Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería en Tecnologías de la Telecomunicación

Trabajo Fin de Grado

Análisis de patrones de movimiento de bebés mediante esterilla de presión

Autor: Lucía Gómez González

Tutor/es: Antonio García Herraiz

2021

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

Grado en Ingeniería en Tecnologías de la Telecomunicación

Trabajo de Fin de Grado

Análisis de patrones de movimiento de bebés mediante esterilla de presión

Autor: Lucía Gómez González

Tutor/es: Antonio García Herraiz

TRIBUNAL:

Presidente: Bernardo Alarcos Alcázar

Vocal 1º: Andrés Navarro Guillén

Vocal 2º: Antonio García Herraiz

Fecha: ***Septiembre - 2021***

A mis padres por enseñarme a no rendirme, a mi hermana por su apoyo incondicional y a mis amigos por estar ahí.

Agradecimientos

Quisiera agradecer en primer lugar a mi familia, a mis padres por todo su cariño y apoyo durante todos estos años, por animarme a luchar por lo que quiero y no rendirme. A mi hermana, mi otra mitad, por acompañarme en esta etapa tan importante, por animarme en los momentos difíciles y por demostrarme que una hermana es el tesoro más bonito que alguien puede tener.

Por supuesto agradecer también a mis amigos y compañeros, a todos aquellos que han formado parte de esta gran experiencia, por que sin ellos esto no habría sido lo mismo.

Por último, pero no menos importante, a mi tutor Antonio por toda la confianza puesta en mí. A su vez agradecer a Bernardo y Susana por su colaboración y ayuda, sin ellos tres este proyecto no habría sido posible.

Resumen

El presente Trabajo de Fin de Grado (TFG) consiste en el análisis de patrones de movimiento de bebés para la detección de anomalías o posibles patologías. Con ello, pretendemos ayudar a profesionales médicos en el diagnóstico de enfermedades del movimiento. Para eso, utilizaremos unas esterillas de presión, gracias a las cuales podremos recoger los movimientos de los sujetos para su posterior estudio. Estos movimientos quedarán registrados en forma de mapas de presión dinámicos, que evolucionan con el tiempo.

Estudiaremos, entre otras cosas, la evolución de los movimientos a lo largo del tiempo, como varían las diferentes zonas de presión y sus centros, así como, el peso que se ejerce en cada una de ellas. El objetivo principal es que, con los resultados obtenidos de este análisis, los especialistas sean capaces de determinar si existen desviaciones en el paciente, algún tipo de atraso en el movimiento, respecto a su edad, o intuir alguna patología no relacionada con el movimiento.

Palabras clave: Esterillas de presión; Patrones de movimiento; Zonas; Mapa de presión; Patología

Abstract

The present Bachelor's Degree Final Project consists in the analysis of movement patterns of babies for the detection of anomalies or possible pathologies. With this, we intend to help medical professionals in the diagnosis of movement diseases. For this, we will use a pressure mat, thanks to which we will be able to collect the movements of the subjects for further study. These movements will be recorded in the form of dynamic pressure maps, which evolve over time.

We will study, among other things, the evolution of movements over time, how different pressure zones and their centers vary, as well as the weight exerted on each of them. The main objective is that with the results obtained from this analysis, the specialists will be able to determine if there are deviations in the patient, some type of delay in movement, with respect to their age, or to intuit any pathology not related to movement.

Keywords: Pressure mats; Movement patterns; Zones; Pressure map; Pathology

Que nada
nos defina, Que nada nos sujete.

Que la libertad sea nuestra propia sustancia.

Simone de Beauvoir (1908 – 1986)

Índice General

Índice de Figuras

1. Introducción

La salud de los más pequeños es algo que siempre nos preocupa y más aún la de los bebés, los cuales no pueden comunicarse y expresar con claridad si algo les sucede. Por ello, controlar el desarrollo de los más pequeños en los primeros meses de vida es algo a lo que prestar especial atención. Para determinar su correcto estado se tienen una serie de escalas y percentiles.

En ocasiones, controlar determinados aspectos como puede ser el correcto desarrollo de la movilidad, se hace especialmente difícil si tenemos en cuenta que los movimientos de un bebe son involuntarios y muy numerosos.

Por ello, los profesionales médicos, como los fisioterapeutas, buscan instrumentos o técnicas que les puedan servir de utilidad a la hora de diagnosticar patologías relacionadas con este tema.

Una de las tareas en las que hay interés en explorar es el nivel de presión ejercido por diferentes partes del cuerpo cuando está tumbado en una superficie, sentado o de pie. Es difícil comprobar si un paciente ejerce más apoyo en un lado del cuerpo que en el otro, la distribución del peso que se ejerce en cada zona del cuerpo o como va variando el centro de gravedad a medida que avanza el tiempo, entre otras cosas.

Hoy en día la tecnología avanza muy rápido, y podemos encontrar en el mercado herramientas de gran utilidad para la salud, una de ellas son las esterillas o mantas de yoga las cuales, pueden integrar sensores de presión para poder identificar las zonas donde apoyamos partes de nuestro cuerpo y que presión aplicamos en cada una de ellas.

El trabajo desarrollado en este TFG trabajará en las líneas de avanzar en el estudio del registro de datos y su análisis, para generar información que sirva de ayuda a los especialistas de la salud. Para la identificación de dicha información, han colaborado un grupo de fisioterapeutas de la Universidad de Alcalá.

1.1 Motivación

La principal motivación de este proyecto es la de poder brindar cierta ayuda a los especialistas de la sanidad, facilitándoles en este caso el diagnóstico de enfermedades relacionadas con el movimiento en bebés.

Como se ha comentado previamente, los movimientos realizados por este tipo de pacientes son involuntarios y en muchos casos son simples reflejos que desaparecen a los pocos meses de vida. En muchas ocasiones, conocer cuanta presión ejercen estos pacientes en cada parte del cuerpo es vital para detectar posibles patologías o retrasos en el desarrollo motriz.

Por ello, creemos que el desarrollo de este proyecto puede ser de gran utilidad y ayuda a los especialistas, para que ellos puedan identificar y diagnosticar enfermedades relacionadas con el movimiento de forma más eficaz y sencilla.

Gracias a las esterillas de presión seremos capaces de observar como son los movimientos de los bebes y si estos se corresponden con los de un bebé sano.

1.2 Objetivos

El objetivo principal de este proyecto es el de crear un programa capaz de estudiar los movimientos realizados por bebés, los cuales serán obtenidos gracias a los mapas dinámicos de presión registrados por las esterillas de presión.

Como objetivos concretos tenemos, en primer lugar, el analizar el formato de la información de las esterillas, definiendo posteriormente los datos a generar a partir de dicha información. Una vez hecho esto, pasaremos a realizar el análisis propiamente dicho. Identificaremos las zonas registradas, calcularemos diferentes parámetros, como el número de zonas de apoyo, el área y centro de cada una de ellas o la presión ejercida

Gracias a esto, obtendremos patrones de movimiento, para poder observar cómo varía todo a lo largo de tiempo.

Por otro lado, este proyecto está orientado a la posterior visualización de estos datos para que sea más sencilla su interpretación por parte de los expertos. Esto se realizará en otro TFG distinto que correrá a cargo de otra persona.

1.3 Estructura de la memoria

Para finalizar con la introducción, pasaremos a explicar de cuántos capítulos constará el siguiente proyecto, así como, su principal contenido con una explicación breve y concisa.

Capítulo 1: Introducción. Se explicará concisamente el motivo principal por el cual se ha decidido realizar este proyecto, que objetivos se pretenden perseguir, así como una visión muy general del mismo.

Capítulo 2: Descripción de los materiales y medios usados. En este capítulo se describirán las herramientas usadas para la realización del proyecto, así como lo medios necesarios para llevarlo a cabo.

Capítulo 3: Desarrollo de la aplicación 1. Se procederá a detallar todos los pasos a seguir para el desarrollo de la aplicación 1, que se realizará con la primera esterilla adquirida, se especificará todo el código creado, así como los comandos y herramientas utilizadas para la creación del programa de análisis.

Capítulo 4: Desarrollo de la aplicación 2. Al igual que en el capítulo anterior detallarán los pasos a seguir para el desarrollo de la aplicación número 2, que en este caso implica a la segunda esterilla adquirida. Se especificará el código creado, así como los comandos y herramientas utilizadas para la creación del programa de análisis.

Capítulo 5: Conclusiones y líneas futuras: Para concluir se expondrán los resultados obtenidos, las conclusiones a las que se ha llegado una vez finalizado el proyecto, así como posibles líneas de continuación o ampliación de este proyecto.

Bibliografía y referencias: Se detallará las páginas, revistas y artículos empleados, al igual que las referencias precisas para la realización del proyecto. Se utilizará el estilo de citación del IEEE, tal y como se expone en la normativa sobre TFG's de la UAH.

2. Descripción de los materiales y medios usados

En el presente capítulo, detallaremos cuales son los materiales usados, así como los medios empleados.

Debemos tener en cuenta que vamos a trabajar con unas esterillas que generan datos que son exportados y procesados para obtener de ellos información de interés para los especialistas. Este procesado de la información podríamos realizarlo de varias formas en función de las herramientas y lenguajes escogidos.

2.1 Lenguaje empleado y editores de texto

Emplearemos Python como lenguaje de programación gracias a su sencillez y versatilidad. A lo largo de los años, Python se ha convertido en uno de los principales lenguajes para el desarrollo de aplicaciones capaces de analizar, tratar y procesar datos.

Si bien es cierto, existen otros lenguajes capaces de realizar funciones similares, el hecho de que Python sea un lenguaje cuyo aprendizaje es realmente sencillo y a su vez, que un fragmento de código se pueda reproducir en todas las plataformas, hace que este lenguaje se imponga en el campo del procesado de datos.

En cuanto a editores de texto podríamos usar muchos en este caso trabajaremos con Spyder y Sublime Tex.

Spyder es un entorno de código abierto gratuito escrito en Python, y diseñado especialmente para trabajar con Python. Cuenta con un editor con navegador de funciones y herramientas de análisis de código, una o varias consolas IPython en una interfaz gráfica de usuario, pudiendo ejecutar código, línea por línea. A su vez, dispone de un explorador de variables, gracias al cual podemos observar el valor de cada variable en todo momento, así como, modificarlo.

Sublime Text es un editor de texto diseñado para escribir código en multitud de lenguajes, es muy adecuado por su sencillez, ya que ofrece un espacio de trabajo que resalta el código con colores y muestra los errores.

Debido a que Sublime Text no cuenta con depurador usaremos en ocasiones el símbolo del sistema (cmd) para compilar y ejecutar código.

2.2. Librerías usadas en la aplicación

Deberemos hacer referencia a las librerías que vamos a utilizar. La nomenclatura que usaremos para ello será, *import* seguido de la librería que queremos. Para librerías con nombres largos o complicados podemos especificar el comando *as* seguido de un nombre más breve, tal y como se muestra en la figura 2.1, a continuación:

```
import json
import numpy as np
import seaborn as sb
from matplotlib import animation
from matplotlib import pyplot
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import pandas as pd
from pandas import ExcelWriter
```

Figura 2.1. Librerías

Librería matplotlib: es una librería de Python dedicada a crear gráficos de dos dimensiones. Se pueden crear entre otros gráficos: diagramas de barras, histogramas, diagramas de dispersión y para lo que lo vamos a usar nosotros, mapas de calor, como el de la figura 2.2.

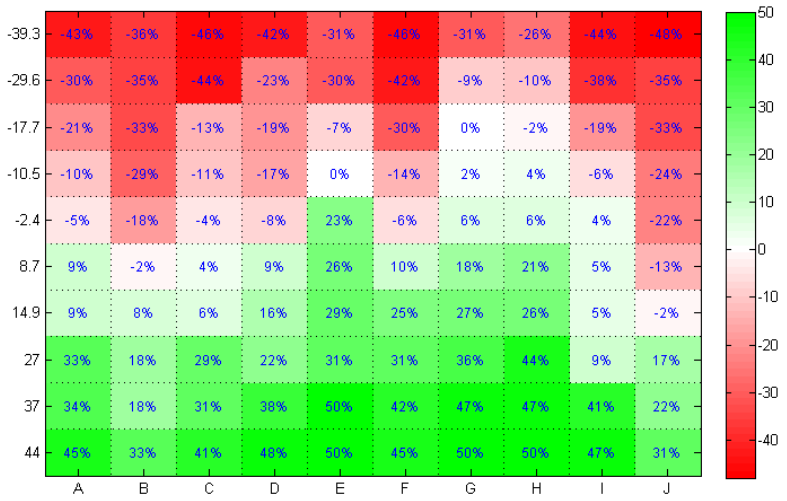


Figura 2.2. Mapa de calor

De esta librería importaremos los módulos:

- **animation**: clase base para realizar animaciones. Como veremos posteriormente, vamos a poder animar las matrices de datos, de tal forma, que podremos ver como varían los datos a lo largo del tiempo.
- **pyplot**: provisto de una serie de funciones que hacen que matplotlib realice aplicaciones similares a MATLAB.

Como podíamos ver en la figura 2.1, la forma de indicar en lenguaje Python, que queremos importar un módulo de una librería es poniendo *from* seguido de la librería, *import* y el módulo que queremos.

Librería JSON: JSON (Javascript Objet Notation) es un formato de lenguaje de texto, usado como alternativa a XML, escrito con notación de objetos JavaScript. Es una sintaxis

para el intercambio de datos. Python cuenta con un paquete JSON que se puede emplear para trabajar con datos JSON.

Librería NumPy: es una librería de Python dedicada al análisis de datos y el cálculo numérico, usada sobre todo para grandes volúmenes de datos. Incluye una clase *arrays* para representar conjuntos de datos del mismo tipo en varias dimensiones, proporcionando numerosas funciones para el manejo de estos.

Librería Seaborn: esta librería de Python facilita la creación de sofisticados gráficos. Basada en matplotlib provee una interfaz de alto nivel, la cual se estudia fácilmente.

Librería Pandas: es una librería de Python dedicada al análisis y control de estructuras de datos. Entre sus principales funciones destacan la de definir estructuras de datos fundamentadas en los arrays de la librería NumPy, añadiendo otras funcionalidades, y la de leer y escribir ficheros en diferentes formatos como Excel, CSV o bases de datos SQL.

2.3 Esterillas Fitness Mat Dev Kit 1.9 de Sensing.Tex

Una de las herramientas que vamos a usar para la detección de movimiento de los más pequeños son las esterillas Fitness Mat de Sensing.Tex. Estas esterillas son capaces de reconocer patrones de movimiento y de administrar los datos capturados con el objetivo de poder ofrecer soluciones inteligentes.

Las medidas de la esterilla que nosotros vamos a emplear son 630x 1710 mm, siendo el área de detección ligeramente menor, 560 x 1600 mm. Cuenta con 2240 sensores redondos de unos 10 mm^2 de superficie. La esterilla cuenta con una capa antideslizante y escalonada de neopreno.

Los sensores cuentan con un coeficiente de linealidad del 0,99. La linealidad de un sensor nos dice si este dispone de la misma exactitud para todos los tamaños de objetos que se pudieran medir. Tienen una precisión del 10% y un rango de presión de 20 – 1000mmHg (milímetro de mercurio).

En cuanto a las características hardware, disponen de un ADC de resolución 12 bits, una frecuencia de escaneo de 5-10 Hz, dependiendo de la comunicación, la compresión de los datos y el dispositivo receptor. Compatibles con Windows, Linux y Android entre otros. Tienen un estándar de conectividad BTClassic/BLE, WIFI y USB serial. Cuentan con una batería integrada de 1100 mAh.

En cuanto a las características software, la empresa de fabricación provee un software de demostración Windows, gracias al cual podemos tomar las diferentes muestras.

En el apartado de anexos, se expone un pequeño manual, sobre como instalar y configurar la esterilla y el software de demostración.

Debemos comentar, que en las primeras pruebas que se hicieron con esta esterilla se comprobó que esta carecía de la sensibilidad necesaria para detectar los movimientos

de los bebés con precisión y es por ello, que se ha trabajado con datos simulados del movimiento de un bebé.

2.4 Esterillas The Wellness Mat Dev Kit

La otra herramienta que vamos a emplear es la esterilla Wellness Mat Dev Kit. Esta esterilla ha hecho a medida, es decir, se ha hablado con el fabricante para conseguir mayor nivel de precisión para poder adaptarse a los pesos de los bebés.

Aunque como acabamos de comentar, esta otra esterilla dispone de la sensibilidad necesaria para reconocer correctamente los movimientos de un bebé, no se han podido realizar pruebas con bebés reales, ya que esta esterilla se recibió en el momento de finalizar este TFG; y como se verá más adelante a su vez, el análisis y procesado de los datos no se ha podido realizar tan exhaustivamente como con la anterior esterilla.

Una vez conocidas en profundidad las herramientas con las que vamos a trabajar, vamos a proceder a conocer cómo y dónde se almacenan los datos objeto de nuestro estudio.

3. Desarrollo de la aplicación 1

En este capítulo iremos desarrollando los pasos seguidos para la realización de la aplicación 1, que se llevará a cabo con los datos obtenidos de la primera esterilla.

En este capítulo, veremos cómo se ha realizado el análisis y procesado de los datos. A partir de donde obtenemos los datos y como manejamos estos para obtener de ellos la mayor información posible que sirva de utilidad a los especialistas.

Esta parte, será la que mayor volumen acaparará debido a su gran extensión, ya que se pretende detallar de manera concisa, pero clara, las diferentes partes del código y su finalidad principal.

3.1 Formato de los datos recopilados de las esterillas

El software de demostración nos permite tomar las muestras, grabando durante el tiempo que nosotros estimemos oportuno. Una vez grabadas las muestras, estas son almacenadas en un archivo JSON el cual tiene la estructura que se muestra en la figura 3.1:


```
{"softwareVersion":"1.9","calibrationConfiguration":{"calibrationTable":
{"conversionTable":null,"name":null},"customCalibrationTable":null,"calibrationUnits":0,"isVoltageSharingCompensationActivated":true,"isOutputImpedanceCompensationActi
vated":true,"isAutomaticBrokenRowsCompensationActivated":false,"isRoundMatrixFilterActivated":false,"isManualRowInterpolationActivated":
[false,false,false,false,false,false,false,false,false],"manualRowInterpolationValue":[0,0,0,0,0,0,0,0,0],"isManualColumnInterpolationActivated":
[false,false,false],"manualColumnInterpolationValue":[0,0,0]},"representationConfiguration":
{"threshold":0,"scaleSetting":0,"automaticScaleDefinition":0,"manualScaleDefinition":0,"manualScaleMax":50,"isContourActivated":false,"rotateDegrees":90,"flip":1,"plot
BackgroundColor":{"A":255,"B":255,"G":255,"R":255},"colorScale":{"Colors":[{"A":255,"B":139,"G":0,"R":0},{"A":255,"B":190,"G":113,"R":0},
{"A":255,"B":242,"G":226,"R":0},{"A":255,"B":170,"G":255,"R":84},{"A":255,"B":56,"G":255,"R":198},{"A":255,"B":0,"G":235,"R":255},{"A":255,"B":0,"G":195,"R":255},
{"A":255,"B":0,"G":146,"R":242},{"A":255,"B":0,"G":73,"R":190},
{"A":255,"B":0,"G":0,"R":139}]},"isDiscardBorderActivated":false,"numDiscardedBorderLines":1,"isShowNumbersActivated":true},"pressureData":[{"dateTime":"2021-04-
26T10:17:28.668628+02:00","pressureMatrix":
[[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0
.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0],
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0
.0],
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0
.0],
```


Figura 3.1. Archivo JSON

El archivo JSON lo podemos abrir con cualquier editor de texto y nos encontraremos algo como la imagen anterior. En el recuadro en amarillo podemos observar una serie de parámetros de configuración que pueden ser modificados desde el software de demostración, algunas de estas características tienen que ver con la calibración, la escala de colores, la interpolación de las columnas etc. No vamos a entrar en mayor detalle respecto a estas características ya que no nos van a hacer falta.

Cabe destacar que el archivo JSON que genera el software de prueba tiene un error, le faltan una serie de paréntesis y corchetes. Se contactó al fabricante para ver si podían solucionar este aspecto, pero no se obtuvo respuesta alguna, por lo que se ha creado un programa en Python capaz de solucionar este error.

La parte de la imagen que no aparece de color amarillo son las matrices que contienen propiamente los datos. Son matrices de dimensiones 28 x 80 (filas x columnas). Dentro de estas matrices se representa la presión ejercida sobre cada sensor.

Hemos de tener en cuenta que podemos ajustar de cierta forma la precisión de los sensores, ya que no es lo mismo que se apoye un bebé, que un adulto. Por ello, al trabajar con bebés esta precisión debe ser la mayor posible debido a su reducido peso.

Debido a esto en ocasiones, los sensores reconocen presión incluso en zonas en las que no se está ejerciendo presión alguna. A consecuencia de esto, como ya se verá posteriormente en el código, no tendremos en cuenta las dimensiones reales de estas matrices si no que trabajaremos con una zona más pequeña donde quepa el propio bebé, así evitaremos reconocer presiones en zonas que sabemos que por la longitud del bebé es imposible que aparezcan.

Tendremos tantas matrices como tiempo dure el experimento. El software toma muestras cada 0,0862 segundos aproximadamente. Cada matriz de datos viene precedida de la siguiente línea:

{"dateTime":"2021-04-26T10:17:28.7683288+02:00","pressureMatrix":

Que nos indica la fecha y hora a la que se tomó la muestra y que a continuación, viene la matriz de presión. Conocido todo esto podemos empezar a diseñar nuestros programas.

3.2 Procesamiento de la matriz de datos

Como se ha indicado previamente, usaremos Python como lenguaje principal. Todo archivo Python debe acabar en .py para que el sistema sea capaz de reconocerlo como tal. Para facilitar la compresión, iremos dividiendo la explicación del programa en partes, ejecutando cada una de estas partes una función diferente.

Para la realización del programa, se han creado unas muestras, ya que por falta de disponibilidad y situación de covid, al principio del trabajo no se pudieron recoger muestras con bebés reales. Estas muestras se han creado suponiendo las zonas básicas de apoyo de un bebé como son cabeza, escápulas, parte baja de la espalda o zona de los riñones y los pies. En esta esterilla no vamos a tener en cuenta las manos del bebé ya que, debido a la precisión, estas no llegan a apreciarse.

3.2.1 Obtención de la matriz de datos

Las numerosas variables que se van a emplear se irán explicando a lo largo del desarrollo del proyecto, cabe destacar que en Python no es necesario inicializar las variables, como en otros lenguajes de programación.

Lo que vamos a hacer en primer lugar es, abrir el fichero JSON ya modificado, tal y como se ve en la figura 3.2, recordemos que el fichero original, generado por el software del fabricante, contenía un error, el cual solucionamos con un fichero.

```
json_fname = "C:\\Users\\Luyma\\OneDrive - Universidad de Alcala\\extraeDatos_v0\\muestraBMOD.json"
with open(json_fname) as file:
        data = json.load(file)
```

Figura 3.2. Abrir fichero JSON

Lo que hacemos es guardar la ruta del archivo en una variable, para trabajar más cómodamente, abrimos el archivo y almacenamos su contenido en la variable *data*.

