CUNEO ITIS DELPOZZO

Students:

Cristian Andrei Cojan,

Leonardo Tassone,

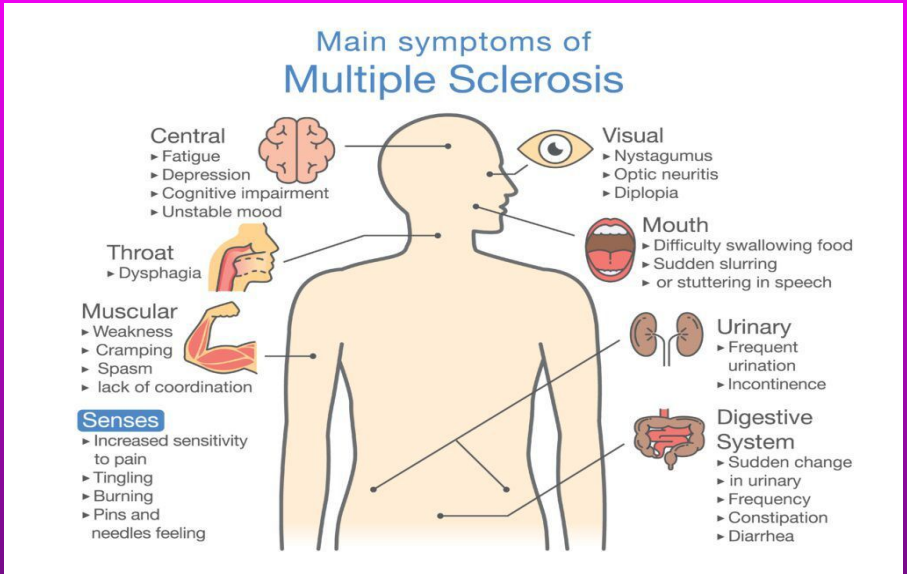
Canonico Francesco,

Barale Novero Alessandro,

Meinero Samuele

The Idea:

Our idea is to recreate a holographic piano playable with the use of our feet, so as to induce the movement of a person but also call upon imagination.
The intelligence that we are using for the body tracking is called OpenCV, in that way, with the help of a camera we can track the position of the body so the piano can be easily played.
This game is projected and created by us for the people suffering from multiple sclerosis or amyotrophic lateral sclerosis. With this program, they can play the piano anyway, though if they are on the wheelchair.



The MS and the ALS are a neurodegenerative pathology of the central nervous system, it manifests visive, sensibility, muscles and many other problems.
Our goal through this project is to raise a smile to these people.

Devices:

For this project we didn't use many physical devices, because most of the work is done by the code, therefore we used only:

