Trabajo Fin Grado
Ingeniería de Telecomunicación

DESARROLLO DE UNA HERRAMIENTA PARA EL RECONOCIMIENTO FACIAL DE EMOCIONES

Autor: Miguel Ángel Gil Jaime

Tutor: Juan José Murillo Fuentes

Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo Fin Grado
Ingeniería de Telecomunicación

DESARROLLO DE UNA HERRAMIENTA PARA EL RECONOCIMIENTO FACIAL DE EMOCIONES

Autor:
Miguel Ángel Gil Jaime

Tutor:
Juan José Murillo Fuentes
Profesor catedrático

Dpto. de Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2020

Trabajo Fin Grado: DESARROLLO DE UNA HERRAMIENTA PARA EL RECONOCIMIENTO FACIAL DE EMOCIONES

Autor: Miguel Ángel Gil Jaime

Tutor: Juan José Murillo Fuentes

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2020

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

Después de un largo periodo en la escuela hoy pongo fin a una de las mejores etapas de mi vida.

Ha sido un día que me lo he imaginado muchas veces y se ha alargado más de lo que me hubiera gustado, pero como dice el dicho “más vale tarde que nunca”.

Quería agradecer a mis profesores las enseñanzas de todos estos años, algunas las he podido usar en mi vida laboral y otras no, pero hay algo generalizado: nunca ponen las cosas fáciles y eso me ha ayudado a poder resolver cualquier problema que se me plantee en mi vida.

A mi tutor por la ayuda prestada en el momento que más la necesitaba y darme este último empujón.

A todos mis amigos agradecerles esa desconexión que siempre es útil para luego volver con más fuerzas y ese apoyo en los momentos malos.

A mi pareja por el apoyo en todos los momentos difíciles y por darme los ánimos para buscar un hueco y terminar el TFG.

Por último, a mi familia en especial a mi madre por tanta paciencia que ha tenido conmigo y a mi abuela por todos los santos a los que le ha rezado, este TFG va dedicado a ellas dos.

Miguel Ángel Gil Jaime

Sevilla, 2020

Resumen

Las expresiones faciales son los cambios que ocurren en nuestro rostro, como respuesta a los estados emocionales internos.

El análisis de expresión facial tiene múltiples campos de aplicación, desde investigación psicológica, pasando por la cirugía plástica, hasta llegar a la ingeniería y el diseño.

Este proyecto tiene como objetivo el desarrollo de una herramienta que sea capaz de detectar estas respuestas e identificar qué estado emocional refleja el sujeto en estudio.

Apoyándonos en un software libre de reconocimiento facial, llamado *OpenFace*, vamos a realizar una serie de modificaciones, para conseguir que el programa final detecte en un vídeo qué emociones se producen en el sujeto bajo estudio cuando realiza alguna expresión.

Además, se ha creado una interfaz gráfica para facilitar la manera en la que interactuamos con la herramienta.

Los resultados son los esperados: Diferencia las emociones con las que trabaja, aunque a veces se detectan erróneamente. Hay que tener en cuenta que la forma de expresar una emoción es muy distinta dependiendo del sujeto. A pesar de ello, el resultado es aceptable y la interfaz ofrece muchas opciones para el estudio y mejora del software.

Abstract

Facial expressions are the changes that occur in our face, in response to internal emotional states.

Facial expression analysis has multiple fields of application, from psychological research, through plastic surgery, to engineering and design.

This project aims to develop a tool that is capable of detecting these responses and identifying which emotional state reflects the subject under study.

Using free facial recognition software, called OpenFace, we are going to make a series of modifications to ensure that the final program detects in a video which emotions are produced in the subject under study when they make an expression.

In addition, a graphical interface has been created to facilitate the way in which we interact with the tool.

The results are as expected: It differentiates the emotions with which it works, although sometimes they are detected in unwanted frames. Keep in mind that the way of expressing an emotion is very different depending on the subject. Despite this, the result is acceptable and the interface offers many options for studying and improving the software.