La variable data contendrá algo muy similar a lo que se muestra en la figura 2.1, de este mismo capítulo, una serie de propiedades las cuales no vamos a usar, y todas las matrices del experimento. A continuación, veremos cómo acceder a ellas.

Ya se especificó con anterioridad, que era posible que la esterilla FitnessMat reconociera en algún momento algún valor por defecto, es decir, que reconociera una zona de presión inexistente. Por ello, pondremos un límite para que esos valores no sean tomados en cuenta.

Como podemos ver en la figura 3.3, establecemos el límite inferior en 100, de tal forma que todos los valores inferiores a 100, no serán tenidos en cuenta y su valor pasará a valer 0. Por otra parte, descartaremos todas aquellas matrices cuyos valores sean todos 0. Esto lo hacemos, ya que se puede dar el caso en el que empecemos a tomar las muestras sin haber colocado al bebé en la esterilla, o que, por el contrario, antes de finalizar se retire al bebé antes de tiempo.

```
limite_inferior = 100
paquete_datos = []
for i in range (0, len(data["pressureData"])):
    datos = np.array(data["pressureData"][i]["pressureMatrix"])

    # Pone a 0 los valores de la matriz que sean menores que el limite
    datos[datos <= limite_inferior] = 0
    # Nos quedamos con las matrices con datos distintos de cero y su fecha
    x = datos.sum()
    if x > 0:
        paquete_datos.append(data["pressureData"][i])
```

Figura 3.3. Límite

Hecho todo esto, ya tendríamos almacenadas en la variable *paquete_datos* todas las matrices con datos.

3.2.2 Localización de las diferentes zonas de presión

Una de las primeras cosas que debemos realizar es identificar las diferentes zonas registradas. Como ya se comentó en el apartado anterior, es posible que debido a la precisión que utilizamos para reconocer las presiones pequeñas de los bebés, aparezcan zonas que registren cierta presión, bastante baja, en lugares donde el bebé no realiza ningún apoyo.

Es por ello que, debido a la longitud de la esterilla 160 cm y teniendo en cuenta, que la longitud media de los bebés de entre 0 y 3 meses es de 55 cm a 64 cm, aproximadamente, no tendremos en cuenta toda la superficie de la esterilla, lo que se traduce en que no tendremos en cuenta las matrices completas.

Las dimensiones de las matrices que genera el software del fabricante son 28 x 80, en nuestro caso, solo nos quedaremos con los datos comprendidos entre 4 y 22 para las filas y entre 10 y 70 para las columnas, evitando así zonas de error.

El código se muestra en las figuras 3.4 y 3.5:

```
#Obtenemos datos de una matriz en un instante de tiempo
for i in range (0,len(paquete_datos)):
    datos = np.array(paquete_datos[i]["pressureMatrix"])
    zona_x=0
    aux=0
    for p in range (4,23):
        for q in range (10,70):
            if (datos[p][q] > 149 ):
                if (len(zonas)>0):
                    for z in range (0,len(zonas)):
                        punto=np.array(zonas[z])

                        if(int(punto[2])>int(aux)):
                            aux=int(punto[2])

                        if (((p-3) < punto[0] < (p+3)) and ((q-3) < punto[1] < (q+3))):
                            zona[0]=p
                            zona[1]=q
                            zona[2]=int(punto[2])
                            zona[3]=datos[p][q]
                            zonas.append([zona[0],zona[1],punto[2],zona[3]])
                            zone=1
                            break
                    if(zone==0):
                        zona[0]=p
                        zona[1]=q
                        zona[2]=int(aux+1)
                        zona[3]=datos[p][q]
                        zonas.append([zona[0],zona[1],zona[2],zona[3]])
```

Figura 3.4. Identificar zonas (1)

```
                if (len(zonas)==0):
                    zona[0]=p
                    zona[1]=q
                    zona[2]=0
                    zona[3]=datos[p][q]
                    zonas.append([zona[0],zona[1],zona[2],zona[3]])
            q=q+1
            zone=0
        p=p+1
    paquete_zonas.append(zonas)
    zonas=[]
```

Figura 3.5. Identificar zonas (2)

Para empezar, iremos recorriendo el array que contiene todas las matrices del experimento. De cada matriz nos quedaremos solo con la parte que contiene los datos de los sensores, para ello, tendremos que especificar que esta información se encuentra donde pone “pressureMatrix”. (Recordar la Figura 3.1).

Cuando ya tenemos esta matriz almacenada, iremos recorriéndola elemento a elemento, teniendo en cuenta las restricciones de las que ya hablamos previamente, respecto a las dimensiones de la misma.

La variable *zonas* indicará cuantas zonas hay en cada matriz. Para diferenciar una zona de otra lo que haremos es imponer como norma que si el punto que estamos analizando está a más de cuatro posiciones tanto a derecha como a izquierda como hacia arriba o hacia bajo de un punto ya registrado no pertenecerá a esa zona.

Compararemos el punto “actual” con los ya registrados, si tras compararlos todos, el punto actual no pertenece a ninguna de las zonas, crearemos una nueva zona y la variable *zonas* aumentará en 1.

Como podemos ver en la figura 3.4, lo primero que comprobamos es el tamaño de la variable zonas, la primera vez que se ejecute el bucle, la variable zonas tendrá dimensión 0, y por ello, añadiremos una zona. La variable *zona* es un array bidimensional de 4 columnas en el cual almacenaremos las coordenadas ‘x’ e ‘y’ de cada punto, la zona a la que va a pertenecer dicho punto y el valor registrado en ese punto, valor de presión.

La segunda vez que se entre en el bucle, la variable *zonas* ya no tendrá dimensión 0, entonces, iremos almacenando en la variable *punto*, los diferentes puntos ya registrados. Debemos tener en cuenta, que los puntos se irán almacenando de manera ordenada respecto a su posición en la matriz principal, pero que esto no ocurrirá para las zonas, por ello, guardaremos en la variable aux el valor de la zona de mayor número, para asegurarnos que siempre que queramos añadir una nueva zona esta, sea una más que la de mayor número.

Como se observa en la figura 3.5, comparamos la posición x, en este caso hace referencia a ella la variable *p*, del punto actual con la coordenada x de todos los puntos almacenados, sumando y restando tres, para comprobar si el punto pertenece a la misma zona que con el que lo estamos comparando. En la misma condición del *if*, hacemos lo mismo para la coordenada y, a la cual hace referencia la variable *q*.

Si el punto pertenece a alguna zona ya existente, almacenamos los valores que identifican al punto, en el array *zona[ ]* y posteriormente, con el comando *append* lo guardaremos en la variable *zonas*, que contendrá todos los puntos significativos de cada matriz. Por último, pondremos la variable *zone* a 1, de esta forma si tras comparar todos los puntos ya registrados, ninguno coincide con el punto que estamos comparando, la variable *zone* valdrá cero y entonces tendremos que añadir una nueva zona.

Debemos poner a cero la variable zone antes de analizar un nuevo punto. Una vez que hemos recorrido una matriz, añadimos la variable *zonas* a *paquete*_zonas que contendrá tantas matrices *zonas* como matrices correctas tenga el experimento.

Es importante que antes de analizar una nueva matriz vaciemos la variable *zonas*, para que no contenga viejas zonas. Para ver cómo quedaría todo, si imprimimos el primer array del matriz paquete_*zonas* con el siguiente comando:

print(paquete_zonas[0])

Veríamos lo que se muestra en la figura 3.6, que como dijimos anteriormente indica [x, y, zona, valor].

```
[[10, 49, 0, 885.0], [10, 50, 0.0, 911.0], [11, 49, 0.0, 289.0], [11, 50,
0.0, 384.0], [12, 20, 1, 678.0], [12, 36, 2, 758.0], [13, 35, 2.0, 830.0],
[14, 35, 2.0, 466.0], [16, 48, 3, 1169.0], [16, 49, 3.0, 443.0], [17, 48,
3.0, 591.0], [17, 49, 3.0, 705.0]]
```

Figura 3.6. paquete_zonas[0]

3.2.3 Identificación de las zonas de presión

Previamente, hemos localizado las diferentes zonas que aparecían, identificándolas como zona 0, zona 1 y así sucesivamente. Para los expertos es de vital importancia saber a qué parte del cuerpo pertenece cada una de las zonas de apoyo.

Según ellos, cuando un bebé está acostado boca arriba, sus principales puntos de apoyo son, la cabeza en primer lugar, la cual rotará de izquierda a derecha en función de los estímulos que perciba el bebé. El bebé no será capaz de levantar la cabeza estando boca arriba hasta el cuarto mes de vida. En segundo lugar, tendremos la espalda o más concretamente, las escapulas, posteriormente la pelvis y, por último, los talones. Lo más normal, es que, con esos meses de vida, un bebé no tenga los talones permanentemente apoyados si no que vaya dando pequeñas patadas y estén apoyados intermitentemente. Por ello, es importante que la primera muestra que tomemos sea aquella en la cual todas estas partes estén apoyadas teniendo contacto con la esterilla.

Para identificar cada zona en primero se pedirá a quienes tomen estas muestras, que en primer lugar se deberá colocar al bebé en la esterilla de forma que todas las partes principales del cuerpo permanezcan apoyadas. De esta forma, nos será mucho más fácil poder identificar las partes del cuerpo a lo largo del experimento.

Entonces lo que haremos es coger la primera matriz con datos distintos de cero y calcular el centro de cada zona que pasaremos a explicar a continuación.

Debemos tener en cuenta la diferencia entre calcular el centro de una zona y el centro de masas de esa misma zona, ya que no es lo mismo. Para calcular el centro no hace falta que tengamos en cuenta el valor de presión ejercido, tendremos que contar cuantos puntos hay en cada zona, sumar por un lado las coordenadas 'x' de cada punto perteneciente a una zona y, por otro lado, sumar las coordenadas 'y' de esos mismos puntos. Después solo tendremos que dividir estos valores por el número total de puntos por cada zona y lo tendremos. Vamos a explicar todo esto un poco más en detalle, ayudándonos del código.

Como podemos observar en la figura 2.9, lo primero que hacemos es guardar el contenido de la primera matriz de *paquete_zonas* en una variable para trabajar más cómodamente. Iremos recorriendo esa matriz elemento a elemento. En la variable *pzona* que será un array unidimensional de tamaño igual al número de zonas de esta primera matriz, guardaremos en la posición de cada zona, el número de puntos que hay en ella, es decir, en *pzona[0]* almacenaremos cuantos puntos conforman la zona 0 y así sucesivamente.

En primer lugar, lo que hacemos es comparar el tamaño de la variable *pzona* con la zona a la que pertenece cada elemento. Si el tamaño de *pzona* es mayor que la zona del punto, entonces, sumaremos 1 en la posición de esa misma zona. Si por el contrario es menor, eso significa que debemos añadir un elemento a la matriz *pzonas*, para poder almacenar la suma de cuantos puntos pertenecen a esa zona.

La variable *puntos_zonas* previamente declarada, variable bidimensional, formada por 3 columnas que indicarán: la suma de las coordenadas 'x', la suma de las coordenadas 'y' y la suma de presiones (aunque para el cálculo de estos centros no sea necesario). Y tantas filas como zonas haya.

Ahora compararemos el tamaño de la variable *puntos_zonas*, restándole 1, con el número de la zona a la que pertenece un punto. Si es igual o mayor significará que previamente ya se ha agregado otro punto a la matriz y, por tanto, solo tendremos que sumar las coordenadas y el valor de presión a lo guardado previamente. Si es menos entonces, tendremos que añadir una nueva fila a *puntos_zonas* que indicará que el punto que estamos analizando pertenece a una zona distinta a las previas añadidas.

Una vez analizados todos los puntos de *matriz_zona*, solo tendremos que dividir la suma de las coordenadas 'x' e 'y' por el número de puntos que componen cada zona. Estos datos los guardaremos en la matriz *centro_cuerpo*. Todo lo explicado previamente, corresponde al código de la figura 3.7.

```
########################## Identificar zonas cuerpo
matriz_zona=np.array(paquete_zonas[0])
zone=0
pzona=[]#matriz para contar cuantos sensores hay en cada zona
for t in range(0,len(matriz_zona)):
    #leemos un punto de la matriz x,y,zona,valor
    punto=np.array(matriz_zona[t])
    if(len(pzona) > int(punto[2])):
        pzona[int(punto[2])]=pzona[int(punto[2])]+1
    else:
       zone=zone+1
       pzona.append(1)
    if((len(puntos_zonas)-1)==punto[2] or (len(puntos_zonas)-1)>punto[2]):
        puntos_zonas[int(punto[2])][0]=puntos_zonas[int(punto[2])][0]+punto[0]
        puntos_zonas[int(punto[2])][1]=puntos_zonas[int(punto[2])][1]+punto[1]
        puntos_zonas[int(punto[2])][2]=puntos_zonas[int(punto[2])][2]+punto[3]
    else:
        puntos_zonas.append([int(punto[0]),int(punto[1]),int(punto[3])])
for a in range(0,(len(pzona))):
    centro=np.array(puntos_zonas[a])
    #Suma coordenadas X dividido nº sensores de 1 zona
    centro_z[0]=(centro[0]/(int(pzona[a])))
    centro_z[1]=(centro[1]/(int(pzona[a])))
    centro_cuerpo.append([centro_z[0],centro_z[1]])
```

Figura 3.7. Identificar zonas (1)

Una vez calculados los centros, nos falta identificar a que parte del cuerpo pertenece cada zona. Esto podría parecer una de las partes más difíciles de realizar, pero es tan sencillo como tener en cuenta que a los bebés siempre los vamos a situar de la misma forma con la cabeza a la izquierda, de tal forma que identificar las zonas será tan sencillo como ver que partes están más a la izquierda o a la derecha de la esterilla. Así el centro que se encuentre más a la izquierda, es decir, el que tenga menor valor de coordenada 'y', pertenecerá a la zona de la cabeza, la siguiente zona será las escapulas, luego la pelvis o parte baja de la espalda. Por último, tendremos los pies, los cuales diferenciaremos en función de la coordenada 'x', la zona cuya coordenada 'x' sea menor, corresponderá al pie derecho y la otra al pie izquierdo.

Como podemos ver en el código de la figura 3.8, primero ordenamos las zonas de izquierda a derecha, por eso siempre comparamos la segunda columna de cada matriz, y las almacenamos en la variable *cuerpo*, array de 5 elementos. Para finalizar comparamos las coordenadas 'y' de las dos últimas zonas, la variable cuerpo contiene de forma ordenada las diferentes partes del cuerpo, los pies serán las dos últimas zonas ya que son las que más a la derecha estarán; y las ordenaremos. Finalmente, imprimiremos los resultados.

Resaltar en última instancia, que debido a que las primeras muestras con las que se está trabajando no son de bebés reales si no simuladas, es decir, se han usado diferentes pesos para simular los movimientos de las diferentes partes del cuerpo; la matriz que

utilizamos para identificar las diferentes zonas no es la 0, si no la 74, que corresponde a una matriz en la que aparecen las cinco zonas a la vez.

```
parte=0
for j in range(0,5):
    for i in range(0,(len(centro_cuerpo))):
        if(centro_cuerpo[j][1]>centro_cuerpo[i][1]):
            parte=parte+1
            print(parte)
    cuerpo[parte]=j
    parte=0

if(centro_cuerpo[cuerpo[3]]>centro_cuerpo[cuerpo[4]]):
    auxiliar=0
else:
    auxiliar=cuerpo[4]
    cuerpo[4]=cuerpo[3]
    cuerpo[3]=auxiliar
print("Cabeza:",cuerpo[0],"Escapulas:",cuerpo[1],"Parte baja espalda:",cuerpo[2],"Pie Izquierdo:",cuerpo[3],"Pie Derecho:",cuerpo[4])
##############################
```

Figura 3.8. Identificación zonas (2)

3.2.4 Eliminación de matrices redundantes

Debido a que vamos a imponer como condición que en primer lugar se coloque al bebé en la esterilla con la cabeza a la izquierda y asegurándonos que las partes principales de su cuerpo tengan contacto con la esterilla, es posible, que al analizarlas todas veamos que al principio algunas están repetidas, es por ello, que sería interesante eliminar este tipo de muestras ya que no nos aportarían gran información.

Para ello, como primera aproximación podríamos pensar que sería tan sencillo como comparar las matrices almacenadas en la variable *paquete_zonas*, que recordemos contiene matrices con los diferentes puntos que registran presión, pero debido a la gran precisión que tenemos es muy difícil que los valores de presión en los puntos sean exactamente los mismos, así pues, un código como el de la figura 3.9, no nos serviría.

```
#############Eliminar matrices repetidas
print(len(paquete_zonas))
for i in range (0,(len(paquete_zonas)-1)):
    comparar=np.array_equal(paquete_zonas[i],paquete_zonas[i+1])
    if(comparar == "False"):
        paquete_zonas.remove[i]
print(len(paquete_zonas))
```

Figura 3.9. Eliminación matrices repetidas (error)

Teniendo en cuenta lo dicho previamente, lo adecuado sería comparar el tamaño de las matrices, si estas tienen el mismo tamaño significa que tienen el mismo número de puntos, entonces pasaríamos a comparar todos los puntos de cada una de ellas.

Como es lógico las coordenadas 'x' e 'y' deberán coincidir, pero para el valor de presiones es distinto, ya que es normal que, aunque en dos instantes distintos se esté ejerciendo la misma fuerza, los valores de presión varíen ligeramente debido a la precisión de los mismos. Por ello, asumiremos que si la diferencia de estos valores es inferior a 100 ambos puntos son el mismo.

También podríamos pensar, que lo adecuado sería realizar esta operación en primer lugar, es decir, antes de calcular los puntos y zonas, para ahorrar tiempo. Pero, si lo pensamos detenidamente, si decidiéramos comparar las matrices completas tendríamos que recorrer bastantes más elementos, ya que recordemos que la longitud de estas es 18 filas por 60 columnas (matrices reducidas para evitar interferencias), es por esto que es preferible comparar las matrices que contienen exclusivamente los puntos y zonas, es decir, las almacenadas en *paquete_zonas*.

Tal y como podemos ver en la figura 3.10, lo primero que hacemos es comparar las longitudes de dos matrices consecutivas, si tienen el mismo tamaño entonces, pasaremos a comparar los diferentes puntos de ambas matrices.

```
############Eliminar matrices repetidas
print("Longitud matriz paquete_zonas:",len(paquete_zonas))
puntos_igual=0
m_rep=[]
for i in range (0,(len(paquete_zonas)-1)):
    #Comparamos tamaños
    if (len(paquete_zonas[i])==len(paquete_zonas[i+1])):
        for j in range(0,len(paquete_zonas[i])):
            #Comparamos coordenadas x
            if(paquete_zonas[i][j][0]== paquete_zonas[i][j][0]):
                #Comparamos coordenas y
                if(paquete_zonas[i][j][1]== paquete_zonas[i][j][1]):
                    #Comparamos diferencia de valores
                    if(abs(paquete_zonas[i][j][3] - paquete_zonas[i+1][j][3]) < 100):
                        puntos_igual=puntos_igual+1
        if(puntos_igual==len(paquete_zonas[i])):
            puntos_igual=0
            m_rep.append(i)
print("Matrices 'repetidas:",len(m_rep))
```

Figura 3.10. Eliminar matrices repetidas (1)

Primero nos fijamos si ambos tienen la misma coordenada 'x', si es así, pasamos a comprobar si sus coordenadas 'y' son las mismas y, por último, si todo esto se cumple, comprobaremos si la diferencia de ambos valores de presión es inferior a 100. Usamos el comando *abs* para expresar valor absoluto. Si es así, sumamos 1 a la variable *puntos_igua*l, que llevará el recuento de cuantos puntos son iguales.

Recorrida toda la matriz y comparados todos los puntos, solo nos queda comprobar si el valor de la variable puntos_igual es igual al tamaño de ambas matrices, ya que esto indicará que todos los puntos de amabas matrices son iguales, y, por tanto, esa matriz deberá ser eliminada posteriormente.

Para saber posteriormente, cuales son aquellas matrices que debemos eliminar, usaremos la matriz *m_rep* (de matrices repetidas) para almacenar estos datos, de tal forma, que cada elemento de la matriz contendrá el número de aquella matriz a borrar.

Una vez localizadas todas estas matrices procederemos a eliminarlas, con el código que se muestra en la figura 3.11. Nos ayudaremos de dos bucles for, para ir comparando si el número de la matriz a borrar coincide con el número de la matriz actual. Tendremos que recorrer toda la matriz *m_rep*, hasta encontrar un valor que coincida con el valor de i, que nos indica la matriz "actual" o hasta recorrerlo entero.

Si tras recorrer entera la matriz *m_rep* encontramos una coincidencia, pondremos a 1 la variable *igual*. Posteriormente comprobaremos si *igual* es 1, si lo es, la pondremos a 0 y si es igual a 0, entonces eso indicará que no se encontró ninguna coincidencia y que esa matriz no es similar a otra y que, por tanto, si podremos guardarla.

Finalmente, vaciaremos la matriz *paquete_zonas* para guardar aquellas matrices no repetidas. (Hacemos esto para no trastocar el resto de código ya escrito y seguir usando el mismo nombre de variable). También haremos lo mismo con la variable *paquete_datos*, ya que como veremos a continuación, necesitaremos tener esta variable actualizada con las matrices no repetidas.

```
paquete_zonas1=[]
paquete_datos1=[]
j=0
for i in range(0,len(paquete_zonas)):
    for j in range(0,len(m_rep)):
        if(i==m_rep[j]):
            igual=1
            break
    if(igual==1):
        igual=0
    else:
        paquete_zonas1.append(paquete_zonas[i])
        paquete_datos1.append(data["pressureData"][i])
paquete_zonas=[]
paquete_datos=[]
paquete_datos=paquete_datos1
paquete_zonas=paquete_zonas1
print("Longitud matriz paquete_zonas sin matrices repetidas:",len(paquete_zonas))
```

Figura 3.11. Eliminar matrices repetidas (2)

Si ejecutáramos todo el código anterior (figuras 3.10 y 3.11) recibiríamos algo como lo mostrado en la figura 3.12 por pantalla, y de esta forma comprobamos que este código funciona correctamente.

```
Longitud matriz paquete_zonas: 241
Matrices 'repetidas: 60
Longitud matriz paquete_zonas sin matrices repetidas: 181
```

Figura 3.12. Comprobación código

3.2.5 Cálculo de la duración de la actividad de toma de datos

Calcularemos cuanto tiempo ha durado la toma de todas las muestras y también cuanto ha durado la toma de las muestras que realmente vamos a utilizar, es decir, las muestras válidas. Esto lo hacemos para posteriormente calcular la velocidad que presenta cada zona. Para ello, obtendremos la hora en la que se tomaron la primera y la última muestra.