- **Camera**
- **Projector**

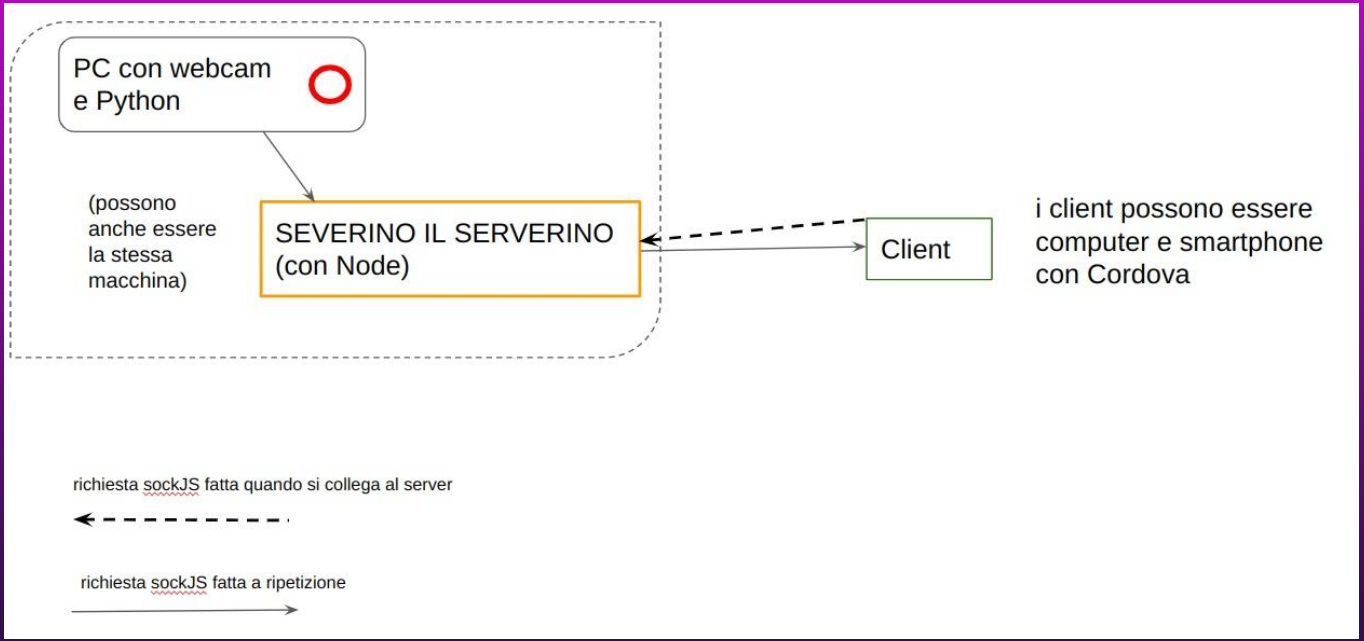
For coding:

- **Nodejs**
- **Python**
- **HTML**
- **Javascript**
- **CSS**
- **OpenCV**
- **npm**
- **Javascript online libraries**

Schematic:

The logic of the project is resumed in the following photo:

“SEVERINO IL SERVERINO” is our server that manage our data, the “Client” is the pc where the program is running.



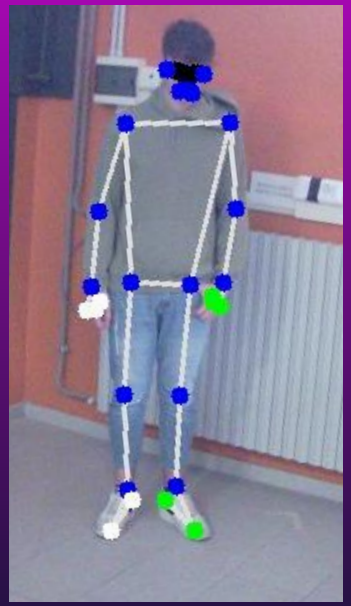
Implementation

When we started thinking about this project the first thing that we did was to imagine the start screen of the application.
We used different code languages for the implementation of the project, first we tried to make a sketch on a whiteboard of the game screen. Then we started to code it in HTML and CSS, here some of us created the home page and the one with the piano.
Meanwhile others started to create a server to manage the data and implement the code on Javascript for the web sockets but also for the virtual pressed buttons.
In python we implemented the code for the pose estimation so that can recognise the one that is playing the piano.