Índice

Agradecimientos viii

Resumen ix

Abstract x

Índice xi

Índice de Figuras xiv

1 Introducción 11
1.1 Organización del documento 11
1.2 Objetivos y alcance 11

2 Contexto del proyecto 12
2.1 Estudios psicológicos 12
2.2 Herramientas software 15
2.2.1 Herramientas de reconocimiento de emociones 15
2.2.2 Herramientas de procesado de imagen para caras 17
2.2.3 Subir y mostrar una imagen con *OpenCV* 20
2.2.4 Dibujar formas simples 20
2.2.5 Lenguaje de programación utilizado 23
2.2.6 HW + SW en producción ¿Que se necesita para poner esto a funcionar? 23

3 Idea inicial y referencia 24
3.1 Idea inicial para transformar OpenFace en un software de análisis de emociones 24
3.2 Vídeos de referencia 25
3.2.1 Miedo 25
3.2.2 Sorpresa 26
3.2.3 Ira 26
3.2.4 Indiferencia 27
3.2.5 Repulsión 28

4 Software definitivo 30
4.1 Presentación del programa y funciones 30
4.1.1 Función de borrado de puntos y números en la cara 31
4.1.2 Funciones relacionadas con la modificación de los factores de una expresión 31
4.1.3 Otras funciones 32
4.2 Expresiones.h 32
4.2.1 Declaración de variables y modificaciones 32
4.2.2 Botón *Menu* 33
4.2.3 Botón *Empezar/Pausa/Continuar* 34
4.2.4 Botón *Limpiar Barra* 34
4.2.5 Botón *Borrar punto* 35
4.2.6 Botones 0-9 35

4.2.7 Botón *Borrar* 35
4.2.8 Botón *Limpiar* 36
4.2.9 Botón *Reset* 36
4.2.10 Botón *No pintar* 36
4.2.11 Botón *Aumentar Fuente* y *Disminuir Fuente* 36
4.2.12 Comprobación de expresiones 36
4.2.13 Botón expresiones 36
4.3 LandmarkDetectorUtils.cpp *37*
4.3.1 Definición de variables e inicialización 37
4.3.2 Función *Pintasector* 37
4.3.3 Función *borraunpunto* 38
4.3.4 Función *resetear* 39
4.3.5 Función *nopintarnada* 39
4.3.6 Función *detectar* 39
4.3.7 Función *pintarnumeroscara* 40
4.3.8 Función *distancia* 40
4.3.9 Función *ponern2a0* 41
4.3.10 Funciones que detectan si se ha producido una expresión 41
4.3.11 Función ponertodoacero 42
4.3.12 Funciones que modifican el factor de las expresiones 42
4.3.13 Funciones para aumentar y disminuir fuente 43
4.3.14 Función *Draw* 43
4.3.15 Función *calculaexpresiones* 46

5 Funcionamiento y análisis de los resultados **49**
5.1 Guía de uso *49*
5.2 Análisis de los vídeos de referencia *54*
5.2.1 Análisis de *Miedo* 54
5.2.2 Análisis de *Ira* 56
5.2.3 Análisis de *Indiferencia* 64
5.2.4 Análisis de *Sorpresa* 65
5.2.5 Análisis de *Repulsión* 67
5.3 Base de datos y análisis *69*
5.3.1 Actor 05 70
5.3.2 Actor 07 74
5.3.3 Actor 09 81
5.3.4 Actriz 02 83
5.4 Experimento fijando imagen de referencia *85*
5.4.1 Análisis de *Miedo* 85
5.4.2 Análisis de *Ira* 86
5.4.3 Análisis de *Indiferencia* 87
5.4.4 Análisis de *Sorpresa* 88
5.4.5 Análisis de Repulsión 89
5.5 Análisis de resultados *89*

6 Conclusiones y futuras líneas de trabajo **94**
6.1 Conclusiones *94*
6.2 Futuras líneas de trabajo *94*
6.3 Otros softwares similares disponibles en la actualidad *95*
6.3.1 FACECODING 95
6.3.2 *EmoPLAY* (Orange) 96
6.3.3 EMOFOCUS 97