Como podemos ver en la figura 3.13, primero calcularemos el tiempo de solo las muestras que vamos a usar. Por ello, obtenemos el primer valor de la primera matriz de la variable *paquete_datos*, fijarse que después de indicar el número de la matriz escribimos *"datetime"* para quedarnos solo con esa información de toda la matriz.

```
##################Calcular tiempo experimento
hora1=paquete_datos[0]["dateTime"]
hora_1=list(hora1)
minutos=[]
segundos=[]
for h in range(0,len(hora_1)):
    if (hora_1[h] == ":"):
        for m in range(0,2):
            minutos.append(hora_1[h+1])
        for o in range(0,9):
            segundos.append(hora_1[h+4+o])
        break
    else:
        a=0
segundosini=''.join(segundos)
minini=''.join(minutos)
hora1=paquete_datos[len(paquete_datos)-1]["dateTime"]
hora_1=list(hora1)
hora=[]
minutos=[]
for h in range(0,len(hora_1)):
    if (hora_1[h] == ":"):
        for m in range(0,2):
            minutos.append(hora_1[h+1])
        for o in range(0,9):
            segundos.append(hora_1[h+4+o])
        break
    else:
        a=0
segundosfin=''.join(segundos)
minfin=''.join(minutos)
```

Figura 3.13. Tiempo experimento

Recordemos que el formato de la fecha y hora es el siguiente:

*"dateTime":"2021-04-26T**10:17:28.7683288**+02:00"*

De toda esta cadena de caracteres, necesitamos solo la parte en negrita que nos indica la hora exacta a la que se tomó la muestra de esa matriz. Guardados los datos en la variable *hora1* pasaremos esta cadena de caracteres al formato lista, para poder recorrer esta cadena carácter a carácter y quedarnos solo con la información necesaria.

variable *hora1* = 2021-04-26T10:17:28.668628+02:00

variable *hora_1* = ['2', '0', '2', '1', '-', '0', '4', '-', '2', '6', 'T', '1', '0', ':', '1', '7', ':', '2', '8', '.', '6', '6', '8', '6', '2', '8', '+', '0', '2', ':', '0', '0']

Recorremos el array hora_1 carácter a carácter hasta localizar el carácter ":". Si nos fijamos en el formato de la hora vemos, que primero aparecen las horas, luego los minutos y por último los segundos, suponemos que las muestras que se tomen nunca durarán una hora, por ello, solo nos quedamos con la parte de los minutos y los segundos, las cuales guardaremos en las variables *minutos* y *segundos* respectivamente.

En las variables *minutos* y *segundos* seguimos teniendo los caracteres en formato lista, entonces debemos hacer la operación contraria a listar una cadena de caracteres, esto lo podemos hacer con el comando *join***,** al cual le debemos indicar a partir de que queremos unir, es decir, en nuestro caso queremos que todos los caracteres separados por ' ' se unan formando una única cadena de caracteres, que guardaremos en las variables *minini y segundosini*.

Posteriormente, repetiremos este mismo proceso, para la hora final, en este caso, nos quedaremos con la última matriz de la variable *paquete_datos*.

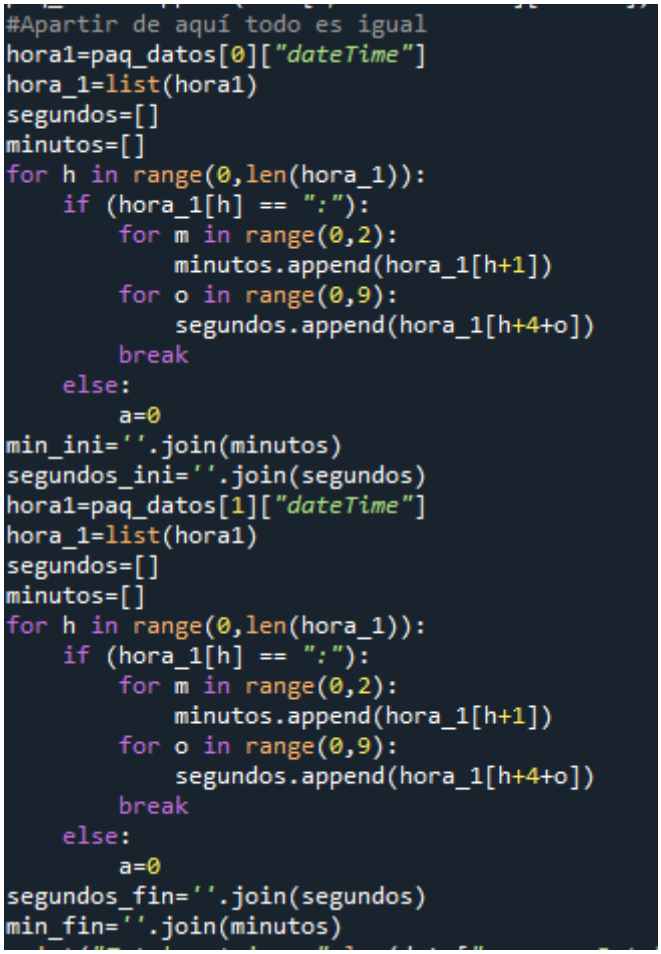
Con esto, ya podríamos calcular la duración de la parte buena de las muestras, es decir, debemos recordar que en la variable *paquete_datos* guardamos todas las matrices distantes de cero y las cuales pasamos por un "filtro" para eliminar posibles errores, pero también queremos calcular el tiempo total.

Para ello, solo tenemos que quedarnos con la primera y la última muestra tal como se muestra en la figura 3.14, la variable *maxi* representa el tamaño total de la variable *data*.

```
#######Calcular tiempo experimento total
paq_datos=[]
#Nos quedamos con la primera y la última matriz
paq_datos.append(data["pressureData"][0])
paq_datos.append(data["pressureData"][maxi-1])
#A partir de aquí todo es igual
```

Figura 3.14. Matrices primera y última

Ahora solo tenemos que repetir los mismos pasos de antes para calcular el tiempo, como vemos en la figura 3.15:



```
#Apartir de aquí todo es igual
hora1=paq_datos[0]["dateTime"]
hora_1=list(hora1)
segundos=[]
minutos=[]
for h in range(0,len(hora_1)):
    if (hora_1[h] == ":"):
        for m in range(0,2):
            minutos.append(hora_1[h+1])
        for o in range(0,9):
            segundos.append(hora_1[h+4+o])
        break
    else:
        a=0
min_ini=''.join(minutos)
segundos_ini=''.join(segundos)
hora1=paq_datos[1]["dateTime"]
hora_1=list(hora1)
segundos=[]
minutos=[]
for h in range(0,len(hora_1)):
    if (hora_1[h] == ":"):
        for m in range(0,2):
            minutos.append(hora_1[h+1])
        for o in range(0,9):
            segundos.append(hora_1[h+4+o])
        break
    else:
        a=0
segundos_fin=''.join(segundos)
min_fin=''.join(minutos)
```

Figura 3.15. Tiempo total del experimento

Por último, solo nos quedaría mostrar los resultados obtenidos. Pasaremos las variables a tipo 'float' para poder restarlas y obtener datos más precisos. Por otro lado, también mostraremos el número total de muestras recogidas y el número de muestras que en verdad vamos a utilizar para el análisis de los datos. Este código queda reflejado en la figura 3.16.

```
print("Total matrices:",len(data["pressureData"]), "Matrices OK: ",len(paquete_datos))
print("El experimento total ha durado :",float(min_fin)-float(min_ini), "minutos",
      float(segundos_fin)-float(segundos_ini),"segundos El experimento ha durado :",
      float(minfin)-float(minini),"minutos", float(segundosfin)-float(segundosini),"segundos")
```

Figura 3.16. Mostrar los resultados

Lo que se mostraría sería lo observado en la figura 3.17:

```
Total matrices: 241 Matrices OK:  181
El experimento ha durado : 0.0 minutos 15.615560000000002 segundos
El experimento total ha durado : 0.0 minutos 20.187898000000004 segundos
```

Figura 3.17. Resultados

3.2.6 Función comparar

A partir de ahora, para evitar numerosas líneas de código y bucles for, vamos a definir una función a la cual llamaremos tantas veces como matrices tenga el experimento, de esta forma dentro de esta función realizaremos varias tareas como el cálculo del área de cada zona, determinar en qué zona se está ejerciendo mayor o menor presión, cálculo de los centros normales y de masa de cada zona, etc. Llamaremos a la función tal como se muestra en la figura 3.18:

```
for i in range(0,len(paquete_datos)):
    comparar(i)
```

Figura 3.18. Llamada a la función comparar

Y la definiremos como en la figura 3.19:

```
def comparar(c):
```

Figura 3.19. Definición función comparar

3.2.6.1 Determinación del lado predominante

Una de las cosas más difíciles de identificar para los profesionales es conocer que parte del cuerpo ejerce mayor presión, si la parte derecha o la izquierda, ya que esto podría indicar ciertas deficiencias en el desarrollo de los pacientes. Como se comentó en la introducción, observar que predomina considerablemente una zona podría indicar, por ejemplo, cierta pérdida de audición, ya que el bebé al no escuchar correctamente no es capaz de reaccionar a los estímulos que recibe de esta zona.

Para facilitar los cálculos, situaremos en la mitad de la esterilla una línea con algún tipo de cinta, de forma que los expertos a la hora de tomar las muestras deberán situar al paciente en la mitad de esa línea, tal y como se muestra en la figura 3.20.

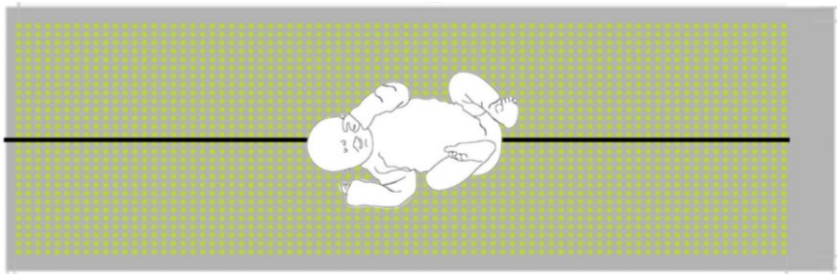


Figura 3.20. Línea divisoria.

Debido a que redujimos el tamaño de las esterillas, la mitad de estás sería, teniendo en cuenta esto último, la fila 13 (nos quedamos solo con las filas que van desde la 4 a la 22).

Para saber si un punto está por encima o por debajo de la línea nos fijaremos en su coordenada 'y'. Tendremos una variable de nombre *superior* que será una matriz de 3 elementos que representará cuantos puntos están en la parte superior, cuantos en la parte inferior y, por último, cuantos están justo en el medio.

Como podemos ver en el código de la figura 3.21, iremos recorriendo elemento a elemento una matriz de *paquete_zonas*, recordemos que todo esto lo estamos haciendo dentro de la función comparar y por ello, cada vez que llamemos a esta función recorreremos una matriz diferente.

El código es relativamente sencillo, comprobamos si la coordenada 'y' de cada punto es mayor, igual o menor a 13 y sumamos 1 en cada variable en función de este resultado.

Posteriormente, comprobaremos cual es la zona que predomina, y sumaremos 1 al elemento de la variable *superior* que corresponda. De esta forma, tras comprobar todas las matrices del experimento, sabremos gracias a esta variable en cuantas matrices predomina cada zona. Es posible que en algún momento ambas zonas estén equilibradas, es decir, haya el mismo número de puntos por encima y por debajo de 13, en ese caso pondremos a 1 la variable *empate*.

```
matriz_zona=np.array(paquete_zonas[c])
#Saber que lado predomina
for t in range(0,len(matriz_zona)):
    punto=np.array(matriz_zona[t])#leemos un punto de la matriz x,y,zona,valor
    if (punto[0]>13):#punto parte superior (nos fijamos en la coordenada x)
        arriba=arriba+1
    if (punto[0]<13):
        abajo=abajo+1
    if (punto[0]==13):
       mitad=mitad+1
if (mitad >arriba and mitad >abajo): #La mayoría de puntos queda en la mitad
    superior[2]=superior[2]+1
if(arriba>abajo): #La mayoría de puntos están en la parte superior
    superior[0]=superior[0]+1
if(arriba == abajo):
    empate=1
else:
    superior[1]=superior[1]+1
```

Figura 3.21. Código lado predominante

Ya fuera de la función comparar solo nos quedará echar un vistazo a los resultados obtenido y mostrarlos, así como se ve en la figura 3.22.

```
print("De las ", len(paquete_zonas), "muestras tomadas en ",predomina,
      "predomina la parte superior. En ", superior[2],"predomina la línea divisoria.
      "En ",superior[1], "predomina la zona inferior")
```

Figura 3.22. Resultados zona predominante

3.2.6.2 Cálculo de los centroides y centros de masa

En el apartado 3.2.3 Identificación de las zonas de presión ya se explicó como calcular el centro de cada zona y dijimos que, para calcular el centro de masa de cada zona, a su vez teníamos que tener en cuenta el valor de presión ejercido en cada punto.

Como podemos ver en la figura 3.23, a la hora de calcular el centro de masa de una zona, debemos sumar todas las coordenadas 'x' multiplicadas cada una por el valor de presión de ese punto y lo mismo para la coordenada 'y'. Para calcular el centro normal solo teníamos que sumar las coordenadas 'x' e 'y', por separado, de cada punto.

```
#para calcular el centro de cada zona
for t in range(0,len(matriz_zona)):
    if(len(pzona) > int(punto[2])):
        pzona[int(punto[2])]=pzona[int(punto[2])]+1
    else:
       zone=zone+1
       pzona.append(1)
    if((len(puntos_zonas)-1)==punto[2] or (len(puntos_zonas)-1)>punto[2]):
        puntos_zonas[int(punto[2])][0]=puntos_zonas[int(punto[2])][0]+punto[0]
        puntos_zonas[int(punto[2])][1]=puntos_zonas[int(punto[2])][1]+punto[1]
        puntos_zonas[int(punto[2])][2]=puntos_zonas[int(punto[2])][2]+punto[3]
    else:
        puntos_zonas.append([int(punto[0]),int(punto[1]),int(punto[3])])

for a in range(0,(len(pzona))):
    centro=np.array(puntos_zonas[a])
    centro_z[0]=(centro[0]/(int(pzona[a])))#Suma coordenadas X dividido nº sensores de 1 zona
    centro_z[1]=(centro[1]/(int(pzona[a])))
    centro_m.append([centro_z[0],centro_z[1]])
#Calcular centro de masas
for t in range(0,len(matriz_zona)):
    punto=np.array(matriz_zona[t])
    if((len(puntos_zonas_m)-1)==punto[2]):
        puntos_zonas_m[int(punto[2])][0]=puntos_zonas_m[int(punto[2])][0]+(punto[0]*punto[3])
        puntos_zonas_m[int(punto[2])][1]=puntos_zonas_m[int(punto[2])][1]+(punto[1]*punto[3])

    else:
        puntos_zonas_m.append([int(punto[0]*punto[3]),int(punto[1]*punto[3])])
a=0
for a in range(0,(len(pzona))):
    centro=np.array(puntos_zonas_m[a])
    centro_zm[0]=(centro[0]/(int(puntos_zonas[a][2])))
    centro_zm[1]=(centro[1]/(int(puntos_zonas[a][2])))
    centro_mm.append([centro_zm[0],centro_zm[1]])
```

Figura 3.23. Centros y centros de masas

Para terminar de calcular el centro de masa, debemos a diferencia de para calcular un centro normal, en el cual solo dividíamos los valores previamente calculados por el

número de puntos que formaban una zona; dividir por la suma de todos los valores de presión de los puntos que forman una zona. Las variables tienen nombres similares, añadimos una m al final para identificar los valores de los centros de masas.

3.2.6.3 Cálculo del área de cada zona

Calcular el área de las diferentes zonas es bastante sencillo, solo tenemos que saber las medidas de la esterilla y el número de sensores en total para calcular el área de un solo sensor. Teniendo el número de puntos (sensores) que forman cada zona solo tendríamos que multiplicar este número por el área de uno de ellos y tendríamos el área total de cada zona.

En la figura 3.24 podemos ver las dimensiones de la zona que ocupan los sensores. Si calculamos el área de esta zona nos salen 938100 mm^2. Si dividimos este área por el número de sensores que conforman la esterilla, obtendremos aproximadamente el área de 1 sensor. Ya se dijo en el apartado 2.3 Esterillas Fitness Mat Dev Kit 1.9 de Sensing.Tex que los sensores tenían una superficie de 1 cm^2, pero los sensores no están pegados unos a otros si no que hay espacio entre ellos, por ello, para calcular el área de cada zona lo más precisa posible, debemos hacer como que en realidad no hay separación entre ellos, y que los sensores tienen mayor superficie. Teniendo todo esto en cuenta si hay 2240 sensores entonces, cada sensor tendrá un área aproximada de 4 cm^2.

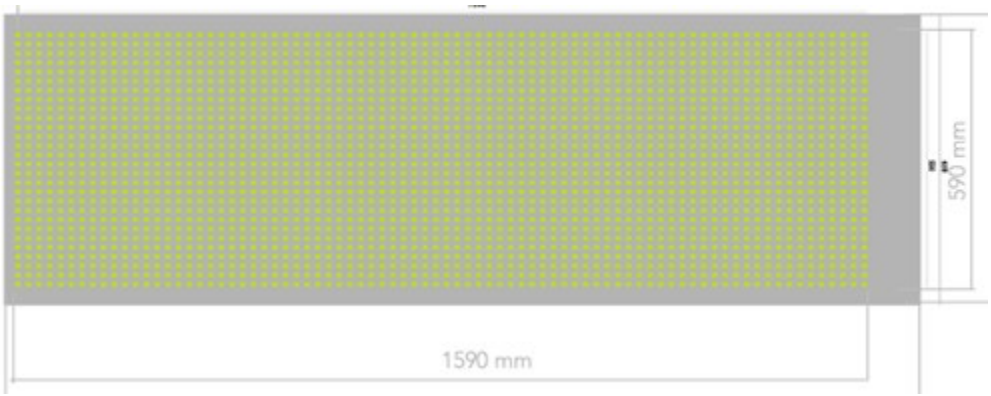


Figura 3.24. Dimensiones Fitness Mat

Con todo lo explicado anteriormente, el código sería el mostrado en la figura 3.25:

```
#Cálculo del área de cada zona
#Teniendo en cuenta las dimensiones de la esterilla y el número de sensores, cada punto tendr
for i in range(0,len(pzona)):
    areas.append((pzona[i]*4))
```

Figura 3.25. Cálculo área zonas

3.2.6.4 Cálculo del peso total de cada zona

También es importante saber cómo se reparte el peso en las diferentes partes del cuerpo, ya que a medida que un bebé crece y, sobre todo, en los primeros meses de vida su centro de gravedad va variando y, por tanto, su peso se reparte de forma diferente.

En primer lugar, calcularemos el peso total de cada zona, es decir, sumaremos los valores de presión de los puntos que conforman cada zona. Posteriormente, ordenaremos estos pesos de menor a mayor e identificaremos a que parte del cuerpo pertenecen.

Como podemos ver en la figura 3.26, en primer lugar, en la variable *pesos_z* almacenamos no solo el valor del peso total sino también las coordenadas del centro de cada zona, esto lo hacemos para posteriormente poder saber a qué parte del cuerpo corresponde cada peso.

```
######Comparación pesos zonas######
pesos=[]
pesos_z=[]

for s in range(0,len(pzona)):#puntos_zonas)):
    #Guardamos las coordenadas del centro y el valor total de presión
    pesos_z.append([puntos_zonas[s][2],puntos_zonas[s][0]/int(pzona[s]),puntos_zonas[s][1]/int(pzona[s])])
print("Pesos:",pesos_z)
#Ordenamos los pesos de menor a mayor con la función sort
pesos_z.sort()
print("Pesos ordenados:",pesos_z)
```

Figura 3.26. Cálculo peso total

Recordar que en la variable *puntos_zonas* almacenábamos la suma de las coordenadas 'x' e 'y', en las dos primeras columnas y en la tercera el valor del peso total. La variable *pzonas* contenía tantos elementos como zonas estuviéramos detectando, y a su vez, el número de puntos que conformaban una zona.

Almacenados los valores, nos toca ordenarlos. La librería Numpy cuenta con una función capaz devolver una copia ordenada de una matriz. Gracias a ellas, seremos capaces de ordenar los pesos sin tener que recurrir a ningún algoritmo de ordenación. Esta función es capaz de ordenar los elementos de una matriz en función del eje que nosotros le indiquemos, de esta forma, esta función puede ordenar arrays bidimensionales.

Como podemos ver en la figura 3.27, al imprimir los resultados, la función *sort* ordena perfectamente la matriz de pesos, teniendo en cuenta solo este valor. De esta forma, como ya se ha dicho previamente nos ahorramos código.

```
Pesos: [[[2514.0], [10.5], [49.5]], [[324], [12.0], [20.0]], [[2061.0], [13.0],
[35.333333333333336]], [[2946.0], [16.5], [48.5]]]
Pesos ordenados: [[[324], [12.0], [20.0]], [[2061.0], [13.0], [35.333333333333336]],
[[2514.0], [10.5], [49.5]], [[2946.0], [16.5], [48.5]]]
```

Figura 3.27. Resultados pesos

Ahora solo nos queda saber a qué parte del cuerpo pertenece cada peso. El procedimiento es casi idéntico al usado previamente para identificar las zonas, solo que ahora en vez de comparar los centros de cada zona, con las diferentes partes del cuerpo, compararemos las coordenadas almacenadas en la variable *pesos_z*, tal y como se muestra en la figura 3.28.

```
for w in range(0,len(pesos_z)):
    for z in range(0,5):
        if(pesos_z[w][1]-4<centro_cuerpo[cuerpo[z]][0]<pesos_z[w][1]+4
           and pesos_z[w][2]-4<centro_cuerpo[cuerpo[z]][1]<pesos_z[w][2]+4):
            if(z==0):
                pesos_z[w].append("cabeza")
            if(z==1):
                pesos_z[w].append("escapulas")
            if(z==2):
                pesos_z[w].append("Riñones")
            if(z==3):
                pesos_z[w].append("pie izquierdo")
            if(z==4):
                pesos_z[w].append("pie derecho")
```

Figura 3.28. Identificación pesos

Añadiremos a esta misma variable un nuevo elemento al final de cada fila para identificar las diferentes zonas. En la figura 3.29 podemos observar los resultados.

```
[[324, 12.0, 20.0, 'cabeza'], [2061.0, 13.0, 35.333333333333336, 'Riñones'],
[2514.0, 10.5, 49.5, 'pie derecho'], [2946.0, 16.5, 48.5, 'pie izquierdo']]
```

Figura 3.29. Resultados pesos ordenados y zonas

A simple vista podríamos decir que los resultados son erróneos si tenemos en cuenta el valor del peso y a qué zona corresponde, ya que es muy poco probable que la fuerza que un bebé ejerza con los pies sea mayor que el peso de su propia cabeza, que recordemos es igual al 30% de su peso total en las primeras semanas de vida. Pero debemos tener en cuenta que los datos con los que estamos trabajando son simulados y no corresponden a datos obtenidos de bebés reales. Lo que debemos fijarnos es en las coordenadas y ver que por su posición si se corresponden con las zonas indicadas.