Link for the Github with all the codes:
https://github.com/theCocoCj/I-Severini

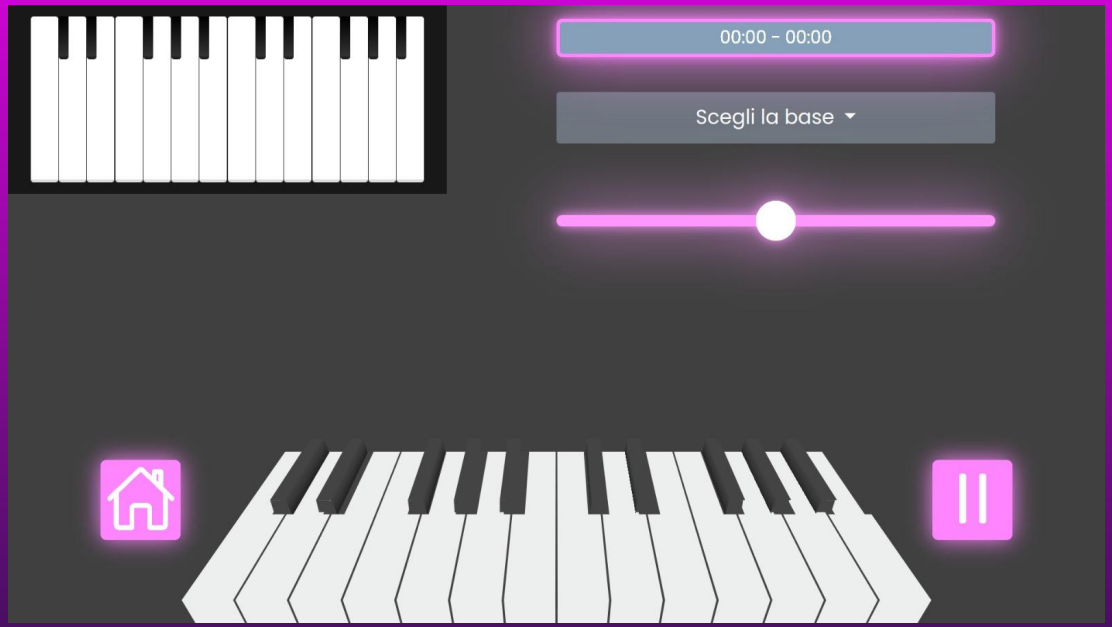
OpenCV

This technology is called OpenCV, it is an Open Source Computer Vision Library that contains functions for real-time computer vision. Through these functions the image can be processed to identify some dots on the body which helps us to use the feet as a mouse. Here is an example of how we use OpenCv:



As we can see OpenCV tracks a lot of dots, but we only use the ones on the feet and on the hands. As we have already said, the feet are used to play the piano and to move on the site, while the ones on the hands are used only for the calibration.

