the menu with a selected mode

The following screenshots represent the different paths and the relative scores. Both time spent to follow the entire path and the error are updated in real time and showed in the graph simultaneously.

At the end of each test the interface provides a new screen that shows the final results. To give an immediate assessment of the performance an emoticon which changes according to different range of scores obtained is present.

Moreover a function that allows to save all the results in order to create a clinical history of the patient has been implemented. By pressing the button "STATISTICS" it is possible to have access to all the previous test results: a graph shows for each run both the accuracy and the total time. This tool can be very useful for monitoring the effects of rehabilitation programs or other interventions over a long period of time.

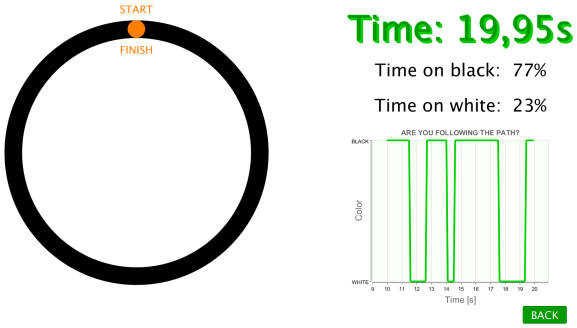


Figure 6: Screenshot of the easy mode with scores

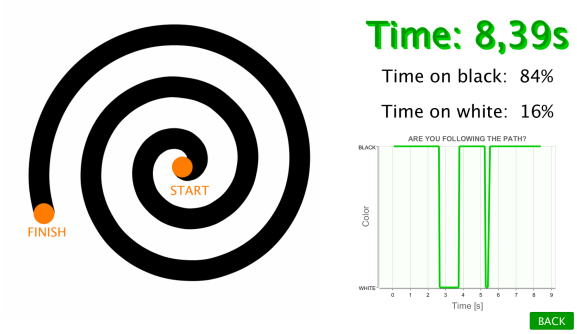


Figure 7: Screenshot of the medium mode with scores

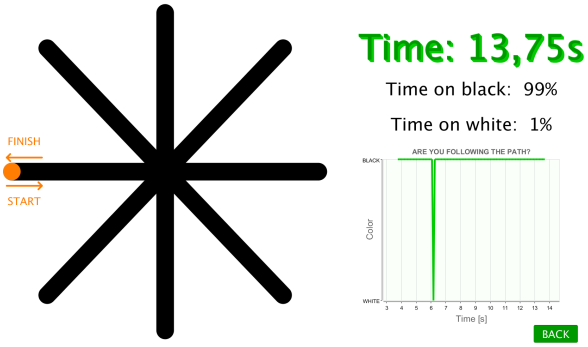


Figure 8: Screenshot of the hard mode with scores

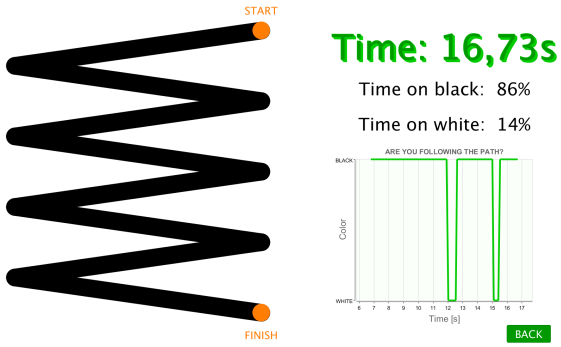


Figure 9: Screenshot of the extreme mode with scores

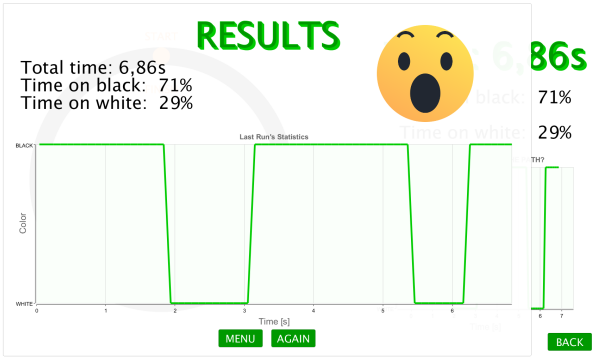


Figure 10: Screenshot of the results

Figure 11: Screenshot of statistics

Bibliography

[1] Jane Alty, Jeremy Cosgrove, Deborah Thorpe, and Peter Kempster. How to use pen and paper tasks to aid tremor diagnosis in the clinic. *Practical Neurology*, 17(6):456–463, 2017.

[2] Esther J. Smits, Antti J. Tolonen, Luc Cluitmans, Mark van Gils, Bernard A. Conway, Rutger C. Zietsma, Klaus L. Leenders, and Natasha M. Maurits. Standardized handwriting to assess bradykinesia, micrographia and tremor in parkinson's disease. *PLOS ONE*, 9(5):1–8, 05 2014.