3.2.6.5 Cálculo de la velocidad de cada zona

Por último, antes de proceder a almacenar los datos para su posterior estudio por parte de los especialistas, vamos a calcular la velocidad de cada zona, es decir, la distancia que recorre cada zona a lo largo del experimento.

En primer lugar, deberemos calcular la distancia recorrida por cada zona, esto lo haremos teniendo en cuenta los centros de cada una de ellas.

Para empezar, tendremos que comprobar si el número de zonas de una matriz con respecto a la siguiente es igual o no. Para ello, compararemos el tamaño de la matriz *centro_m* con el tamaño de la matriz *centro_m_anterior* que contendrá la información de los centros de la matriz anterior. Nos quedaremos con el menor tamaño, para no excedernos al recorrer la matriz posteriormente. Esto se muestra en la figura 3.30.

```
for u in range(0,len(paquete_datos)):
    comparar(u)
    t=0
    matrices.append(u)
    matriz_zona=np.array(paquete_zonas[u])#i

    if(u==0):
        centro_m_anterior=centro_m
    else:
        if (len(centro_m_anterior)<len(centro_m)):
            limitee=len(centro_m_anterior)
        if(len(centro_m_anterior)>len(centro_m)):
            limitee=len(centro_m)
        if(len(centro_m_anterior)==len(centro_m)):
            limitee=len(centro_m)
```

Figura 3.30 Comparar tamaños

Una vez calculado este límite, calcularemos la distancia entre el centro de una zona de dos matrices consecutivas. Emplearemos la fórmula de la distancia entre dos puntos que se puede ver en la figura 3.31.

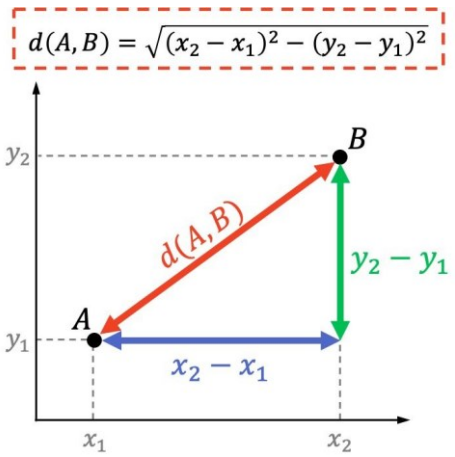


Figura 3.31 Fórmula distancia entre dos puntos.

Para elevar al cuadrado y calcular la raíz cuadrada emplearemos la función pow. Recordar que calcular la raíz cuadrada de un número no es más que elevarlo a 0.5.

Para identificar a que zona pertenecen los centros, haremos igual que previamente, comprobaremos si las coordenadas del centro coinciden con las coordenadas de las zonas del cuerpo previamente identificadas. Debemos tener en cuenta que tenemos que comprobar las coordenadas de ambos centros, ya que no tienen por qué haberse identificado en el mismo orden. Posteriormente, calcularemos la distancia entre ambos puntos con la fórmula comentada previamente. Iremos almacenando estos valores para tener la distancia total, como vemos en la figura 3.32.

```
for w in range(0,limitee):
    for z in range(0,5):
        if((centro_m[w][0]-4<centro_cuerpo[cuerpo[z]][0]<centro_m[w][0]+4
            and centro_m[w][1]-4<centro_cuerpo[cuerpo[z]][1]<centro_m[w][1]+4 )):
            for z1 in range(0,5):
                if((centro_m_anterior[w][0]-4<centro_cuerpo[cuerpo[z1]][0]<centro_m_anterior[w][0]+4
                and centro_m_anterior[w][1]-4<centro_cuerpo[cuerpo[z1]][1]<centro_m_anterior[w][1]+4)):
                    if(z==0 and z1==0):
                        dist_cab=(pow((pow((centro_m_anterior[w][0]-centro_m[w][0]),2.0)+
                                       (pow((centro_m_anterior[w][1]-centro_m[w][1]),2.0))),0.5))
                        discab=discab + dist_cab
                    if(z==1 and z1==1):
                        dist_esc=(pow((pow((centro_m_anterior[w][0]-centro_m[w][0]),2.0)+
                                       (pow((centro_m_anterior[w][1]-centro_m[w][1]),2.0))),0.5))
                        disesc=disesc + dist_esc
                    if(z==2 and z1==2):
                        dist_riñ=(pow((pow((centro_m_anterior[w][0]-centro_m[w][0]),2.0)+
                                       (pow((centro_m_anterior[w][1]-centro_m[w][1]),2.0))),0.5))
                        disriñ=disriñ + dist_riñ
                    if(z==3 and z1==3):
                        dist_pi=(pow((pow((centro_m_anterior[w][0]-centro_m[w][0]),2.0)+
                                      (pow((centro_m_anterior[w][1]-centro_m[w][1]),2.0))),0.5))
                        dispi=dispi + dist_pi
                    if(z==4 and z1==4):
                        dist_pd=(pow((pow((centro_m_anterior[w][0]-centro_m[w][0]),2.0)+
                                      (pow((centro_m_anterior[w][1]-centro_m[w][1]),2.0))),float(0.5)))
                        dispd=dispd + dist_pd
```

Figura 3.32 Calcular distancias

Antes de vaciar los arrays empleados, para poder usarlos nuevamente debemos guardar la matriz con la información de los centros, en otra auxiliar para poder compararla posteriormente, tal como se muestra en la figura 3.33.

```
centro_m_anterior=centro_m
areas=[]
zonas=[]
centro_mm=[]
puntos_zonas=[]
centro_m=[]
```

Figura 3.33 Matriz auxiliar

Por último, calculadas las distancias totales solo nos queda hallar la velocidad. Para ello, calcularemos el tiempo en segundos que ha durado el experimento. Previamente, ya habíamos calculado los minutos y segundos de duración, por ello, ahora solo tendremos que multiplicar por 60 los minutos para hallar el tiempo total en segundos. La velocidad se calcula como distancia partido de tiempo, pero la distancia que hemos calculado no es la distancia real. Si tenemos en cuenta que el área que ocupan los sensores es de 160x56 cm y que la matriz que recoge los datos es de 80x28, con una simple regla de tres podemos calcular la velocidad en cm/s como podemos observar en la figura 3.34.

```
#Calculamos velocidad media como distancia partido por tiempo
tiempo=float(minn[0]*60+seg[0])
v_c=(float(discab/tiempo))*16/8
v_e=(float(disesc/tiempo))*16/8
v_r=(float(disriñ/tiempo))*16/8
v_i=(float(dispi/tiempo))*16/8
v_d=(float(dispd/tiempo))*16/8
```

Figura 3.34 Calculo de velocidades

3.2.7 Almacenamiento de los datos

El estudio del movimiento de un bebé es algo que requiere de tiempo, es decir, es necesario ir observando una evolución a medida que pasan las semanas. Es por ello que, necesitamos almacenar estos datos para poder compararlos. Para ello, nos ayudaremos de los dataframes.

Un dataframe es una estructura de datos bidimensional que almacena datos de diferentes tipos, como caracteres, valores enteros, punto flotante, etc; en columnas. Se parece a una hoja de cálculo. Tiene un índice que empieza en cero, el cual representa la posición que ocupa un elemento en la estructura de datos.

Un dataframe en Python es fácilmente exportable a Excel. Gracias a esto seremos capaces de comparar los datos de dos experimentos distintos, pudiendo observar la evolución que siguen los pacientes.

Como se ha dicho previamente, un dataframe no es más que un array bidimensional, el cual tendrá tantas filas y columnas como deseemos. Crearemos varios arrays unidimensionales que contendrán los valores de las columnas, es decir, cada elemento de dicho array corresponderá a un valor de cada matriz del experimento.

Por ejemplo, crearemos un array llamado *cabeza_p* que contendrá el valor del peso (presión) que tiene la zona de la cabeza en todo momento, así este array estará formado por tantos elementos como matrices tenga nuestro experimento.

Para la zona de la cabeza tendremos 6 matrices distintas, que guardarán diferentes parámetros como el peso, el área que ocupa que ocupa esta zona, las coordenadas del centro y el centro de masas, así como en cuantas matrices de las totales aparece la zona de la cabeza. Este último es un parámetro significativo, al cual los expertos le dan gran importancia, ya que durante los primeros meses uno de los grandes desarrollos motores que se detectan en este tipo de pacientes es el aguante de la cabeza. La declaración de variables se realiza tal y como se muestra en la figura 3.35

```
####################Dataframe#################################
cabeza_p=[]#Peso de la zona de la cabeza
cabeza_a=[]#Área de la zona de la cabeza
cabeza_x=[]#Coordenada x centro zona cabeza
cabeza_y=[]#Coordenada y centro zona cabeza
centro_m_x=[]#Coordenada x centro de masas zona cabeza
centro_m_y=[]#Coordenada y centro de masas zona cabeza
num_cabeza=[]#En cuantas matrices aparece la zona cabeza
```

Figura3.35 Matrices zona cabeza

Esto lo repetiremos para el resto de las zonas del cuerpo, tal y como aparece en la figura 3.36:

```
escapulas_p=[]
escapulas_a=[]
escapulas_x=[]
escapulas_y=[]
escapulas_m_x=[]
escapulas_m_y=[]
num_escapulas=[]

riñones_p=[]
riñones_a=[]
riñones_x=[]
riñones_y=[]
riñones_m_x=[]
riñones_m_y=[]
num_riñones=[]

pied_p=[]
pied_a=[]
pied_x=[]
pied_y=[]
pied_m_x=[]
pied_m_y=[]
num_pied=[]

piei_p=[]
piei_a=[]
piei_x=[]
piei_y=[]
piei_m_x=[]
piei_m_y=[]
num_piei=[]
```

Figura 3.36 Matrices resto partes del cuerpo

También guardaremos otros datos de interés como son la duración del experimento o qué lado predomina, como vemos en la figura 3.37.

```
minn=[float(minfin)-float(minini)]
seg=[float(segundosfin)-float(segundosini)]
minT=[float(min_fin)-float(min_ini)]
segT=[float(segundos_fin)-float(segundos_ini)]

predomina=[]
```

Figura 3.37 Matrices tiempo y lado predominante

La duración es algo que ya calculamos previamente y es un valor único, por lo tanto, estas matrices solo tendrán un elemento.

A continuación, guardaremos los datos correspondientes en cada matriz.

Como podemos ver en la figura 3.38 en el código previamente ya descrito en el apartado 3.2.6.4 Cálculo del peso total de cada zona añadimos las líneas que aparecen subrayadas en amarillo. De esta forma, cada vez que identifiquemos a que zona corresponde cada peso, almacenaremos su valor para posteriormente guardarlo.

```
cab=0
esc=0
riñ=0
pi=0
pd=0
for w in range(0,len(pesos_z)):
    for z in range(0,5):
        if(pesos_z[w][1]-4<centro_cuerpo[cuerpo[z]][0]<pesos_z[w][1]+4
           and pesos_z[w][2]-4<centro_cuerpo[cuerpo[z]][1]<pesos_z[w][2]+4):
            if(z==0):
                pesos_z[w].append("cabeza")
                cabeza_p.append(pesos_z[w][0])
                cab=1
            if(z==1):
                pesos_z[w].append("escapulas")
                escapulas_p.append(pesos_z[w][0])
                esc=1
            if(z==2):
                pesos_z[w].append("Riñones")
                riñones_p.append(pesos_z[w][0])
                riñ=1
            if(z==3):
                pesos_z[w].append("pie izquierdo")
                piei_p.append(pesos_z[w][0])
                pi=1
            if(z==4):
                pesos_z[w].append("pie derecho")
                pied_p.append(pesos_z[w][0])
                pd=1
```

Figura 3.38 Guardamos los pesos

A su vez declaramos las variables *cab*, *esc*, *riñ*, *pi*, *pd*, las cuales valdrán 1 si en la matriz que estamos analizando aparecen esas zonas. Esto lo hacemos porque para crear el

dataframe, que posteriormente exportaremos a Excel, necesitamos que todos los arrays tengan la misma longitud. Así, cuando detectemos que una zona no aparece deberemos agregar un cero a los arrays correspondientes, tal y como se muestra en el código de la figura 3.39.

```
if(cab+esc+riñ+pi+pd<5): #Comprobamos si aparecen las 5 zonas
    if(cab==0):
        cabeza_p.append(0)
        cabeza_a.append(areas[w])
        cabeza_x.append(0)
        cabeza_y.append(0)
        cabeza_m_x.append(0)
        cabeza_m_y.append(0)
    if(esc==0):
        escapulas_p.append(0)
        escapulas_a.append(areas[w])
        escapulas_x.append(0)
        escapulas_y.append(0)
        escapulas_m_x.append(0)
        escapulas_m_y.append(0)
    if(riñ==0):
        riñones_p.append(0)
        riñones_a.append(0)
        riñones_x.append(0)
        riñones_y.append(0)
        riñones_m_x.append(0)
        riñones_m_y.append(0)
    if(pi==0):
        piei_p(0)
        piei_a.append(0)
        piei_x.append(0)
        piei_y.append(0)
        piei_m_x.append(0)
        piei_m_y.append(0)
    if(pd==0):
        pied_p.append(0)
        pied_a.append(areas[w])
        pied_x.append(0)
        pied_y.append(0)
        pied_m_x.append(0)
        pied_m_y.append(0)
```

Figura 3.39 Detectar que zonas no aparecen

Para guardar el área de cada zona primero debemos identificar a que zona pertenece, igual que hemos hecho previamente con los pesos, debemos comparar las coordenadas de los centros, que están guardadas en el mismo orden que las áreas.

También aprovecharemos para guardar los datos de las coordenadas ‘x’ e ‘y’ de los centros y centros de masa, y contaremos cuantas veces aparece cada zona. De esta forma, tenemos almacenados todos los datos con dos simples bucles for, como se muestra en las figuras 3.40 y 3.41.

Como se ha comentado previamente, todos los arrays que formen parte del dataframe deben tener el mismo tamaño. Sin embargo, las matrices para el recuento de veces que

aparece cada zona están formadas por un único elemento. Por ello, crearemos una datraframe a parte que guardaremos en el mismo Excel, pero en una hoja de cálculo diferente. Haremos lo mismo con los valores del tiempo.

```
#Identificamos las áreas
for w in range(0,len(centro_m)):
    for z in range(0,5):
        if((centro_m[w][0]-4<centro_cuerpo[cuerpo[z]][0]<centro_m[w][0]+4
            and centro_m[w][1]-4<centro_cuerpo[cuerpo[z]][1]<centro_m[w][1]+4)):
            if(z==0):
                cabeza_a.append(areas[w])
                cabeza_x.append(centro_m[w][0])
                cabeza_y.append(centro_m[w][1])
                cabeza_m_x.append(centro_mm[w][0])
                cabeza_m_y.append(centro_mm[w][1])
                num_cabeza[0]=num_cabeza[0]+1
            if(z==1):
                escapulas_a.append(areas[w])
                escapulas_x.append(centro_m[w][0])
                escapulas_y.append(centro_m[w][1])
                escapulas_m_x.append(centro_mm[w][0])
                escapulas_m_y.append(centro_mm[w][1])
                num_escapulas[0]=num_escapulas[0]+1
            if(z==2):
                riñones_a.append(areas[w])
                riñones_x.append(centro_m[w][0])
                riñones_y.append(centro_m[w][1])
                riñones_m_x.append(centro_mm[w][0])
                riñones_m_y.append(centro_mm[w][1])
                num_riñones[0]=num_riñones[0]+1
```

Figura 3.40. Guardar e identificar áreas (1)

```
            if(z==3):
                piei_a.append(areas[w])
                piei_x.append(centro_m[w][0])
                piei_y.append(centro_m[w][1])
                piei_m_x.append(centro_mm[w][0])
                piei_m_y.append(centro_mm[w][1])
                num_piei[0]=num_piei[0]+1
            if(z==4):
                pied_a.append(areas[w])
                pied_x.append(centro_m[w][0])
                pied_y.append(centro_m[w][1])
                pied_m_x.append(centro_mm[w][0])
                pied_m_y.append(centro_mm[w][1])
                num_pied[0]=num_pied[0]+1
```

Figura 3.41 Guardar e identificar áreas (2)

Por último, nos falta almacenar los valores de qué lado predomina cada vez, para ello, simplemente añadiremos las líneas que aparecen subrayadas en amarillo, en la figura 3.42, al código del apartado 3.2.6.1 Determinación del lado predominante.

```
#Saber que lado predomina
for t in range(0,len(matriz_zona)):
    punto=np.array(matriz_zona[t])#leemos un punto de la matriz x,y,zona,valor
    if (punto[0]>13):#punto parte superior (nos fijamos en la coordenada x)
        arriba=arriba+1
    if (punto[0]<13):
        abajo=abajo+1
    if (punto[0]==13):
       mitad=mitad+1
if (mitad >arriba and mitad >abajo): #La mayoría de puntos queda en la mitad
    predomina.append("mitad")
    superior[2]=superior[2]+1
if(arriba>abajo): #La mayoría de puntos están en la parte superior
    superior[0]=superior[0]+1
    predomina.append("superior")
if(arriba == abajo):
    empate=1
    predomina.append("empate")
else:
    superior[1]=superior[1]+1
    predomina.append("inferior")
```

Figura 3.42 Guardar lado predominate

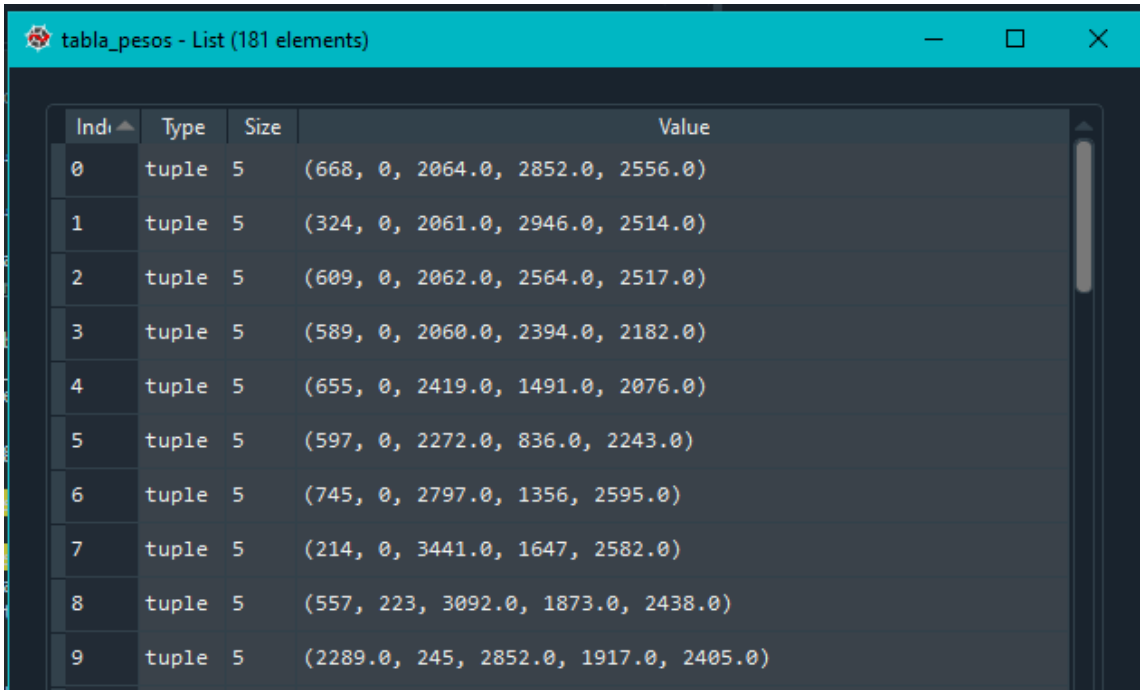
Almacenados todos los datos en sus correspondientes matrices nos queda juntar todas las matrices, para formar una tabla y exportarla a Excel, como vemos en la figura 3.43.

Para ello, emplearemos la función *zip()*, la cual coge como argumentos varios elementos iterables, como mínimo dos, preferiblemente que tengan el mismo tamaño y devuelve un nuevo objeto iterable formado por tuplas, las cuales están compuestas por un elemento de cada uno de los iteradores originales.

```
tabla_pesos = list(zip(cabeza_p,escapulas_p,riñones_p,piei_p,pied_p))

tabla_areas = list(zip(cabeza_a,escapulas_a,riñones_a,piei_a,pied_a))

tabla_centros = (list(zip(cabeza_x,cabeza_y,escapulas_x,escapulas_y,
                riñones_x,riñones_y,piei_x,piei_y,pied_x,pied_y)))

tabla_centrosmasa= (list(zip(cabeza_m_x,cabeza_m_y,escapulas_m_x,
                   escapulas_m_y,riñones_m_x,riñones_m_y,
                   piei_m_x,piei_m_y,pied_m_x, pied_m_y)))

tabla_tiempos= list(zip(minn,seg,minT,segT))
```

Figura 3.43 Tablas formadas con la función zip()

Para comprobar que esto se realiza correctamente, si nos fijamos, dentro del explorador de variables que nos proporciona la aplicación Spyder, en la variable tabla_pesos, veremos cómo la tabla está formada por las diferentes tuplas. Esto se refleja en la figura 3.44.



tabla_pesos - List (181 elements)

Ind	Type	Size	Value
0	tuple	5	(668, 0, 2064.0, 2852.0, 2556.0)
1	tuple	5	(324, 0, 2061.0, 2946.0, 2514.0)
2	tuple	5	(609, 0, 2062.0, 2564.0, 2517.0)
3	tuple	5	(589, 0, 2060.0, 2394.0, 2182.0)
4	tuple	5	(655, 0, 2419.0, 1491.0, 2076.0)
5	tuple	5	(597, 0, 2272.0, 836.0, 2243.0)
6	tuple	5	(745, 0, 2797.0, 1356, 2595.0)
7	tuple	5	(214, 0, 3441.0, 1647, 2582.0)
8	tuple	5	(557, 223, 3092.0, 1873.0, 2438.0)
9	tuple	5	(2289.0, 245, 2852.0, 1917.0, 2405.0)

Figura 3.44 Variable tabla:pesos

La tabla que contiene los valores de qué lado predomina en cada momento, no es necesaria aplicarle esta función ya que es una matriz unidimensional.

Para poder visualizar los datos con mayor claridad dentro de Excel, los guardaremos en diferentes hojas de cálculo, así pues, deberemos crear varios dataframes diferentes.