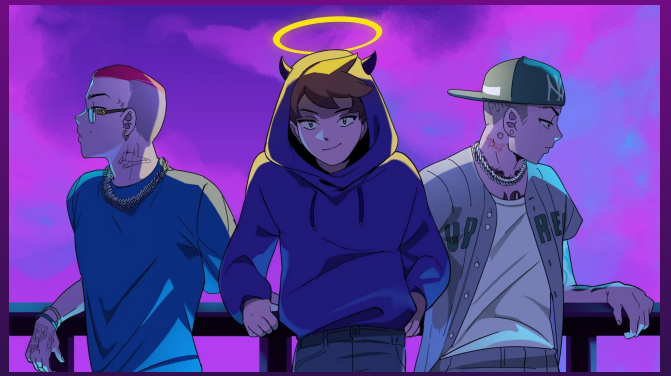
Piano with musical base



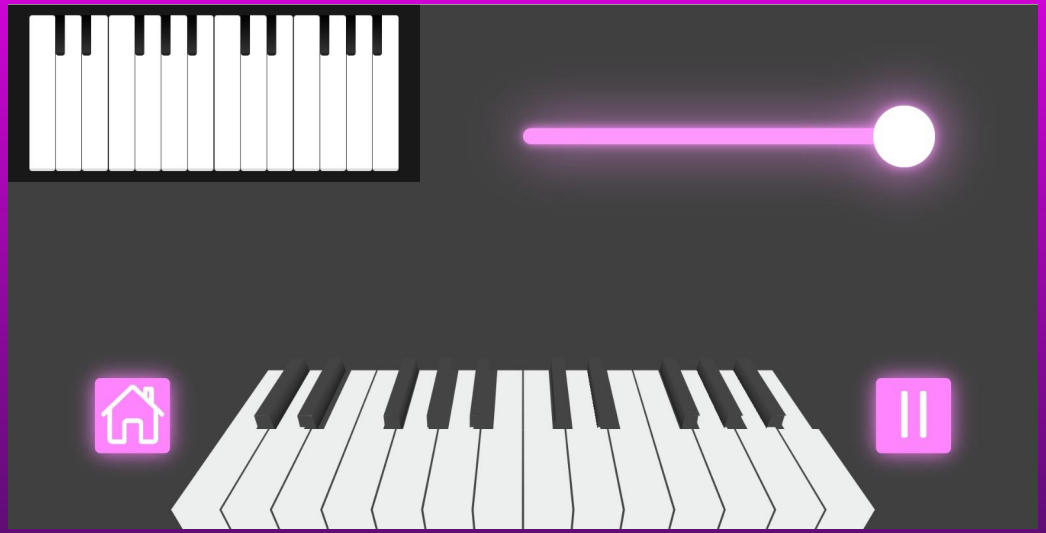
The piano can be played on the screen of the device or using the feet on the ground with the projection. On the left of the piano there is a button to return to the Homepage while on the right there is a button to pause the music. In the background it is also possible to choose a song and to change the volume of it.
An important part of the screen is the piano 2D where the tiles and the corresponding musical notes are shown.

The bases

- Dimmi Che C’è - thasup ft. Tedua
- s!r! - thasup ft. Lazza & Sfera Ebbasta
- Cheeky - Good Enough
- dreamy nights - ikeya
- Bridges - softy



Piano libero



The screen of this page is very similar to the other one, with the difference that there isn’t any music so some of the buttons aren’t necessary.
Here the piano is played without musical accompaniment.

Calibration

This page is one of the most important of the site and the project. In such a manner, the game can work in the correct way.
If the calibration hasn't been done yet and we try to play, we are redirected to the calibration page.

The calibration is very easy to do. The tip of the foot must be put on all the corners of the projection that at a first time are red, they become green at the same time when the corresponding hand is raised. When all the corners become green the calibration is completed.

Link for the **Calibration tutorial**:
https://github.com/theCocoCj/I-Severini/blob/main/Calibration_Tutorial.mp4

Codes

The hardest part is here, with the Javascript codes:
There are some modules and libraries that have to be installed:
npm is installed with Node.js, this means that you have to install Node.js to get npm installed on your computer.

Link for **Node.js** download:
https://nodejs.org/en

For the server there are some additional modules, used for most of the javascript framework, that must be downloaded before starting it, in particular:

- ➔ **NODEMON** → automatic saving of changes in a project.
- ➔ **FS** → reading and writing of files.
- ➔ **EXEC in Child_Process**→ running a command from the terminal.
- ➔ **WS** →used to create a webSocket server to manage the client's webSocket requests.

these modules can be installed from the terminal in the severe folder writing:
npm i name-of-the-module

- ➔ **Express** → is the most important framework, and with this the server can be created.

Link for the installation of the framework **Express**:
https://expressjs.com/en/starter/installing.html

```python
"""
Code in Python in order to track the body of the user
"""
import cv2
import mediapipe as mp
import time

#function that read the status of the camera from a file
def inClosing():
    f = open('files/statoCamera.txt', 'r')
    l = f.readline()
    f.close()
    return l == 'da spegnere'

# Define landmarks for various body parts using their landmark IDs
PIEDE_SX, PIEDE_DX = [29, 31], [30, 32]         #left and right foot
OCCHI = [n + 1 for n in range(6)]               #eyes
MANO_SX, MANO_DX = [17, 19, 21], [18, 20, 22]   #left and right hand

# Initialize mediapipe drawing and pose detection utilities
mpDraw = mp.solutions.drawing_utils
mpPose = mp.solutions.pose
pose = mpPose.Pose()

# Open the camera and set up some initial parameters
cap = cv2.VideoCapture(2)
pTime = 0

# Write the status of the camera to a file
f = open('files/statoCamera.txt', 'w')
f.write('accesa')
f.close()

# Start a loop to continuously read frames from the camera and process them
while not inClosing():
    # Read a frame from the camera
    success, img = cap.read()
    # Convert the image from BGR to RGB
    imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    # Use mediapipe to detect the landmarks in the pose
    results = pose.process(imgRGB)
```