Para ello, deberemos emplear la librería pandas y hacer uso de la función *DataFrame()*, a la cual pasaremos como argumentos la tabla indicada y el nombre que queremos que tengan las distintas columnas. También, podemos hacer que no aparezca el índice de las filas, pero en nuestro caso este índice no es de utilidad para saber a qué matriz pertenecen los datos. El código referente a esto, se muestra en la figura 3.45.

```
dframe1 = pd.DataFrame(tabla_pesos, columns=['cabeza_p','escapulas_p','riñones_p','piei_p','pied_p'])

dframe2= pd.DataFrame(tabla_areas, columns=['cabeza_a','escapulas_a','riñones_a','piei_a','pied_a'])

dframe3=(pd.DataFrame(tabla_centros, columns=['cabeza_x','cabeza_y','escapulas_x','escapulas_y',
                                              'riñones_x','riñones_y','piei_x','piei_y',
                                              'pied_x','pied_y']))

dframe4=(pd.DataFrame(tabla_centrosmasa, columns=['cabeza_m_x','cabeza_m_y','escapulas_m_x','escapulas_m_y',
                                              'riñones_m_x','riñones_m_y','piei_m_x','piei_m_y',
                                              'pied_m_x','pied_m_y']))

dframe5=pd.DataFrame(tabla_tiempos, columns=['Minutos','Segundos','MinutosTotales','SegundosTotales'])

dframe6=pd.DataFrame(predomina, columns=['Zona predominante'])
```

Figura 3.45 Creación de los dataframes

Hecho esto, tenemos que llamar al escritor de Excel para indicarle, como queremos que se llame nuestro documento y la extensión que queremos que este tenga, como se ve en la figura 3.46.

```
writer = pd.ExcelWriter('datosbebes.xls')
```

Figura 3.46 Excel Writer

Posteriormente, indicaremos el nombre que deseamos tengan las diferentes hojas de cálculo, llamando a la función *to_excel()* haciendo uso del escritor previamente definido. Finalmente, llamaremos a la función *save()* para guardar el documento Excel, como se ve en la figura 3.47.

```
with ExcelWriter('datosbebes.xls') as writer:
    dframe1.to_excel(writer, 'PESOS')
    dframe2.to_excel(writer, 'ÁREAS')
    dframe3.to_excel(writer, 'CENTROS')
    dframe4.to_excel(writer, 'CENTROS_MASA')
    dframe5.to_excel(writer, 'TIEMPOS')
    dframe6.to_excel(writer, 'PREDOMINA')
writer.save()
```

Figura 3.47 Hojas de cálculo

Ahora solo nos queda comprobar que efectivamente el documento Excel se ha creado correctamente. Como podemos observar en la figura 3.48, a continuación, el archivo ha sido creado:

Figura 3.48 Documento Excel

Si abrimos dicho archivo vemos como se han creado las diferentes hojas de cálculo, y nos aparecen todos los datos. Estos datos se muestran en las figuras desde la 3.49 a la 3.55.

	A	B	C	D	E	F
1		cabeza_p	escapulas_p	riñones_p	piei_p	pied_p
2	0	668	0	2064	2852	2556
3	1	324	0	2061	2946	2514
4	2	609	0	2062	2564	2517
5	3	589	0	2060	2394	2182
6	4	655	0	2419	1491	2076
7	5	597	0	2272	836	2243
8	6	745	0	2797	1356	2595
9	7	214	0	3441	1647	2582
10	8	557	223	3092	1873	2438
11	9	2289	245	2852	1917	2405
12	10	4373	333	3157	1820	2318
13	11	4719	316	2717	1427	2161
14	12	5590	495	2726	842	2118
15	13	5988	487	3331	1783	2232
16	14	6336	518	3733	2276	2290

Figura 3.49 Hoja cálculo PESOS

	A	B	C	D	E	F
1		cabeza_a	escapulas_a	riñones_a	piei_a	pied_a
2	0	4	16	12	16	16
3	1	4	16	12	16	16
4	2	4	16	12	16	16
5	3	4	8	12	8	16
6	4	4	8	20	8	12
7	5	4	8	12	8	12
8	6	4	4	12	4	16
9	7	4	4	16	4	16
10	8	4	4	16	8	12
11	9	16	4	16	8	12
12	10	24	4	16	8	12
13	11	20	4	12	8	12
14	12	20	4	12	8	8
15	13	16	4	20	12	8
16	14	20	8	20	8	8

Figura 3.50 Hoja cálculo ÁREAS

	A	B	C	D	E	F	G	H	I	J	K
1		cabeza_x	cabeza_y	escapulas_x	escapulas_y	riñones_x	riñones_y	piei_x	piei_y	pied_x	pied_y
2	0	12	20	0	0	13	35.3333333	16.5	48.5	10.5	49.5
3	1	12	20	0	0	13	35.3333333	16.5	48.5	10.5	49.5
4	2	12	20	0	0	13	35.3333333	16.5	48.5	10.5	49.5
5	3	12	20	0	0	13	35.3333333	17	48.5	10.5	49.5
6	4	12	20	0	0	12.8	35.4	17	48.5	10.6666667	49.3333333
7	5	12	20	0	0	12.3333333	35.3333333	17.5	48	10.6666667	49.3333333
8	6	12	20	0	0	12.3333333	35.3333333	18	48	10.5	49.5
9	7	14	22	0	0	12.75	35.25	18	48	10.5	49.5
10	8	12	20	15	29	12.75	35.25	18	48.5	10.3333333	49.3333333
11	9	13.5	20.75	11	28	12.75	35.25	18	48.5	10.3333333	49.3333333
12	10	13.1666667	20.8333333	15	29	12.75	35.25	18	48.5	10.6666667	49.3333333
13	11	12.4	20.6	11	28	13	35	18	48.5	10.6666667	49.3333333
14	12	12	20.8	15	29	13	35	17.5	48	11	48.5
15	13	11.75	21	11	28	12.8	34.6	17.3333333	48.3333333	11	49.5
16	14	11.6	20.8	14.5	28.5	12.8	34.6	17	48.5	11	49.5
17	15	11.5	21	11	28	12.8	34.6	17	48.5	11.3333333	49.6666667
18	16	11.5	21	14.5	28.5	12.8	34.6	17	48.5	11.3333333	49.6666667
19	17	11.5	21	11	28	12.8	34.6	17	48.5	11.5	49.5

Figura 3.51 Hoja cálculo CENTROS

	A	B	C	D	E	F	G	H	I	J	K
1		cabeza_m_x	cabeza_m_y	scapulas_m_	scapulas_m_	riñones_m_x	riñones_m_y	piei_m_x	piei_m_y	pied_m_x	pied_m_y
2	0	12	20	0	0	12.8624031	35.369186	16.3856942	48.3804348	10.2832551	49.521518
3	1	24.7407407	41.2345679	0	0	12.8811257	35.4206696	15.8628649	46.8367278	10.4550517	50.3488465
4	2	13.1625616	21.9376026	0	0	12.8748788	35.4034918	18.226209	53.8147426	10.4425904	50.2888359
5	3	13.6095076	22.6825127	0	0	12.8873786	35.4378641	19.5204678	57.6361738	12.0458295	58.0096242
6	4	12.2381679	20.3969466	0	0	10.974783	30.1785862	31.342723	92.5425889	12.6608863	60.97158
7	5	13.4271357	22.3785595	0	0	11.6848592	32.131162	55.8995215	165.049043	11.7182345	56.4320107
8	6	10.7597315	17.9328859	0	0	9.49159814	26.1001073	34.4631268	101.7559	10.1287091	48.777264
9	7	124.056075	341.130841	0	0	2.32955536	3.88259227	28.3740134	83.7771706	10.1797057	49.0228505
10	8	14.3913824	23.9856373	209.560538	618.748879	8.58602846	23.6099612	4.61825948	22.6294714	10.780968	51.9183757
11	9	11.5980778	31.8925295	32.7183673	54.5306122	16.3856942	48.3804348	4.77308294	23.8654147	10.9288981	52.6307692
12	10	6.07088955	16.6938029	25.975976	127.282282	14.8026608	43.7063668	5.02747253	25.1373626	11.3390854	54.606126
13	11	1.69866497	2.83110829	16.1939394	26.989899	17.1998528	50.7843209	6.41205326	32.0602663	12.1628876	58.5733457
14	12	4.7019678	22.6434705	27.3734177	134.129747	17.1430668	50.6166544	10.8669834	54.3349169	12.5344665	34.4674221
15	13	4.38944556	21.1384436	54.513347	149.901437	14.0294206	41.4232963	5.13180034	25.6590017	11.8942652	32.7069892
16	14	4.14835859	19.9774306	16.6988417	81.8243243	12.5186177	36.9624967	4.0202109	20.1010545	11.5930131	31.8786026
17	15	3.95664609	19.0541924	17.9730942	29.955157	11.9887122	35.3978964	3.51923077	17.5961538	14.530925	39.9573071
18	16	3.96979308	19.1175049	15.7846715	77.3448905	11.3620229	33.5475322	3.25970787	16.2985394	24.0907441	66.2450091
19	17	3.89508002	18.757706	21.0947368	35.1578947	11.4343039	33.7609494	3.51340374	17.2156783	29.3672566	80.7544248
20	18	3.81702004	18.3817891	13.6650869	66.9589258	5.70801978	15.6959794	0	0	3.4611399	5.76856649

Figura 3.52 Hoja cálculo CENTRO_MASA

	A	B	C	D	E
1		Minutos	Segundos	MinutosTotales	SegundosTotales
2	0	0	15.61556	0	20.187898
3					
4					
5					

Figura 3.53 Hoja cálculo TIEMPOS

	A	B	C
1		Zona predominante	
2	0	inferior	
3	1	inferior	
4	2	inferior	
5	3	inferior	
6	4	inferior	
7	5	inferior	
8	6	inferior	
9	7	inferior	
10	8	inferior	
11	9	empate	
12	10	inferior	
13	11	inferior	
14	12	inferior	
15	13	inferior	
16	14	inferior	

Figura 3.54 Hoja cálculo PREDOMINA

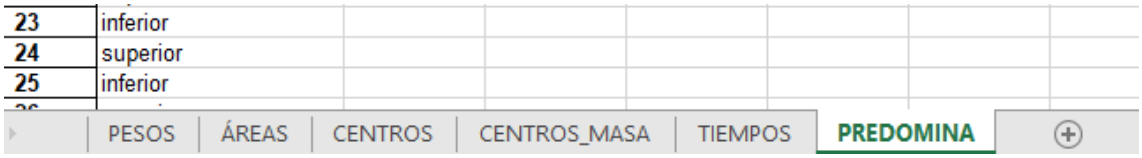


Figura 3.55 Todas las hojas de cálculo

3.2.8 Dando formato a los ficheros de datos Excel generados

Como hemos visto el Excel anterior almacena correctamente todos los datos, si bien es cierto no tiene ningún formato y eso hace que su lectura sea algo complicada. Por ello, vamos a aplicar formato a una de las hojas, el resto de ellas se haría de forma muy similar variando colores y gráficos, es por esta razón que se va a omitir el dar formato al resto de las hojas, quedando esto como posible trabajo futuro.

En primer lugar, vamos a definir las funciones encargadas de dar formato a la tabla, es decir, aplicar color y tamaño al borde, también podremos agrupar celdas; y otra función encargada de crear en esa misma hoja de Excel un gráfico de barras.

3.2.8.1 Función formato_excel():

La función recibirá como parámetro la hoja del Excel a la que se le quiere aplicar el formato. Como se muestra en la figura 3.56, primero definimos uno de los colores de nuestra tabla. Posteriormente, guardamos en la variable *max_col_info* el número de columnas que tendrá nuestra tabla e indicaremos la anchura que queremos que tenga cada una de estas columnas. A su vez, también indicamos la altura que deseamos tengan determinadas filas. Por último, agrupamos las 5 primeras celdas para poder introducir en ellas el título de la tabla.

```
def formato_excel(hoja):#Función para dar formato al excel

    color='333F4F'
    max_col_info = 6 #Columnas de la A a la F
    hoja.column_dimensions["A"].width = 4
    hoja.column_dimensions["B"].width = 15
    hoja.column_dimensions["C"].width = 15
    hoja.column_dimensions["D"].width = 15
    hoja.column_dimensions["E"].width = 15
    hoja.column_dimensions["F"].width = 15
    hoja.row_dimensions[1].height = 35
    hoja.row_dimensions[2].height = 25
    hoja.row_dimensions[5].height = 15

    hoja.merge_cells(('A1:F1'))
```

Figura 3.56 Función formato_excel()

Ahora pasaremos a especificar el formato por celdas. Con la función iter_rows() podemos indicar desde que fila y columna hasta que fila y columna aplicar el formato que deseemos. Como vemos en la figura 3.57 declaramos las variables *double* y *thin* que van a representar el ancho de las líneas que forman la tabla, siendo double una línea gruesa, la cual usaremos para el contorno de la tabla y thin una línea fina para delimitar el resto de las celdas.

Con el primer bucle *for* daremos formato al título de la tabla. Iremos recorriendo las diferentes filas que seleccionemos y aplicándoles el formato. En este caso indicaremos que la letra de esas celdas debe ser de color blanco, que el color de relleno debe ser sólido, que el texto debe situarse en el centro de la celda y que los bordes superior, inferior, derecho e izquierdo deben tener línea gruesa.

```
double = Side(border_style="double", color="000000")
thin = Side(border_style="thin", color="000000")
for rows in hoja.iter_rows(min_row=1, max_row=1, min_col=1, max_col=max_col_info):
    for cell in rows:
        cell.font=Font(color=colors.WHITE)
        cell.fill = PatternFill(fgColor=color, fill_type = "solid")
        cell.alignment = Alignment(wrap_text=True, horizontal='center', vertical='center')
        cell.border = Border(top=double, left=double, right=double, bottom=thin)
```

Figura 3.57 Primer bucle for

Con el segundo bucle *for* que se muestra en la figura 3.58 haremos que las líneas del resto de celdas de la tabla tengan como borde una línea fina.

```
for rows in hoja.iter_rows(min_row=2,max_row=len(paquete_datos)+2, min_col=1, max_col=6):
    for cell in rows:
        cell.border = Border(top=thin, left=thin, right=thin, bottom=thin)
```

Figura 3.58 Segundo bucle for

Con los dos siguientes bucles *for*, que aparecen en la figura 3.59, haremos que la columna que indica el número de filas y los títulos de cada columna tengan el mismo color y el texto aparezca centrado.

```
for rows in hoja.iter_rows(min_row=2, max_row=(len(paquete_datos)+2), min_col=1, max_col=1):
    for cell in rows:
        color='94A5F9'
        cell.fill = PatternFill(fgColor=color, fill_type = "solid")
        cell.alignment = Alignment(wrap_text=True, horizontal='center', vertical='center')

for rows in hoja.iter_rows(min_row=2, max_row=2, min_col=2, max_col=6):
    for cell in rows:
        color='94A5F9'
        cell.fill = PatternFill(fgColor=color, fill_type = "solid")
        cell.alignment = Alignment(wrap_text=True, horizontal='center', vertical='center')
```

Figura 3.59 Tercer y cuarto bucle for

Con el cuarto bucle for que vemos en la figura 3.60 pondremos cada fila de un color, para ello, usaremos la variable *x1*, a la cual iremos sumando 1 por cada iteración del bucle. Si *x1* es par, asignaremos a la variable *color* un código y si es impar otro. Las celdas tendrán borde fino y la alineación del texto será central.

```
for rows in hoja.iter_rows(min_row=3, max_row=(len(paquete_datos)+2), min_col=2, max_col=6):
    color='F3F4FD'
    x1 += 1
    if (x1%2 == 0):
        color='D7DDFC'
    for cell in rows:
        cell.fill = PatternFill(fgColor=color, fill_type = "solid")
        cell.alignment = Alignment(wrap_text=True, horizontal='center', vertical='center')
        cell.border = Border(top=thin, left=thin, right=thin, bottom=thin)
```

Figura 3.60 Cuarto bucle for

Por último, como vemos en la figura 3.61 daremos formato a una tabla pequeña que contendrá la media de los valores de la tabla grande que contiene todos los datos. Con esta tabla crearemos un gráfico de barras de tal forma que de un simple vistazo podremos ver qué zonas han ejercido mayor presión. El formato de dicha tabla será el mismo que el de la tabla grande.

```
for rows in hoja.iter_rows(min_row=3, max_row=(len(paquete_datos)+2), min_col=2, max_col=6):
    color='F3F4FD'
    x1 += 1
    if (x1%2 == 0):
        color='D7DDFC'
    for cell in rows:
        cell.fill = PatternFill(fgColor=color, fill_type = "solid")
        cell.alignment = Alignment(wrap_text=True, horizontal='center', vertical='center')
        cell.border = Border(top=thin, left=thin, right=thin, bottom=thin)

for rows in hoja.iter_rows(min_row=4, max_row=8, min_col=8, max_col=8):
    for cell in rows:
        color='94A5F9'
        cell.fill = PatternFill(fgColor=color, fill_type = "solid")
        cell.alignment = Alignment(wrap_text=True, horizontal='center', vertical='center')
        cell.border = Border(top=thin, left=thin, right=thin, bottom=thin)
for rows in hoja.iter_rows(min_row=3,max_row=3, min_col=9, max_col=9):
    for cell in rows:
        color='333F4F'
        cell.fill = PatternFill(fgColor=color, fill_type = "solid")
        cell.alignment = Alignment(wrap_text=True, horizontal='center', vertical='center')
        cell.font=Font(color=colors.WHITE)
        cell.border = Border(top=double, left=double, right=double, bottom=thin)
```

Figura .3.61 Formato tabla pequeña

3.2.8.2 Función *graficas*():

La función *graficas*() recibe como parámetro el nombre de la hoja de Excel donde se van a realizar los cambios. Como se ha dicho previamente en la hoja de Excel en la que aparecen los datos vamos a crear una pequeña tabla la cual vamos a usar para introducir un gráfico de barras.

Lo primero que haremos es crear la tabla que va a contener los datos de la media de los pesos de cada zona. Para ello, tal y como se muestra en la figura 3.62, usaremos la función sum() de la librería Numpy, la cual es capaz de sumar todos los valores de un array. Posteriormente, para calcular la media solo tendremos que dividir por el número de veces que aparece cada zona. Debemos tener en cuenta que no podemos dividir por el tamaño de los arrays que previamente hemos sumado, ya que en ellos hay valores que son cero, para hacer que todos los arrays tengan la misma longitud.

```
def graficas(hoja):
    #Suma todos los pesos de cada zona
    sumc=np.sum(cabeza_p)
    sume=np.sum(escapulas_p)
    sumr=np.sum(riñones_p)
    sumd=np.sum(pied_p)
    sumi=np.sum(piei_p)
    #Calculamos la media
    #Dividimos por el numero de veces que la zona está apoyada
    medc=sumc/num_cabeza
    mede=sume/num_escapulas
    medr=sumr/num_riñones
    medd=sumd/num_pied
    medi=sumi/num_piei
```

Figura 3.62 Función graficas()

A continuación, solo tenemos que escribir los datos previamente calculados, en el Excel, en las celdas que nosotros queramos tal y como se ve en la figura 3.63.

```
#Escribimos los datos en el excel
hoja.cell(row=3,column=9,value="Media")
hoja.cell(row=4,column=8,value="Cabeza")
hoja.cell(row=4,column=9,value=medc[0])
hoja.cell(row=5,column=8,value="Escapulas")
hoja.cell(row=5,column=9,value=mede[0])
hoja.cell(row=6,column=8,value="Riñones")
hoja.cell(row=6,column=9,value=medr[0])
hoja.cell(row=7,column=8,value="Pie izqd")
hoja.cell(row=7,column=9,value=medi[0])
hoja.cell(row=8,column=8,value="Pie drch")
hoja.cell(row=8,column=9,value=medd[0])
```

Figura 3.63Función graficas() (1)

Escritos los datos en la tabla correspondiente, solo nos queda crear el gráfico de barras a partir de estos. Como vemos en la figura 3.64 primero con la función Reference(), seleccionamos de donde se deben tomar los datos, la variable *valores* contendrá el nombre de las columnas y la variable *val*, los valores numéricos de cada una de ellas. Después, solo nos queda crear la gráfica, llamando a la función BarChart() que dibuja gráficos de barras, añadir los datos y establecer los nombres de las diferentes barras.

Por último, indicaremos a partir de que casilla queremos que se dibuje el gráfico.

```
valores=Reference(hoja,min_col=8, min_row=4,max_row=8)
val=Reference(hoja,min_col=9, min_row=4,max_row=8)
graf=BarChart()
graf.add_data(val)
graf.set_categories(valores)
hoja.add_chart(graf, "K4")
```

Figura 3.64 Creación gráfico

Para finalizar, nos queda guardar todos los datos en una hoja de Excel nueva. Para ello, como se muestra en la figura 3.65, crearemos dos arrays uno que contendrá el título de la tabla y otro el nombre de las columnas. Para crear un nuevo Excel debemos llamar a la función *Workbook().*

Posteriormente, para trabajar con una hoja de ese mismo Excel debemos activarla. Ya solo nos quedará introducir los datos. Le ponemos título a la hoja creada. Añadiremos los arrays previamente definidos e iremos recorriendo con un bucle for el contenido del array *tabla_pesos1,* que recordemos contiene los valores de los pesos de las diferentes zonas. Añadidos los datos solo nos queda llamar a las funciones previamente definidas, para aplicar el formato y crear el gráfico de barras. En última instancia, guardamos el Excel con un determinado nombre.

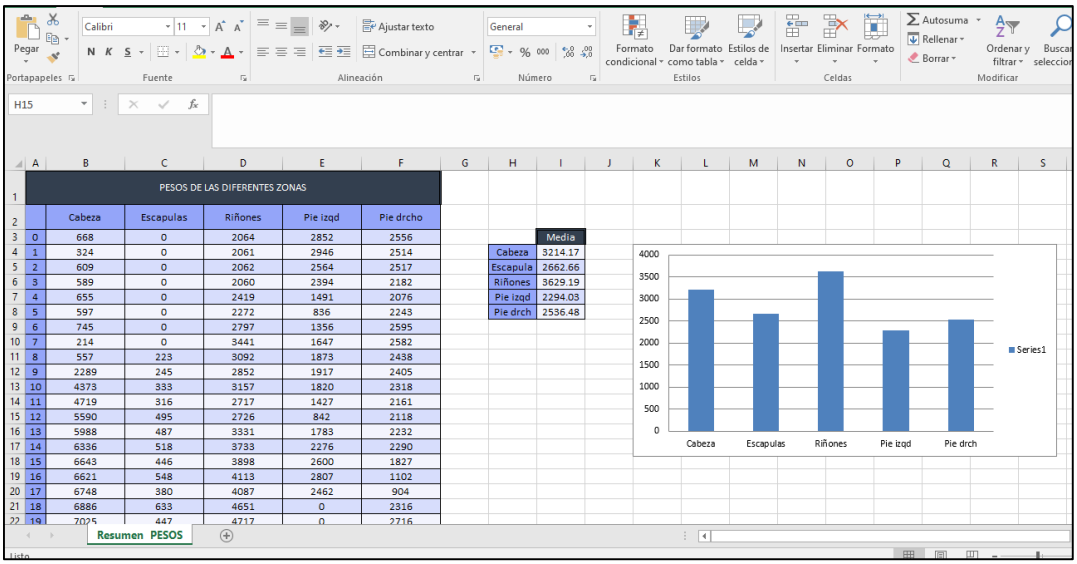
```
header=["PESOS DE LAS DIFERENTES ZONAS"]
columnass=["","Cabeza","Escapulas","Riñones","Pie izqd", "Pie drcho"]
libro1 = Workbook()
nsheet = libro1.active
nsheet.title = "DATOS " + hojas[0]
nsheet.append(header)
nsheet.append(columnass)
for n in range(0,len(tabla_pesos1)):
    nsheet.append(tabla_pesos1[n])
graficas(nsheet)
formato_excel(nsheet)

libro1.save("datosbebes_format.xlsx")
```

Figura 3.65 Creación Excel

Comprobamos que el Excel se ha creado sin problemas, como vemos en la figura 3.66.

Figura 3.66 Archivo Excel

Para finalizar, solo nos queda abrir el fichero y ver que los datos se han guardado correctamente, con el formato adecuado. El Excel completo lo vemos en la figura 3.67.



PESOS DE LAS DIFERENTES ZONAS

	Cabeza	Escapulas	Riñones	Pie izqd	Pie drcho
0	668	0	2064	2852	2556
1	324	0	2061	2946	2514
2	609	0	2062	2564	2517
3	589	0	2060	2394	2182
4	655	0	2419	1491	2076
5	597	0	2272	836	2243
6	745	0	2797	1356	2595
7	214	0	3441	1647	2582
8	557	223	3092	1873	2438
9	2289	245	2852	1917	2405
10	4373	333	3157	1820	2318
11	4719	316	2717	1427	2161
12	5590	495	2726	842	2118
13	5988	487	3331	1783	2232
14	6336	518	3733	2276	2290
15	6643	446	3898	2600	1827
16	6621	548	4113	2807	1102
17	6748	380	4087	2462	904
18	6886	633	4651	0	2316
19	7025	447	4717	0	2216

	Media
Cabeza	3214.17
Escapula	2662.66
Riñones	3629.19
Pie izqd	2294.03
Pie drch	2536.48

Figura 3.67 Excel con formato

3.2.9 Comprobación identificación zonas mediante visualización

Previamente calculamos el área, peso, centros y centros de masa, y velocidad de las diferentes zonas, y en todo momento, una parte importante era la identificación de las zonas, es decir, saber a qué parte del cuerpo pertenecía cada parámetro.

La mejor forma de saber si este proceso se está realizando correctamente es mediante la visualización, simulando una esterilla, los puntos que forman cada zona y denominarlos con el nombre de la parte del cuerpo a la que pertenecen.

Generaremos una animación de tal forma que veremos cómo se mueven las zonas del cuerpo del bebé a medida que pasa el tiempo.

Para ello, haremos uso de la librería *matplotlib*, en concreto de la clase *animation* que nos permitirá animar nuestros gráficos y de la clase pyplot para generar las gráficas.

En primer lugar, tenemos que crear la matriz que le vamos a pasar a la función dedicada a la animación. Podríamos pensar que esta matriz ya la tenemos, pero debemos tener en cuenta que tenemos una matriz que contiene todos los puntos mayores del límite impuesto, pero para que la animación sea posible necesitamos una matriz que tenga las mismas dimensiones que la esterilla y que los valores inferiores al límite valgan cero, ya que los datos que son cero también tenemos que representarlos. Lo que haremos es con un bucle for recorrer la matriz que contiene todas las matrices distintas de cero y poner a cero todos los elementos cuyo valor sea inferior a 150.

Como podemos observar en la figura 3.68 la nueva matriz se llama *matrizdatos* y la obtenemos recorriendo una a una las matrices contenidas en la variable *paquete_datos*.

```
matrizdatos=[]
for i in range(0,len(paquete_datos)):
    datos = np.array(paquete_datos[i]["pressureMatrix"])
    for p in range (0,26):
        for q in range (0,86):
            if(datos[p][q] < 150 ):
                datos[p][q]=0
    matrizdatos.append(datos)
```

Figura 3.68. Matriz para la visualización

Posteriormente, declararemos la figura, gráfica, y sus dimensiones para que tenga dimensiones rectangulares y al visualizarla sea como si estuviésemos viendo la representación de la esterilla misma, con el comando que se muestra en la figura 3.69.

```
fig=plt.figure(figsize=(15, 7))
```

Figura 3.69 Dimensiones gráfica

Ahora declararemos la función que va a realizar la animación, la llamaremos *animate()*. Como vemos en la figura 3.70, en primer lugar llamamos al método plt.clf(), encargado de borrar la figura actual, de esta forma evitaremos que se solapen los datos de figuras diferentes.

El código que aparece a continuación ya se explicó previamente en el apartado 3.2.6.2 Cálculo de los centros y centros de masa. Repetimos este código para evitar mezclar la parte de visualización con el resto de código, de esta forma, si solo quisiéramos visualizar estos datos, sería tan sencillo, como exportar este código a un nuevo archivo Python y ejecutarlo.

```
def animate(i):
    plt.clf()
    zone=0
    pzona=[]#matriz para contar cuantos sensores hay en cada zona
    centro_m=[]
    puntos_zonas=[]
    pzona=[]
    punto_zona=[0,0,0]

    matriz_zona=np.array(paquete_zonas[i])
    for t in range(0,len(matriz_zona)):
        punto=np.array(matriz_zona[t])#leemos un punto de la matriz x,y,zona,valor
        #print(punto)
        if (punto[0]>13):#punto parte superior
            superior[0]=superior[0]+1
        else:
            superior[1]=superior[1]+1
        if(len(pzona) > int(punto[2])):
            pzona[int(punto[2])]=pzona[int(punto[2])]+1
        else:
           zone=zone+1
           pzona.append(1)
        if((len(puntos_zonas)-1)==punto[2] or (len(puntos_zonas)-1)>punto[2]):
            puntos_zonas[int(punto[2])][0]=puntos_zonas[int(punto[2])][0]+punto[0]
            puntos_zonas[int(punto[2])][1]=puntos_zonas[int(punto[2])][1]+punto[1]
            puntos_zonas[int(punto[2])][2]=puntos_zonas[int(punto[2])][2]+punto[3]
        else:
            puntos_zonas.append([int(punto[0]),int(punto[1]),int(punto[3])])
    for a in range(0,(len(pzona))):

        centro=np.array(puntos_zonas[a])
        centro_z[0]=(centro[0]/(int(pzona[a])))#Suma coordenadas X dividido nº sensores de 1 zona
        centro_z[1]=(centro[1]/(int(pzona[a])))

        centro_m.append([centro_z[0],centro_z[1]])
```

Figura 3.70 Código función animate() (1)

En la siguiente parte de la función animate() hacemos nuevamente uso de código ya escrito, para identificar a que parte del cuerpo corresponde cada zona. Para poder visualizar estos valores, guardaremos en la variable texto el nombre de cada zona y posteriormente lo imprimiremos en el gráfico. Para ello, invocaremos al método plt.text() el cual recibe como parámetros las coordenadas donde queremos que aparezca el texto, que texto queremos mostrar y el tamaño del mismo.

En la segunda línea de código de la figura 3.71 hacemos uso del método *plt.title()* gracias al cual indicaremos el título que queremos que tenga nuestro gráfico. En este caso será la fecha y hora de cada matriz.

```
datos=matrizdatos[i]
plt.title(paquete_datos[i]["dateTime"])
for w in range(0,len(centro_m)):
    m=(round(centro_m[w][0]))
    n=(round(centro_m[w][1]))
    for z in range(0,5):
        if((centro_m[w][0]-4<centro_cuerpo[cuerpo[z]][0]<centro_m[w][0]+4 and
            centro_m[w][1]-4<centro_cuerpo[cuerpo[z]][1]<centro_m[w][1]+4)):
            if(z==0):
                texto=("Cabeza")
            if(z==1):
                texto=("Escapulas")
            if(z==2):
                texto=("Riñones")
            if(z==3):
                texto=("Pie Izquierdo" )
            if(z==4):
                texto=("Pie Derecho")
    plt.text(n,m,texto,fontsize=8)
```

Figura 3.71 Código función animate() (2)

Una vez identificadas las zonas, tenemos que crear el gráfico en sí. En este caso usaremos un gráfico de mapa de calor. En este gráfico los datos son visualizados mediante colores, de forma que la variación de ellos vendrá indicada por la variación de colores. En nuestro caso este gráfico es muy útil, ya que gracias al uso del color seremos capaz de observar a simple vista aquellas zonas en las que se está ejerciendo mayor presión, debido a que el color en estas será más oscuro.

Para crear el gráfico, llamaremos al método *sb.heatmap()*, haciendo uso de la librería *seaborn*. A este método le pasamos como parámetros, en primer lugar, los datos, es decir, la matriz que queremos que represente, en segundo lugar, la gama de colores que queremos que tenga el gráfico; se ha escogido una gama de colores que va desde el beige, siendo este el color que representa el valor cero y el azul oscuro el que representa el valor mayor. Después, indicamos el grosor que queremos que tengan las líneas de separación. Luego, los valores mínimo y máximo del eje de color. Por último, indicamos con el comando *square*, que se establezca el aspecto de los ejes de forma igualitaria, de esta forma cada celda tendrá forma cuadrada.

Con el método *plt.xlabel()* indicamos el nombre que deseamos tenga el eje 'x'. También, invertiremos el eje 'y' para que el centro de coordenadas (0,0) esté situado en la esquina inferior izquierda. Por último, colocaremos el eje 'y' vertical con rotación 0. Este código se muestra en la figura 3.72.

```
heat_m = sb.heatmap(datos,cmap="YlGnBu", linewidths=.4, vmin=0, vmax=1500, square=True)
plt.xlabel('Valores de la matriz '+str(i))
plt.gcf().axes[0].invert_yaxis()

heat_m.set_yticklabels(heat_m.get_yticklabels(),rotation=0)
centro_m=[0,0]
```

Figura 3.72 Código función animate() (3)

Por último, deberemos invocar a la función animation.FuncAnimation, la cual recibe como parámetros la figura que creamos, la función encargada de animar la gráfica y por último, el retraso entre los fotogramas en milisegundos, tal y como vemos en la figura 3.73.

```
animacion = animation.FuncAnimation(fig,animate,interval=1000)
plt.show()
```

Figura 3.73 Código función animate() (4)

Finalmente, si ejecutamos todo el código veríamos la animación que aparece en las figuras 3.74 y 3.75. También, podríamos guardar la animación como un vídeo y mostrarlo en una página web, pero esto se realizará en otro TFG diferente.

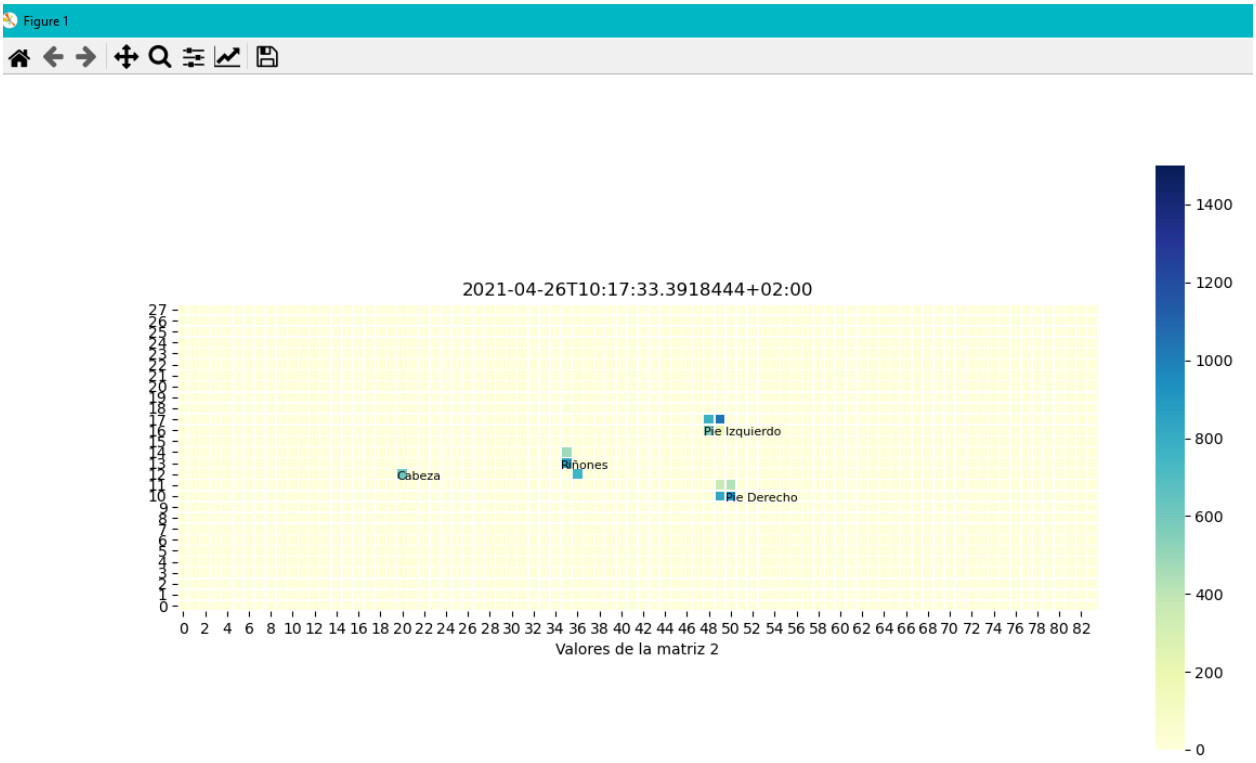


Figura 3.74. Animación Matriz 1

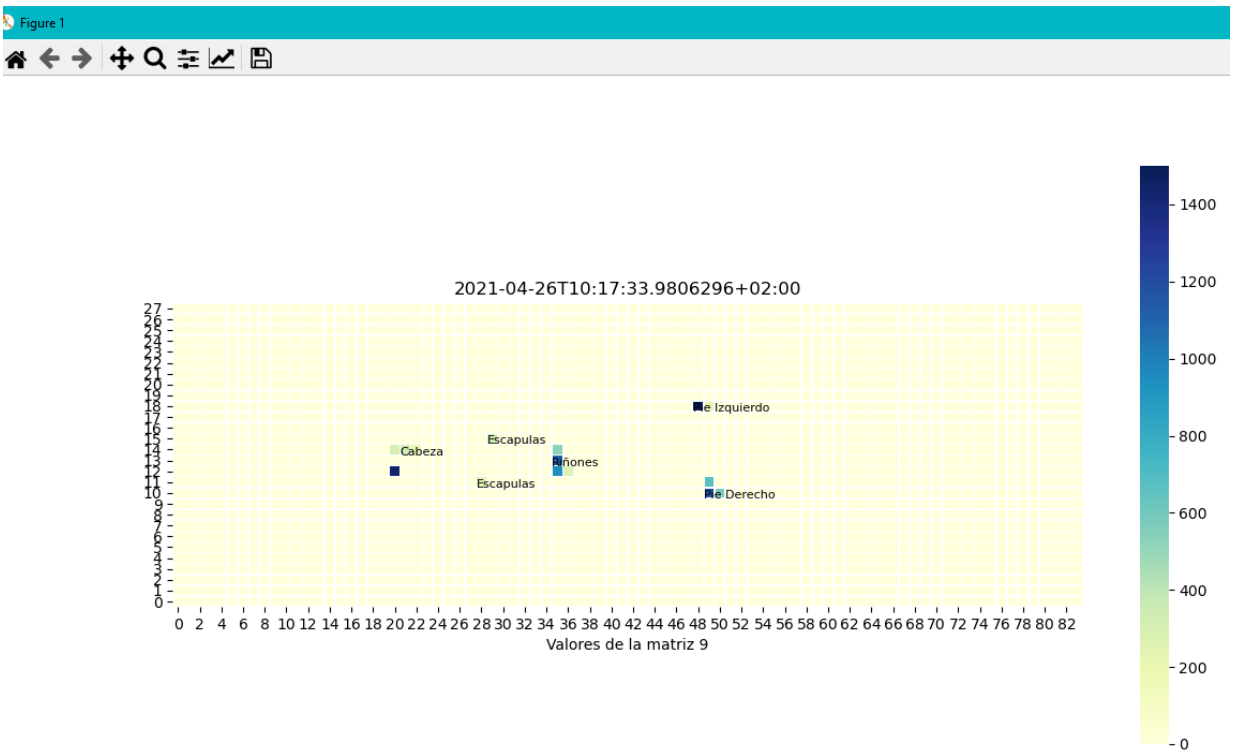


Figura 3.75. Animación Matriz 9

En las figuras 3.74 y 3.75 observamos cómo se identifican las diferentes zonas del cuerpo y como estas varían. Gracias a esto los expertos tienen en todo momento conocimiento de que parte es la que se está moviendo y a su vez la presión aproximada que se está ejerciendo.

4. Desarrollo de la aplicación 2

Al principio de este capítulo, se dijo que haríamos uso de una segunda esterilla, la cual cuenta con una mejor sensibilidad, es decir, es capaz de detectar movimientos más precisos de los pacientes.

Hay que tener en cuenta que esta esterilla se obtuvo la última semana de julio, cuando la mayoría de todo el proyecto ya estaba hecho, es por ello, que el análisis de los datos con esta nueva esterilla será menor, quedando parte del análisis realizado con la primera esterilla como trabajos futuros.

Las muestras tomadas con esta esterilla tampoco son muestras con bebés reales, debemos tener en cuenta la situación actual de covid y el periodo vacacional en el que se está realizando el presente trabajo. Estas muestras se realizaron con la ayuda de un muñeco de características y peso similar al de un bebé y con ayuda de un experto el cual lo movía para simular movimientos similares a los de un bebé real.

Con esta nueva esterilla nos vamos a central principalmente en la identificación de las zonas, lo cual será más complicado, debido a que, en este caso, gracias a la precisión obtenida las zonas no están igual de bien diferenciadas.

Por otro lado, igual que hicimos con la otra esterilla obtendremos el valor del peso de cada zona, ahora representaremos estos valores en un gráfico de barras animado, el cual dará constancia a los expertos de cómo se va distribuyendo el peso entre las diferentes zonas lo largo del experimento.

Como vemos, vamos a tener que crear dos gráficos interactivos diferentes. Por sencillez, ya que la creación de dos gráficos simultáneos animados es algo más elaborada, crearemos dos programas distintos los cual generarán cada uno un gráfico distinto. Estos programas se llamarán heatmap_vis.py y barras_vis.py

Hay que recordar, que estos gráficos, posteriormente, y en otro TFG diferente, serán visualizados y exportados a una página web, por lo que la creación de dos programas diferentes no supone ningún problema.

Dicho todo esto, podemos pasar a explicar el código de cada programa en detalle.

4.1 Programa heatmap_vis.py

En primer lugar, explicaremos el código gracias al cual generaremos un gráfico, en concreto, un mapa de calor o heatmap en el cual veremos cómo se mueven las diferentes partes del cuerpo del bebé.

Creamos una función llamada animar_heatmap(), la cual llamaremos al final del programa, después de la declaración de la propia función, tal y como se ve en la figura 4.1.

```
if __name__ == "__main__":
    animar_heatmap()
```

Figura 4.1. Llamada a la función

En la figura 4.2, podemos ver la declaración de variables, la mayoría de ellas, tienen los mismos nombres y función que en el código de la primera esterilla.

```
zonas=[] #Matriz para guardar las zonas
zona=[0,0,0,0]
zone=0
paquete_zonas=[]
punto_zona=[0,0,0]
puntos_zonas=[]
centro_z=[0,0]#centro de 1 zona
centro_m=[]#todos los centros de 1 matriz
pesos=[]
pzona=[]#matriz para contar cuantos sensores hay en cada zona
pesoss=[]
mm=[]
```

Figura 4.2 Variables

4.1.1 Procesamiento de la matriz de datos

En primer lugar, y como ya se hizo previamente, abriremos el fichero JSON y extraeremos los datos de él. Nos quedaremos únicamente con aquellas matrices cuyos valores sean distintos de cero, de esta forma evitamos analizar matrices sin datos. Este código se muestra en la figura 4.3.

```
def animar_heatmap():
    i=0
    k=0
    json_fname = "C:\\Users\\Luyma\\Downloads\\extraeDatos_v0\\DatosBebe1MOD2.json"

    with open(json_fname) as file:
        data = json.load(file)

    paquete_datos = []
    for i in range (0, len(data["pressureData"])):
        datos = np.array(data["pressureData"][i]["pressureMatrix"])
        # Nos quedamos con las matrices con datos distintos de cero y su fecha
        x = datos.sum()
        if x > 0:
            paquete_datos.append(data["pressureData"][i])
```

Figura 4.3 Abrir json y descartar matrices

En segundo lugar, reduciremos el área de presión, de esta forma, evitaremos detectar interferencias o cualquier apoyo que no corresponda al bebé. Por otro lado, aplicaremos un filtro a las matrices, así haremos que cualquier valor inferior a 80, que es el límite que vamos a imponer valga cero, tal y como vemos en la figura 4.4. Guardaremos las nuevas matrices en la variable *matrizdatos*. Hay que tener en cuenta que este límite es menor que el impuesto con la otra esterilla, ya que la precisión de esta es mayor.

```
matrizdatos=[]
for i in range (0, len(paquete_datos)):
    datos = np.array(paquete_datos[i]["pressureMatrix"])
    for p in range (0,80):
        for q in range (0,80):
            if(q==23 and p==42):
                datos[p][q]=0
            if(q>73 or q<7):
                datos[p][q]=0
            if(p>73 or p<7):
                datos[p][q]=0
            if(datos[p][q] < 80 ):
                datos[p][q]=0
    matrizdatos.append(datos)
```

Figura 4.4 Filtrar matrices

4.1.2. Localizar zonas de presión

Una vez filtrados los datos, vamos a pasar a localizar las diferentes zonas, es decir, localizaremos aquellos puntos distintos de cero y determinaremos a que zona pertenece cada uno de ellos, siendo cada zona por ahora un simple número. El código es muy similar al empleado para localizar las zonas en el apartado 3.2.2 Localización de las diferentes zonas de presión por ello, no se entrará a detallarlo en detalle.