Python

In this first part of the code the file with the status of the camera is ridden, then the landmarks of the most important dots of this program are initialized.
The camera is setted up and then we write again the status of the camera.
Here a loop read frames from the camera and process them.

```python
    # If landmarks are detected in the pose
    if results.pose_landmarks:
        #mpDraw.draw_landmarks(img, results.pose_landmarks, mpPose.POSE_CONNECTIONS)
        # Open a file to write the landmark data to
        f, content = open('files/lm.txt', 'w'), ''

        # Loop through all the landmarks and draw circles around them based on their body part
        for id, lm in enumerate(results.pose_landmarks.landmark):
            h, w, c = img.shape
            #print(id, lm)
            cx, cy = int(lm.x * w), int(lm.y * h)
            content = content + str(lm) + '\n'

            # Draw a circle around the landmark based on the body part
            if id in PIEDE_DX + MANO_DX:
                cv2.circle(img, (cx, cy), 5, (255, 255, 255), cv2.FILLED) # Right foot and hand
            elif id in PIEDE_SX + MANO_SX:
                cv2.circle(img, (cx, cy), 5, (0, 255, 0), cv2.FILLED) # Left foot and hand
            elif id in OCCHI:
                cv2.circle(img, (cx, cy), 5, (0, 0, 0), cv2.FILLED) # Eyes
            else:
                cv2.circle(img, (cx, cy), 5, (255, 0, 0), cv2.FILLED) # Other body parts

        # Write the landmark data to the file and close the file
        f.write(content)
        f.close()

    time.sleep(0.001)           #time for the servere in order to read data
    cTime = time.time()         #obtain the current time
    fps = 1 / (cTime - pTime)   #calculation of the fps
    pTime = cTime

    cv2.putText(img, str(int(fps)), (70, 50), cv2.FONT_HERSHEY_PLAIN, 3, (255, 0, 0), 3) #print on the screen the fps
    #cv2.imshow("Image", img)
    cv2.waitKey(1)

# closing the files opened previously
open('files/lm.txt', 'w').close()
open('files/closePython.txt', 'w').close()
f = open('files/statoCamera.txt', 'w')
f.write('spenta')
f.close()
```

Python

In the second part the landmarks are written on another file, and some circles are drawn on our points of interest.
After we close the file with all the coordinates, there is a delay, in that way the data can be red without problems.

HTML

```html
<!DOCTYPE html>
<html lang="it">
    <head>
        <meta charset="UTF-8">
        <title>LET'S PLAY!</title>

        <link rel="stylesheet" href="../Stylesheets/homePage.css">
        <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha2/dist/css/bootstrap.min.css" integrity="sha384-aFq/bzH65dt+w6FI2ooMVUpc+21e0SRygnTpmBvdBgSdnuTN7QbdgL+OapgHtvPp" crossorigin="anonymous">

        <script type="text/javascript" src="../Javascripts/requests_answers.js"></script>
        <script type="text/javascript" src="../Javascripts/indexs.js"></script>
        <script type="text/javascript" src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha2/dist/js/bootstrap.bundle.min.js" integrity="sha384-qKXV1j0HvMUeCBQ+QVp7JcfGl760yU08IQ+GpUo5hlbpg51QRiuqHAJz8+BrxE/N" crossorigin="anonymous"></script>
        <script type="text/javascript" src="../Javascripts/calcoloCalibrazione.js"></script>
        <script type="text/javascript" src="../Javascripts/homePage.js"></script>
    </head>
    <body class="body-before" background="../Assets/Images/sfondoHomePage.png">
        <button class="btn btn-primary" type="button" disabled>
            <span class="spinner-border spinner-border-sm" role="status" aria-hidden="true"></span>
            Attendi...
        </button>
    </body>
</html>
```