Lo único que cambia es que declaramos 4 nuevas variables que se muestran en la figura 4.5, las cuales representan los máximos y mínimos de las coordenadas ‘x’ e ‘y’.

```
min2 = 50
max2 = 0
min1 = 50
max1 = 0
```

Figura 4.5 Máximos y mínimos

Gracias a estas variables sabremos el punto más a la derecha y el más a la izquierda de la esterilla, es decir, la distancia entre esos 2 puntos representará la longitud del bebé aproximadamente. Todo el código previamente descrito se ve en las figuras 4.6 y 4.7.

```
longcuerpo=[]
for x in range (0,len(matrizdatos)):
    datos = matrizdatos[x]
    zona_x=0
    aux=0
    global zonas
    for p in range (6,74):
        for q in range (6,74):
            if(datos[p][q] != 0):
                if(q < min2):
                    min2=q
                    #min1=p
                if (q > max2):
                    max2=q
                    #max1=p
                if (p < min1):
                    min1=p
                if (p > max1):
                    max1=p
                if (len(zonas)>0):
                    for z in range (0,len(zonas)):
                        punto=np.array(zonas[z])
                        if(int(punto[2])>int(aux)):
                            aux=int(punto[2])

                        if (((p-4) < punto[0] < (p+4)) and ((q-4) < punto[1] < (q+4))):
                            zona[0]=p
                            zona[1]=q
                            zona[2]=int(punto[2])
                            zona[3]=datos[p][q]
                            zonas.append([zona[0],zona[1],punto[2],zona[3]])
                            zone=1
                            break
```

Figura 4.6 Localizar zonas (1)

```
                    if(zone==0):
                        zona[0]=p
                        zona[1]=q
                        zona[2]=int(aux+1)
                        zona[3]=datos[p][q]
                        zonas.append([zona[0],zona[1],zona[2],zona[3]])
                if (len(zonas)==0):
                    zona[0]=p
                    zona[1]=q
                    zona[2]=0
                    zona[3]=datos[p][q]
                    zonas.append([zona[0],zona[1],zona[2],zona[3]])
                zone=0
    paquete_zonas.append(zonas)
    zonas=[]
    longcuerpo.append([max2,min2,max1,max2])
```

Figura 4.7 Localizar zonas (2)

Los datos de los máximos y los mínimos los almacenamos en la variable *longcuerpo* para utilizarlos posteriormente.

Después, declaramos la figura donde vamos a visualizar el mapa de calor como vemos en la figura 4.8.

```
mapa = plt.figure(figsize=[9,11])
```

Figura 4.8 Figura

4.1.3 Función anímate(i)

Ahora declararemos la función anímate, que será la encargada de actualizar los datos para que el mapa de color cambie con las diferentes muestras. En la figura 4.9 vemos la declaración de la función y de determinadas variables. A su vez, debemos escribir el comando *plt.clf()* para que la figura anterior se borre y se cree una nueva, de esa forma la animación se generará creando una imagen detrás de otra.

```
def animate(i):
    plt.clf()
    global puntos_zonas
    global zonas
    global centro_mm
    global puntos_zonas
    global centro_m
    global pzona
    global pesoss
    zone=0
    pzona=[]#matriz para contar cuantos sensores hay en cada zona
    centro_m=[]
    puntos_zonas=[]
    pzona=[]
    punto_zona=[0,0,0]
    zonass=[]
    j = i
    zone=0
    datoss=matrizdatos[j]
    mm=np.array(paquete_zonas[j])
```

Figura 4.9 Declaración función y variables

4.1.4 Cálculo de los centros de cada zona

Lo primero que haremos dentro de esta función es calcular el centro de cada zona, lo cual nos ayudará a identificarlas posteriormente. El código que se muestra en la figura 4.10, es exactamente el mismo que el empleado en el apartado 3.2.6.2 Cálculo de los centros y centros de masa, por ello, no se volverá a detallar.

```
#para calcular el centro de cada zona
for t in range(0,len(mm)):
    punto=np.array(mm[t])#leemos un punto de la matriz x,y,zona,valor
    if(len(pzona) > int(punto[2])):
        pzona[int(punto[2])]=pzona[int(punto[2])]+1
    else:
       zone=zone+1
       pzona.append(1)
    if((len(puntos_zonas)-1)==punto[2] or (len(puntos_zonas)-1)>punto[2]):
        puntos_zonas[int(punto[2])][0]=puntos_zonas[int(punto[2])][0]+punto[0]
        puntos_zonas[int(punto[2])][1]=puntos_zonas[int(punto[2])][1]+punto[1]
        puntos_zonas[int(punto[2])][2]=puntos_zonas[int(punto[2])][2]+punto[3]
    else:
        puntos_zonas.append([int(punto[0]),int(punto[1]),int(punto[3])])

for a in range(0,(len(pzona))):
    centro=np.array(puntos_zonas[a])
    centro_z[0]=(centro[0]/(int(pzona[a])))#Suma coordenadas X
                                          #dividido nº sensores de 1 zona
    centro_z[1]=(centro[1]/(int(pzona[a])))
    centro_m.append([centro_z[0],centro_z[1]])
```

Figura 4.10 Cálculo centros zonas

Calculados los centros de cada zona podemos empezar a identificar a que parte del cuerpo pertenece cada una de las zonas detectadas.

4.1.5 Identificación de las diferentes zonas de presión

En este caso, gracias a la precisión de la esterilla vamos a ser capaces de detectar más zonas como, por ejemplo, la espalda completa del bebé o sus manos. Es por ello, que vamos a distinguir 3 zonas principales las cuales denominaremos “Cabeza”, “medio” y “Pie”.

En primer lugar, iremos recorriendo la matriz que contiene los datos de los centros previamente calculados y viendo a que zona de las anteriormente mencionadas pertenece cada uno. Como vemos en el código de la figura 4.11, calculamos en primera instancia el punto medio del cuerpo, teniendo en cuenta la longitud del sujeto, para ello, hacemos uso de los datos guardados en la variable *longcuerpo* que recordemos contenía los valores máximos y mínimos de las coordenadas ‘x’ e ‘y’. Así para averiguar el punto medio del cuerpo calcularemos la distancia entre ambos extremos dividiremos por 2 para hallar la mitad y después deberemos sumar la coordenada ‘x’ mínima y así tendremos el punto medio real.

```
zones=[]
z_cab=0
z_medio=0
z_pie=0

for w in range(0,len(centro_m)):
    m=((centro_m[w][0]))
    n=((centro_m[w][1]))
    mediocuerpo=((longcuerpo[i][0]-longcuerpo[i][1])/2)+longcuerpo[i][1]
```

Figura 4.11 Punto medio cuerpo

Después, para saber a qué zona pertenece cada centro compararemos para el caso de la cabeza si la coordenada 'x' del centro está en un radio de 6 con respecto a la coordenada 'x' máxima del sujeto, es decir, el extremo de la cabeza. Para la zona de los pies, comprobaremos si la coordenada 'x' del centro está en un radio de 6 con respecto a la coordenada 'x' mínima del sujeto, que corresponde con los pies, por ahora no diferenciaremos entre ambos pies.

Por último, nos falta saber si los centros pertenecen a la zona media del cuerpo del paciente. Para ello, veremos si la coordenada 'x' del centro está, en este caso, en un radio de 10 respecto al punto medio del cuerpo. Aumentamos el radio ya que la zona media del cuerpo es la más extensa. Cada vez que detectemos que el centro de una zona corresponde con alguna de las zonas, guardaremos en la variable *zones* el número de la zona a la que pertenece, el nombre de la zona y sus coordenadas 'x' e 'y'.

El código referente a esta parte se muestra en la figura 4.12.

```
for w in range(0,len(centro_m)):
    m=((centro_m[w][0]))
    n=((centro_m[w][1]))
    mediocuerpo=((longcuerpo[i][0]-longcuerpo[i][1])/2)+longcuerpo[i][1]
    if(centro_m[w][1]-6<longcuerpo[i][0]<centro_m[w][1]+6):
        zones.append([w,"Cabeza",m,n])
        z_cab=z_cab+1
    elif(centro_m[w][1]-6<longcuerpo[i][1]<centro_m[w][1]+6):
        zones.append([w,"Pie",m,n])
        z_pie=z_pie+1
    elif(centro_m[w][1]-10<mediocuerpo<centro_m[w][1]+10):
        zones.append([w,"medio",m,n])
        z_medio=z_medio+1
    else:
        zz=0
```

Figura 4.12 Identificación zonas

Identificadas en primera instancia las zonas de forma general, vamos a pasar a identificar más en detalle las diferentes zonas.

Primero localizaremos aquellas zonas que previamente identificamos como "Cabeza" y las guardaremos en la variable *cabb1*.

Debemos tener en cuenta que en esta zona como máximo vamos a identificar 2 zonas, las correspondientes a la cabeza propiamente dicha y al cuello. Supondremos que si solo detectamos una zona está corresponderá a la cabeza y que si, por el contrario, localizamos dos zonas la que tenga mayor coordenada 'x' pertenecerá a la cabeza. Para que se muestren los nombres de las diferentes zonas en el gráfico debemos hacer uso de la función plt.text(), la cual recibe como parámetros coordenadas 'x' e 'y' donde situar el texto, el texto en sí y el tamaño del mismo. Todo esto se muestra en la figura 4.13.

```
#La primera zona tendrá como máximo dos zonas: cabeza y cuello
cabb=0
cabb1=[]
for a in range(0,len(zones)):
    if(cabb==z_cab):
        break
    if(zones[a][1]=="Cabeza"):
        cabb=cabb+1
        cabb1.append(zones[a])
if(len(cabb1)==1):
    texto=("Cabeza")
    plt.text(cabb1[0][3],cabb1[0][2],texto,fontsize=8)
else:
    if(len(cabb1)==0):
        cabb=0
    else:
        if(cabb1[0][3]>cabb1[1][3]):
            texto=("Cabeza")
            plt.text(cabb1[0][3],cabb1[0][2],texto,fontsize=8)
        else:
            texto=("Cuello")
            plt.text(cabb[1][3],cabb1[1][2],texto,fontsize=8)
```

Figura 4.13 Zonas cabeza

Ahora pasamos a la zona media. El código será muy similar al de la zona de la cabeza y por ende al de la zona de los pies. La diferencia es que en este caso podríamos localizar hasta 5 zonas, siendo estas la zona de las escapulas, la espalda, la parte baja de la espalda y ambas manos.

Para distinguir unas zonas de otras tendremos en cuenta el punto medio del cuerpo, es decir, la mitad del cuerpo según la coordenada ‘x’ y los máximos y mínimos de la coordenada ‘y’. En primer lugar, si solo detectamos una zona sabemos que esta corresponderá a la zona de la espalda, ya que es imposible que el bebé no tenga esta parte del cuerpo apoyada. Si localizamos más de una zona deberemos pasar a analizar la posición de su centro.

Si se trata de la zona de la espalda, el centro deberá estar en un radio de 3, respecto a este punto y a su vez cumplir que la coordenada ‘y’ de este centro, esté 5 por encima del mínimo y cinco por debajo del máximo.

En el caso de las escapulas comprobaremos si la coordenada ‘x’ es mayor que el punto medio del cuerpo más cinco, es decir, este en la parte superior, y que a su vez la coordenada ‘y’ esté al igual que la zona de la espalda 5 por encima del mínimo y cinco por debajo del máximo.

Para la parte baja de la espalda será lo mismo que para la zona de las escápulas, solo que en este caso comprobaremos que la coordenada ‘x’ es inferior a la mitad del cuerpo menos 5, para asegurarnos que está en la parte inferior.

Para las dos últimas zonas que son las manos, comprobaremos que para la mano izquierda la coordenada ‘y’ sea menor o igual que el mínimo de la coordenada ‘y’ más 4, y para la mano izquierda al revés, que la coordenada ‘y’ sea mayor o igual que el máximo más 4. Todo esto se refleja en el código de la figura 4.14.

```
#La segunda zona tendrá como máximo 5 zonas:
#Espalda, las escapulas, parte baja espalda, y las manos
medd=0
medd1=[]

for a in range(0,len(zones)):
    if(medd1==z_medio):
        break
    if(zones[a][1]=="medio"):
        medd=medd+1
        medd1.append(zones[a])
if(len(medd1)==1):
    texto=("Espalda")
    plt.text(medd1[0][3],medd1[0][2],texto,fontsize=8)
else:
    if(len(medd1)==0):
        medd=0
    else:
        for o in range(0,len(medd1)):
            if(medd1[o][3]-3<mediocuerpo<medd1[o][3]+3 and longcuerpo[i][3]+5<medd1[o][2]<longcuerpo[i][2]-5):
                texto=("Espalda")
                plt.text(medd1[o][3],medd1[o][2],texto,fontsize=8)
            elif(medd1[o][3]>mediocuerpo+5 and longcuerpo[i][3]+5<medd1[o][2]<longcuerpo[i][2]-5):
                texto=("Escapulas")
                plt.text(medd1[o][3]+4,medd1[o][2]+2,texto,fontsize=8)
            elif(medd1[o][3]<mediocuerpo-5 and longcuerpo[i][3]+5<medd1[o][2]<longcuerpo[i][2]-5):
                texto=("Parte baja espalda")
                plt.text(medd1[o][3]-4,medd1[o][2]+3,texto,fontsize=8)
            elif(medd1[o][2]<=longcuerpo[i][3]+4):
                texto=("Mano izrda")
                plt.text(medd1[o][3]+2,medd1[o][2],texto,fontsize=8)
            elif(medd1[o][2]>=longcuerpo[i][2]-4):
                texto=("Mano drcha")
                plt.text(medd1[o][3]+2,medd1[o][2],texto,fontsize=8)
```

Figura 4.14 Zonas parte media

Por último, solo nos faltaría identificar la zona de los pies, es decir, cual es el derecho y cual el izquierdo. Esta es la parte más sencilla, ya que si solo localizamos una zona tendremos que ver si su coordenada ‘y’ es mayor que la mitad del cuerpo o menor. Y si localizamos dos zonas tendremos que comprobar cuál de ambas coordenadas ‘y’ es mayor, la cual corresponderá al pie derecho. Todo esto lo podemos ver en la figura 4.15.

```
#La tercera zona tendrá como máximo 2 zonas: los pies
piee=0
piee1=[]
for a in range(0,len(zones)):
    if(piee==z_pie):
        break
    if(zones[a][1]=="Pie"):
        piee=piee+1
        piee1.append(zones[a])
if(len(piee1)==1):
    if(piee1[0][3]>mediocuerpo):
        texto=("Pie drcho")
        plt.text(piee1[0][3]+1,piee1[0][2]-2,texto,fontsize=8)
    else:
        texto=("Pie izrdo")
        plt.text(piee1[0][3]+1,piee1[0][2]-2,texto,fontsize=8)
else:
    if(len(piee1)==0):
        piee=0
    else:
        if(piee1[0][2]>piee1[1][2]):
            texto=("Pie drcho")
            #print("pd")
            plt.text(piee1[0][3]+1,piee1[0][2]-2,texto,fontsize=8)
            texto=("Pie izrdo")
            plt.text(piee1[1][3]+1,piee1[1][2]-2,texto,fontsize=8)

        else:
            texto=("Pie izrdo")
            #print("pi")
            plt.text(piee1[0][3],piee1[0][2],texto,fontsize=8)
            texto=("Pie drcho")
            #print("pd")
            plt.text(piee1[1][3],piee1[1][2],texto,fontsize=8)
```

Figura 4.15 Zonas pies

4.1.6 Dibujando el mapa de calor

Para finalizar con este código, solo nos quedaría dibujar o crear el gráfico animado gracias al cual podremos visualizar como se van moviendo las diferentes zonas previamente localizadas.

Como título del gráfico pondremos la fecha y hora de cada muestra. Al igual que hicimos en el código de la otra esterilla, crearemos el mapa de calor con el comando sb.heatmap que hace uso de la librería Seaborn. Invertiremos y rotaremos los ejes de coordenadas para situar el punto (0,0) en la esquina inferior izquierda y el título del eje 'x' será el texto "Valores de la matriz" seguido del número de la muestra que se está visualizando en ese instante.

Debemos tener en cuenta, que tenemos que vaciar los arrays que hemos usado previamente, ya que agregamos los datos a estos.

Por último, tenemos que llamar a la función encargada de realizar la animación, a la cual le pasamos como argumentos la figura que creamos anteriormente, la función encargada de ir actualizando los datos y el intervalo entre las muestras. También debemos llamar al comando plt.show() para que se muestren los gráficos, como vemos en la figura 4.16.

```
    plt.title(paquete_datos[j]["dateTime"])
    heat_m = sb.heatmap(datoss,cmap="YlGnBu", vmin=0, vmax=1400, linewidths=.4, square=True)
    plt.gcf().axes[0].invert_yaxis()
    plt.xlabel('Valores de la matriz '+str(j))
    heat_m.set_yticklabels(heat_m.get_yticklabels(),rotation=0)


    zonas=[]
    puntos_zonas=[]
    centro_m=[0,0]
    pzona=[]
    pesoss=[]
animacion = animation.FuncAnimation(mapa,animate, interval=500)
plt.show()
```

Figura 4.16 Dibujar gráfico

Como se observa en la figura anterior, la función *animation* y *plt.show*() aparecen más a la izquierda que el resto de las líneas ya que esta parte del código no forma parte de la función animate, pero sí de la función principal *animar_heatmap*().

4.1.7 Visualización de los resultados

Escrito todo el código, solo nos falta ejecutarlo para poder ver la animación del mapa de calor. A continuación, se van a mostrar una serie de figuras para poder ver la animación y como se van moviendo las zonas y la identificación de las mismas.

En las figuras 4.17, 4.18, 4.19, 4.20 y 4.21 podemos ver los resultados de la animación del mapa de calor.