An example of the html files.
This is the homepage, where there are loaded the scripts of JavaScript but also some elements of HTML taken from Bootstrap.
There isn't much written here because it's almost all contained in the css files

CSS

Here is an example of the CSS file with some of the formatting of the internet page tags.
For all the pages there is a CSS file.
With this, is more easier to make some changes.

```css
#div-a {
    width: 100%;
    display: flex;
    flex-direction: row;
    flex-wrap: wrap;
    justify-content: space-evenly;
    align-items: center;
    position: fixed;
    bottom: 50px;
}

a {
    color: white;
    background-color: #cb6ce6;
    filter: drop-shadow(0 0 20px #cb6ce6) contrast(2) brightness(1.5);
    border: 6px solid #cb6ce6;
    border-radius: 15px;
    text-decoration: none;
    padding: 15px 25px;
    font-size: 30px;
    font-family: tahoma, serif;
    letter-spacing: 5px;
    font-weight: bold;
    transition: all 0.5s ease;
}
```

web socket

```
socket.addEventListener('message', function(evt) {
    let message = evt.data;
    let body = document.getElementsByTagName('body')[0];
    //if the message is requesting the turning on of the camera...
    if(message === ANS.ACCENDI_CAM) {
        body.removeChild(body.children[0]);
        setTimeout(() => {
            socket.send(REQ.SEND_POSITION);
        }, 1000);
        let p = document.createElement('p');
        p.id = 'warning';
        body.appendChild(p);
        setAngle();
        //the code checks whether the calibration has already been completed or not
```

Here is an example of how it works the web socket, the severe makes a request, in that case to turn on the camera, then a part from the html, cald child, is executed.

Javascript calibration

```
// Function to calculate perspective points for each element
function getPuntiEachElement(isPiano, exceptionsElement=[]) {
    setPuntoProspettiva(); // Set the perspective point
    let arr = []; // Initialize an empty array to store the results

    // Calculate the width and height based on the given data
    let widthI = ((dati.b_sx.x - dati.b_dx.x) ** 2 + (dati.b_sx.y - dati.b_dx.y) ** 2) ** .5;
    let heightC = dati.a_sx.y - dati.b_sx.y;

    // Loop through each element and calculate the perspective points
    for(let el of (isPiano)? getPianoCoordinates(): getPuntiPerElement(exceptionsElement)) {
        let x_terra_sx = dati.b_sx.x - widthI * el.punti.b_sx.x / innerWidth;
        let y_terra_sx = rette.down[0] * x_terra_sx + rette.down[1];
        let x_terra_dx = dati.b_sx.x - widthI * el.punti.b_dx.x / innerWidth;
        let y_terra_dx = rette.down[0] * x_terra_dx + rette.down[1];

        let re_sx = getM_Q({ x: x_terra_sx, y: y_terra_sx }, puntoProspettiva);
        let re_dx = getM_Q({ x: x_terra_dx, y: y_terra_dx }, puntoProspettiva);

        let y_to = dati.a_sx.y - heightC * el.punti.a_sx.y / innerHeight;
        let y_bo = dati.a_sx.y - heightC * el.punti.b_sx.y / innerHeight;

        let r_y_to = [rette.down[0], rette.down[1] + y_to - dati.b_sx.y];
        let r_y_bo = [rette.down[0], rette.down[1] + y_bo - dati.b_sx.y];

        let __b_sx_x = (r_y_bo[1] - re_sx[1]) / (re_sx[0] - r_y_bo[0]);
        let __b_dx_x = (r_y_bo[1] - re_dx[1]) / (re_dx[0] - r_y_bo[0]);
        let __a_sx_x = (r_y_to[1] - re_sx[1]) / (re_sx[0] - r_y_to[0]);
        let __a_dx_x = (r_y_to[1] - re_dx[1]) / (re_dx[0] - r_y_to[0]);

        // Push the calculated points for each element to the results array
        arr.push({
            element: el.element,
            punti: {
                b_sx: {
                    x: __b_sx_x,
                    y: re_sx[0] * __b_sx_x + re_sx[1]
                },
                b_dx: {
                    x: __b_dx_x,
                    y: re_dx[0] * __b_dx_x + re_dx[1]
                },
                a_sx: {
                    x: __a_sx_x,
                    y: re_sx[0] * __a_sx_x + re_sx[1]
                },
                a_dx: {
                    x: __a_dx_x,
                    y: re_dx[0] * __a_dx_x + re_dx[1]
                }
            }
        });
    }
    return arr;
}
```