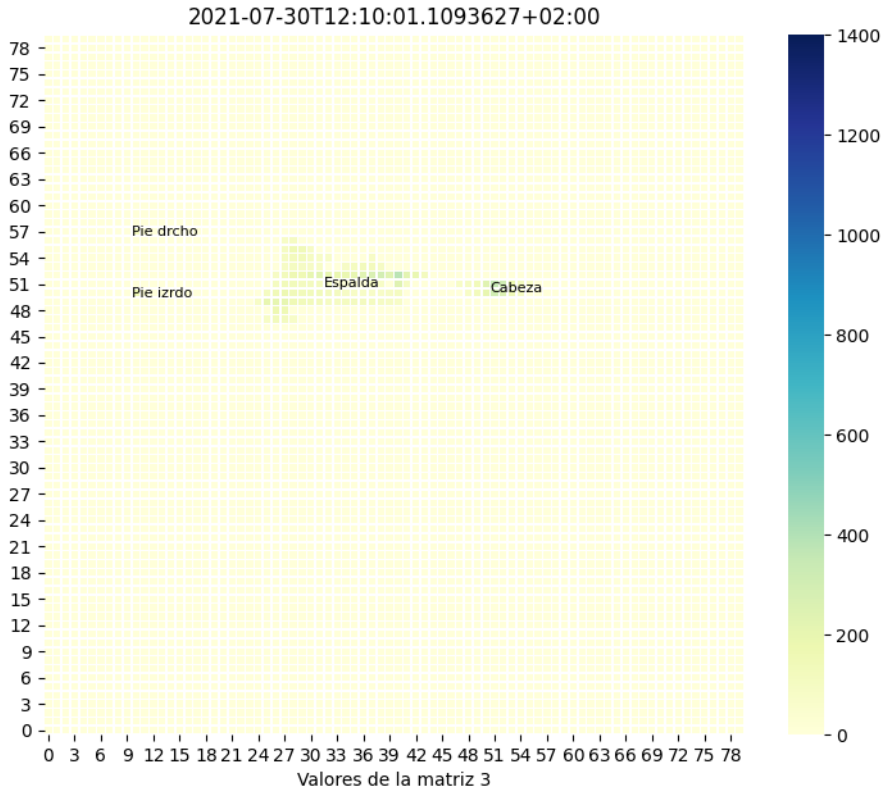


Figura 4.17 Mapa de calor 1

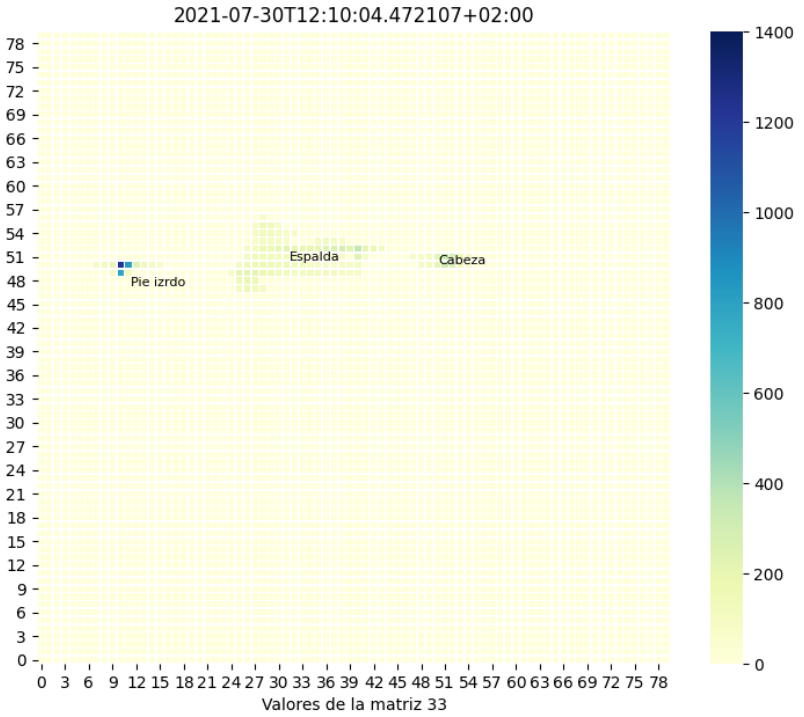


Figura 4.18 Mapa de calor 2

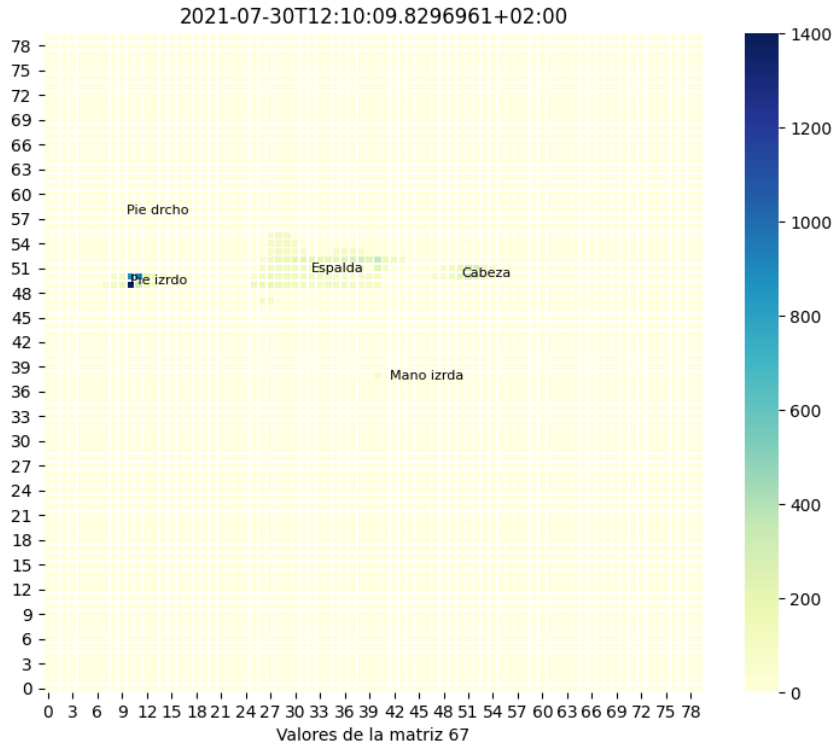


Figura 4.19 Mapa de calor 3

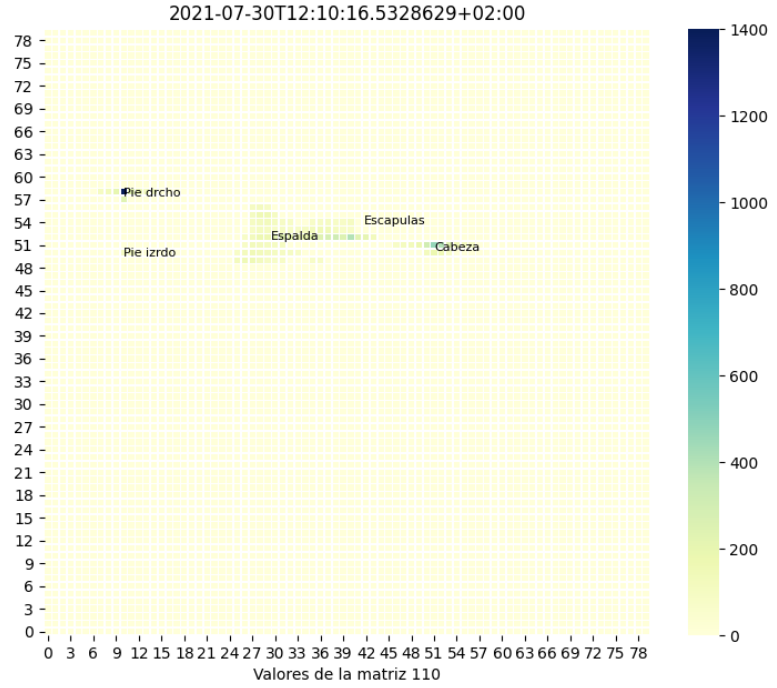


Figura 4.20 Mapa de calor 4

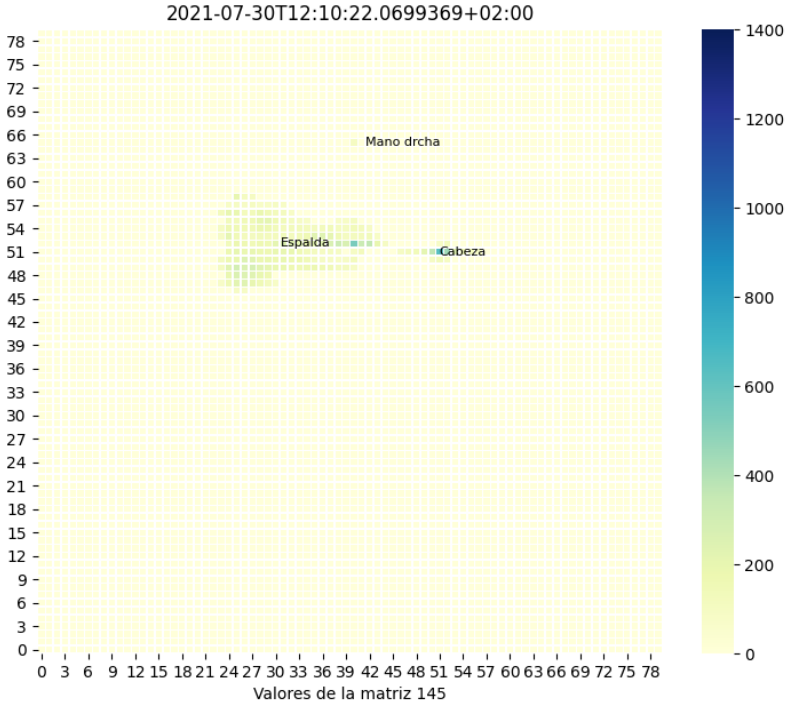


Figura 4.21 Mapa de calor 5

4.2 Programa barras_vis.py

Este fichero será el encargado de generar un gráfico de barras el cual se irá moviendo según vayan variando los pesos de las diferentes zonas. El código principal es el mismo

que el del fichero previamente explicado, por ello, no se volverá a detallar ni se adjuntarán figuras sobre él, para ahorrar espacio y no repetir.

Como se ha dicho previamente, este código es básicamente el mismo que el creado para la generación del mapa de calor lo único que varía en este caso es que debemos guardar el nombre y número de cada zona en diferentes arrays para poder posteriormente dibujar el gráfico correctamente.

En la figura 4.22 vemos un código similar al de la figura 4.13 lo único que varía es que se añaden las líneas que aparecen en amarillo, gracias a las cuales guardamos el nombre y número de cada zona.

```
if (len(zones)==1):
    texto=("Espalda")
    zonass.append(texto)
    lugar.append(zones[0][0])
else:
    #La primera zona tendrá como máximo dos zonas: cabeza y cuello
    #print(z_cab, z_medio, z_pie)
    cabb=0
    cabb1=[]
    lugar=[]
    lug=0
    for a in range(0,len(zones)):
        if(cabb==z_cab):
            break
        if(zones[a][1]=="Cabeza"):
            cabb=cabb+1
            cabb1.append(zones[a])
    if(len(cabb1)==1):
        texto=("Cabeza")
        zonass.append(texto)
        lugar.append(cabb1[0][0])

    else:
        if(len(cabb1)==0):
            cabb=0
        else:
            if(cabb1[0][3]>cabb1[1][3]):
                texto=("Cabeza")
                zonass.append("Cabeza")
                lugar.append(cabb1[0][0])
            else:
                texto=("Cuello")
                zonass.append("Cuello")
                lugar.append(cabb1[1][0])
```

Figura 4.22 Zona cabeza

Como podemos ver en la figura 4.23 para identificar las zonas de la zona media hacemos exactamente lo mismo que en la figura anterior.

```
for a in range(0,len(zones)):
    if(medd1==z_medio):
        break
    if(zones[a][1]=="medio"):
        medd=medd+1
        medd1.append(zones[a])
if(len(medd1)==1):
    texto=("Espalda")
    zonass.append("Espalda")
    lugar.append(medd1[0][0])
else:
    if(len(medd1)==0):
        medd=0
    else:
        for o in range(0,len(medd1)):
            if(medd1[o][3]-3<mediocuerpo<medd1[o][3]+3 and longcuerpo[i][3]+5<medd1[o][2]<longcuerpo[i][2]-5):
                texto=("Espalda")
                zonass.append(texto)
                lugar.append(medd1[o][0])
            elif(medd1[o][3]>mediocuerpo+5 and longcuerpo[i][3]+5<medd1[o][2]<longcuerpo[i][2]-5):
                texto=("Escapulas")
                zonass.append(texto)
                lugar.append(medd1[o][0])
            elif(medd1[o][3]<mediocuerpo-5 and longcuerpo[i][3]+5<medd1[o][2]<longcuerpo[i][2]-5):
                texto=("Parte baja espalda")
                zonass.append(texto)
                lugar.append(medd1[o][0])
            elif(medd1[o][2]<=longcuerpo[i][3]+4):
                texto=("Mano izrda")
                zonass.append(texto)
                lugar.append(medd1[o][0])
            elif(medd1[o][2]>=longcuerpo[i][2]-4):
                texto=("Mano drcha")
                zonass.append(texto)
                lugar.append(medd1[o][0])
```

Figura 4.23 Zona media

Para la zona de los pies es exactamente igual como podemos ver en la figura 4.24.

```
for a in range(0,len(zones)):
    if(piee==z_pie):
        break
    if(zones[a][1]=="Pie"):
        piee=piee+1
        piee1.append(zones[a])
if(len(piee1)==1):
    if(piee1[0][3]>mediocuerpo):
        texto=("Pie drcho")
        lugar.append(piee1[0][0])
        zonass.append(texto)
    else:
        texto=("Pie izrdo")
        zonass.append(texto)
        lugar.append(piee1[0][0])
else:
    if(len(piee1)==0):
        piee=0
    else:
        if(piee1[0][2]>piee1[1][2]):
            texto=("Pie drcho")
            zonass.append(texto)
            lugar.append(piee1[0][0])
            lugar.append(piee1[1][0])
            texto=("Pie izrdo")
            zonass.append(texto)

        else:
            texto=("Pie izrdo")
            zonass.append(texto)
            texto=("Pie drcho")
            zonass.append(texto)
            lugar.append(piee1[0][0])
            lugar.append(piee1[1][0])
```

Figura 4.24 Zona pies

Antes de poder crear el gráfico tenemos que ordenar los pesos en función del orden que se han identificado las zonas, para que a la hora de crear el gráfico el nombre de las columnas corresponda con el valor de los pesos, tal y como se ve en la figura 4.25.

```
t=0
pesos_ord=[]

for t in range(0,len(lugar)):
    pesos_ord.append(pesoss[lugar[t]])
```

Figura 4.25 Ordenar zonas

Ahora ya podemos crear el gráfico de barras. Definiremos una serie de variables como son el número de columnas, el ancho de ambas y el color. Para definir el gráfico llamaremos a la función plt.bar() que recibe como argumentos el número de columnas, que se corresponderá con el número de zonas localizadas, los datos de los pesos, el ancho de las columnas y el color. Para definir el color haremos uso de la función color_palette de la librería seaborn. Por último, definimos el nombre de los ejes como vemos en la figura 4.26.

```
num_group=len(zonass)
ind_barras=np.arange(num_group)
width_barras=0.35
palette = list(reversed(sb.color_palette("husl", len(zonass)).as_hex()))
plt.bar(np.arange(num_group),(pesos_ord),width=width_barras,color=palette)
plt.xticks(ind_barras,zonass)
plt.ylabel("Valores pesos")
plt.xlabel("Zonas"+str(i))
```

Figura 4.26 Gráfico de barras

Para terminar, al igual que antes vaciamos los diferentes arrays y llamamos a la función encargada de animar el gráfico de barras como vemos en la figura 4.27.

```
    zonas=[]
    zonass=[]
    puntos_zonas=[]
    centro_m=[0,0]
    pzona=[]
    pesoss=[]

animacion = animation.FuncAnimation(mapa,animate, interval=1000)
plt.show()
```

Figura 4.27 Animar gráfico

4.2.1 Visualización de los resultados

Ya solo nos queda comprobar que efectivamente el código previamente escrito es correcto, compilando y observando los resultados. Al igual que antes al tratarse de una animación se expondrán diferentes imágenes las cuales mostrarán como se va moviendo el gráfico de barras que se ha creado.

En las figuras 4.28, 4.29, 4.30 y 4.31 podemos observar la animación del gráfico de barras. Vemos como en la primera figura el peso de la zona de la espalda ronda los

12000, es decir, es la zona con mayor peso como cabría esperar; y en la segunda figura vemos como este peso es ligeramente inferior al aumentar el peso de los pies, así comprobamos la distribución de pesos.

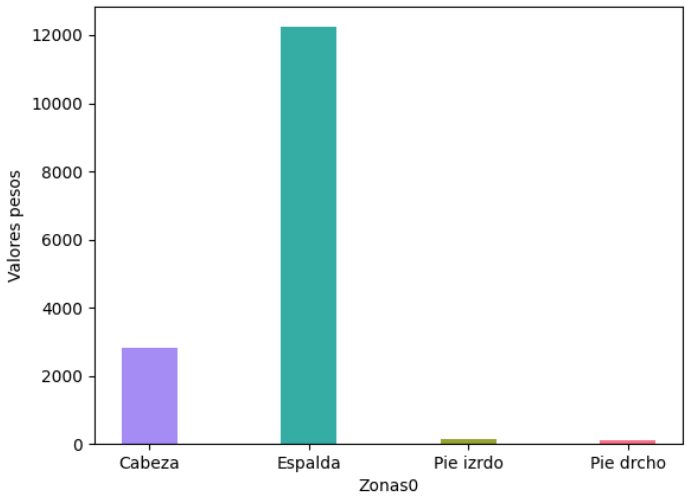


Figura 4.28 Gráfico de barras 1

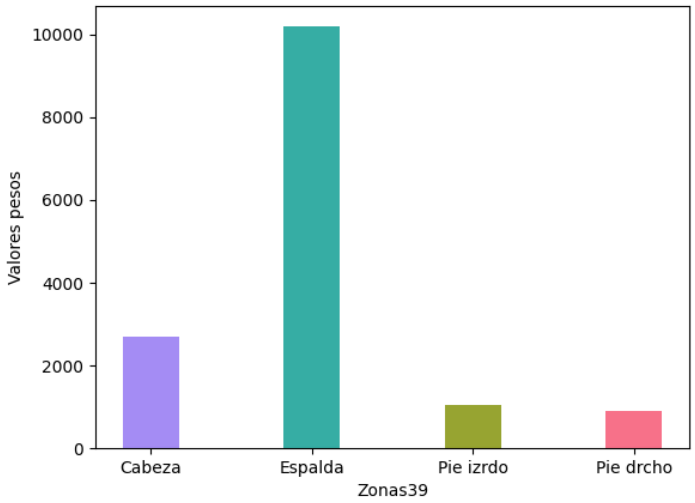


Figura 4.29 Gráfico de barras 2

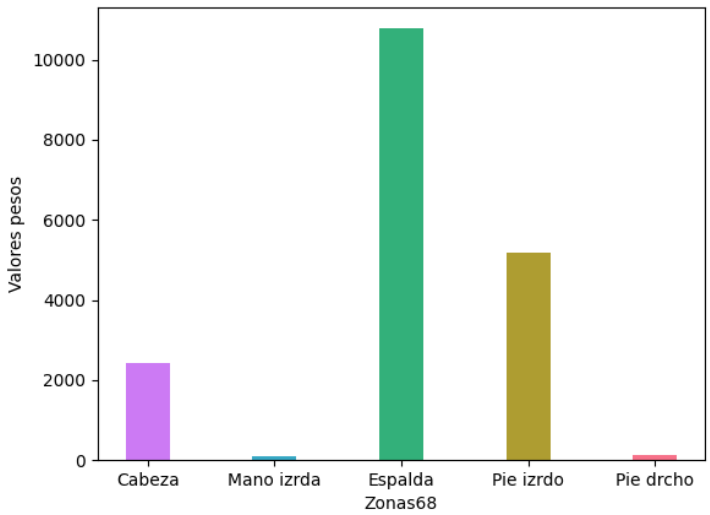


Figura 4.30 Gráfico de barras 3

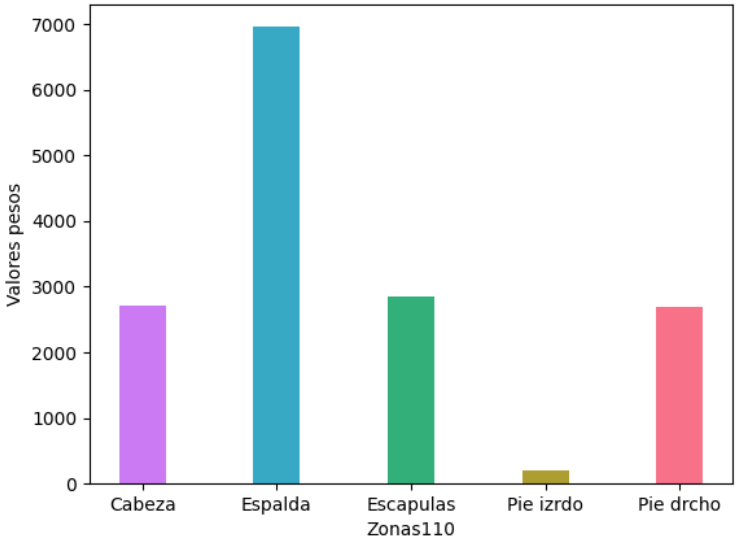


Figura 4.31 Gráfico de barras 4

5. Conclusiones y trabajos futuros

Para finalizar con el presente proyecto se expondrán las conclusiones del mismo, así como los beneficios que supone disponer de una herramienta capaz de detectar e identificar los movimientos de un bebé.

Por otro lado, se comentará brevemente las posibles líneas futuras de trabajo, como poder mejorar y añadir nuevas funcionalidades a las ya presentes.

5.1 Conclusiones

Gracias a un lenguaje de programación como Python hemos sido capaces de analizar los datos obtenidos, almacenarlos en un fichero Excel, e incluso de representarlos. Podríamos haber empleado otros lenguajes para ello, pero las librerías que conforman Python y su sencillez a la hora de programar hacen de este lenguaje la mejor opción.

En este proyecto hemos creado unas herramientas capaces de detectar los movimientos de un bebé, con dos esterillas de presión diferentes, encargadas de recoger y almacenar los datos. Gracias a esto los expertos pueden ser conscientes de que zonas tienen mayor apoyo, del peso que ejerce cada una de ellas, del centro de cada zona, así como del área que estas abarcan y velocidad.

Debemos tener en cuenta que, a edades tan tempranas, como los primeros meses de vida, el diagnóstico de enfermedades y más aquellas relacionadas con el movimiento sea hace especialmente. Por ello, a partir de toda la información calculada, los profesionales podrán ser capaces de sacar conclusiones más precisas acerca de la salud motriz de los pacientes.

El hecho que los expertos puedan observar casi en tiempo real los movimientos de los bebes supone un gran avance. Ya que hasta ahora solo podían valerse de la información que ellos mismos observaran y pequeños detalles. Ahora gracias a toda la información analizada, podrán ser capaces de ver cuáles son las zonas principales de apoyo del paciente, ver como varía la presión o peso que se ejerce en ellas

5.2 Trabajos futuros.

Las líneas de trabajo futuras son muy amplias en este campo. Si tenemos en cuenta que las esterillas empleadas pueden ser usadas no son en pacientes de pocos meses de vida, si no de diferentes rangos de edad. Se podrían crear numerosas herramientas para el análisis de patrones de movimiento, que ayuden en la detección de enfermedades motrices, en rehabilitaciones de pacientes con problemas de movilidad o incluso en aplicaciones de detección del sueño.

En nuestro caso, en primer lugar, como trabajos futuros tendríamos la ampliación del análisis realizado con la segunda esterilla. Tenemos que recordar, que debido al

momento en el que se recibió dicha esterilla, el análisis no ha podido realizarse tan en profundidad como con la primera esterilla. Así pues, quedaría calcular centros de masas, áreas y velocidades de las diferentes zonas. Por supuesto, se podrían incluir nuevas funcionalidades que hagan que el análisis sea lo más completo posible.

Como otro posible trabajo futuro, queda por supuesto el hecho de poder realizar pruebas con bebés reales, de esta forma se podrán validar los datos obtenidos, pudiendo mejorar en gran medida los resultados en función de estas pruebas.

En algún momento de este TFG, se comentó que este trabajo está estrechamente realizado con otro encargado de la visualización y representación de los datos, los cuales serán almacenados en una base de datos y posteriormente exportados a una página web para su visualización. Por ello, no especificamos esto como posibles trabajos futuros ya que esto ya se está realizando.

Como vemos gracias a una completa herramienta como son las esterillas de presión podríamos ser capaces de crear numerosas aplicaciones, con diferentes funcionalidades, las cuales tienen como objetivo principal la de servir de ayuda en el ámbito de la medicina.

Bibliografía

- C. Mlebron. “Como hacer un mapa de calor con Excel”. análisis-web.es [Online] https://analisis-web.es/mapa-de-calor-excel/
- (2020) “Fórmula de la distancia geométrica entre dos puntos (geometría)”. geometriaanalitica.info. [Online] https://www.geometriaanalitica.info/wp-content/uploads/2020/09/distancia-entre-dos-puntos-formula-geometria.jpg
- “Introducción a la librería Matplotlib de Python”. programación.net [Online] https://programacion.net/articulo/introduccion_a_la_libreria_matplotlib_de_python_1599
- A. Sánchez Alberca (2020) “La librería Numpy”. aprendeconalf.es [Online] https://aprendeconalf.es/docencia/python/manual/numpy/
- D. Rodríguez (2020, julio 20) “Visuzalización de datos en Python con Seaborn”. analyticslane.com [Online] https://www.analyticslane.com/2018/07/20/visualizacion-de-datos-con-seaborn/
- A. Sánchez Alberca (2020) “La librería Pandas”. aprendeconalf.es [Online] https://aprendeconalf.es/docencia/python/manual/pandas/
- P. Amoroso “Python books free to read online or download”. github.com [Online] https://github.com/pamoroso/free-python-books

A. Anexo 1 - Manual instalación esterilla Fitness Mat

A continuación, se van a detallar los pasos a seguir para instalar y configurar la esterilla que se va a utilizar.

1. Sacar el contenido de la bolsa de transporte.

2. Retirar las correas de velcro y desenrollar la esterilla.

3. Colocar la esterilla sobre una superficie plana.

4. Cargar la esterilla con el cable USB que se proporciona y un transformador USB compatible durante al menos dos horas.

5. Encender la esterilla usando el botón situado en el lateral de la parte superior de la esterilla. Un led blanco comenzará a parpadear.

Seguidos estos pasos ya tendremos nuestra esterilla en funcionamiento. Una vez hecho esto, procederemos a instalar el software de prueba que provee el fabricante.

En primer lugar, se recomienda que nuestro ordenador disponga de una serie de características, con el fin de no tener problemas con el software, como disponer de Intel Core i5, Windows 10, 4GB de memoria RAM, USB y conectividad bluetooth. Los pasos a seguir son los siguientes:

1. Instalar el software de demostración como administrador para poder utilizar todas las características del sistema. El archivo de instalación es proporcionado a través de un correo electrónico posterior a la compra, en el cual se nos da acceso a una carpeta de Dropbox.

2. Conectar la esterilla al ordenador.

   a. Vía Bluetooth:

      i. Acceder a los ajustes de Bluetooth del ordenador y emparejar su unidad electrónica. Se le solicitará una contraseña, la cual es **1234.** Se debe tener en cuenta que la esterilla no puede estar a más de 5 metros del ordenador para garantizar una conexión adecuada.

      ii. Identificar el puerto de comunicación (COM). Para ello, ir a Panel de control>Hardware y Sonido > Dispositivos e impresoras. Localizar la unidad, hacer clic derecho, seleccionar “Propiedades. En la ventana de propiedades, pinchar la pestaña “Hardware”, situar el cursor del ratón sobre la primera línea disponible y ver el COM asignado a la unidad electrónica. El formato es COM ##, siendo ## un número.

      iii. Ejecutar el software y elegir el modelo de esterilla correspondiente en la pantalla principal del programa.

iv. En la ventana del software, seleccionar el COM identificado previamente, en el menú desplegable de la parte superior izquierda de la pantalla, hacer clic en el botón de “reproducir”.

Finalmente, la esterilla Fitness Mat está conectada correctamente al ordenador. El icono de “reproducir” cambiará a un icono rojo “detener”

Universidad de Alcalá

Escuela Politécnica Superior

ESCUELA POLITECNICA SUPERIOR

Universidad de Alcalá