Javascript calibration

```
function getPianoCoordinates() {
    let arr = []; // Create an empty array to store the coordinates of each piano key
    let piano = document.getElementById('piano'); // Get the piano element from the DOM using its ID

    for (let child of piano.children) { // Loop through each child element of the piano element
        let bsx = child.children[0].getBoundingClientRect(), // Get the coordinates of the top-left corner of the first child element (bottom-left key)
            asx = child.children[1].getBoundingClientRect(), // Get the coordinates of the top-left corner of the second child element (top-left key)
            bdx = child.children[2].getBoundingClientRect(), // Get the coordinates of the top-left corner of the third child element (bottom-right key)
            adx = child.children[3].getBoundingClientRect(); // Get the coordinates of the top-left corner of the fourth child element (top-right key)

        // Push an object containing the element and its corresponding coordinates into the array
        arr.push({
            element: child,
            punti: {
                b_sx: {
                    x: bsx.x,
                    y: bsx.y
                },
                a_sx: {
                    x: asx.x,
                    y: asx.y
                },
                b_dx: {
                    x: bdx.x,
                    y: bdx.y
                },
                a_dx: {
                    x: adx.x,
                    y: adx.y
                }
            }
        });
    }

    return arr; // Return the array of piano key coordinates
}
```

WWW

```
/*CONSTANTS FOR REQUESTS = requests for actions called using these.*/
const REQ = {
  CALIBR_FATTA: '100', // if the calibration was already done
  ACCENDI_CAM: '101',
  SEND_POSITION: '102',
  STOP_SEND_POSITION: '103',
  SAVE_CALIBR: '104',
  GET_DATI_CALIBR: '105',
  FOLLOW_POSITION: '106',
  CHECK_LOGIN: '107',
  CLOSE_CAMERA: '108'
};

/*CONSTANTS FOR RESPONSES = requests for actions called using these.*/
const ANS = {
  CALIBR_FATTA: {
    FALSE: '200',
    TRUE: '201'
  },
  ACCENDI_CAM: '202',
  SEND_POSITION: '203',
  SAVE_CALIBR: '204',
  GET_DATI_CALIBR: '205',
  CHECK_LOGIN: {
    FALSE: '206',
    TRUE: '207'
  }
};
```

ROUTES

```
router.get( path: '/downloadPyInt', handlers: function(req :… , res :Response<ResBody, Locals> , next :NextFunction ) {
  res.download( path: 'python/intCamera.py');
});
router.get( path: '/downloadPyExt', handlers: function(req :… , res :Response<ResBody, Locals> , next :NextFunction ) {
  res.download( path: 'python/extCamera.py');
});
router.get( path: '/targz', handlers: function(req :… , res :Response<ResBody, Locals> , next :NextFunction ) {
  res.download( path: 'applications/severini.tar.gz')
});
```