7 Anexo: Código funciones creadas **99**
7.1 Expresiones.h *99*
7.1.1 Botón *Menu* 100

7.1.2 Botón *Empezar/Pausa/Continuar* 102
7.1.3 Botón *Limpiar Barra* 103
7.1.4 Botón *Borrar punto* 103
7.1.5 Botones 0-9 104
7.1.6 Botón *Borrar* 105
7.1.7 Botón *Limpiar* 105
7.1.8 Botón *Reset* 105
7.1.9 Botón *No pintar* 105
7.1.10 Botón *Aumentar Fuente* y *Disminuir Fuente* 105
7.1.11 Comprobación de expresiones 106
7.2 LandmarkDetectorUtils.cpp *106*
7.2.1 Función *pintasector* 106
7.2.2 Función *borraunpunto* 107
7.2.3 Función *resetear* 107
7.2.4 Función *nopintarnada* 107
7.2.5 Función *detectar* 108
7.2.6 Función *pintarnumeroscara* 108
7.2.7 Función *distancia* 108
7.2.8 Función *ponern2a0* 108
7.2.9 Función *ponertodoacero* 108
7.2.10 Funciones que modifican el factor de las expresiones 109
7.2.11 Funciones que detectan si se ha producido una expresión 109
7.2.12 Funciones para aumentar y disminuir fuente 110
7.2.13 Función *Draw* 110
7.2.14 Función *calculaexpresiones* 112
Referencias **116**

ÍNDICE DE FIGURAS

Figura 1 Charles Darwin incluyó varios retratos realizados por el fotógrafo médico Guillaume Benjamin Duchenne en su libro [1] 12
Figura 2 Regla 7-38-55 13
Figura 3 Expresiones de sorpresa, miedo, disgusto, ira, felicidad y tristeza por Ekman y Friesen [3] 13
Figura 4 Clasificación de emociones por W. G. Parrott 14
Figura 5 Funcionamiento software SHORE 16
Figura 6 Emotion Research Lab apareciendo en Antena 3 17
Figura 7 Entradas que permite *OpenFace*[11] 18
Figura 8 Ejemplo de salida de *OpenFace*[11] 18
Figura 9 Logo de *OpenCV*[12] 19
Figura 10 Código para subir y mostrar imagen en *OpenCV* 20
Figura 11 Código para la creación la matriz image 21
Figura 12 Código para la creación de una línea 21
Figura 13 Código para la creación de los círculos 21
Figura 14 Código para mostrar la matriz ahora modificada 22
Figura 15 Salida de la función *imshow* 22
Figura 16 Momento exacto en el que se produce *miedo* 26
Figura 17 Momento exacto en el que se produce *sorpresa* 26
Figura 18 Momento exacto en el que se produce *ira* - Vídeo referencia 1 27
Figura 19 Momento exacto en el que se produce *ira* - Vídeo referencia 2 27
Figura 20 Momento exacto en el que se produce *indiferencia* 28
Figura 21 Momento exacto en el que se produce *repulsión* 29
Figura 22 Presentación de la interfaz gráfica 31
Figura 23 Desplegable del botón *Menu* 33
Figura 24 Franjas verdes representadas sobre recuadro encima del control de seguimiento 34
Figura 25 Ejemplo de cuando se ha pulsado "Borrar punto" 35
Figura 26 Teclado numérico y *checks* 35
Figura 27 Diagrama de flujo *Pintasector* 38
Figura 28 Diagrama de flujo *borraunpunto* 38
Figura 29 Diagrama de flujo *resetear* 39
Figura 30 Diagrama de flujo *nopintarnada* 39
Figura 31 Diagrama de flujo *detectar* 40
Figura 32 Diagrama de flujo *pintarnumeroscara* 40
Figura 33 Diagrama de flujo *distancia* 41

Figura 34 Diagrama de flujo *ponern2a0* 41
Figura 35 Diagramas de flujo de las funciones que indican si se ha producido una expresión 42
Figura 36 Diagrama de flujo *ponertodoacero* 42
Figura 37 Diagramas de flujo de las funciones que modifican el factor de una expresión 43
Figura 38 Diagramas de flujo de *aumentarfuente* y *disminuirfuente* 43
Figura 39 Diagrama de flujo de función *Draw* 45
Figura 40 Diagrama de flujo *calculaexpresiones* 47
Figura 41 Puntos en la boca 48
Figura 42 Puntos en los ojos, cejas y nariz 48
Figura 43 Guía de uso-Captura 1 49
Figura 44 Guía de uso-Captura 2 50
Figura 45 Guía de uso-Captura 3 50
Figura 46 Guía de uso-Captura 4 51
Figura 47 Guía de uso-Captura 5 51
Figura 48 Guía de uso-Captura 6 52
Figura 49 Guía de uso-Captura 7 52
Figura 50 Guía de uso-Captura 8 53
Figura 51 Guía de uso-Captura 9 53
Figura 52 Imagen de referencia para el vídeo de *miedo* 54
Figura 53 Resultado del vídeo de *miedo* 54
Figura 54 Un momento determinado donde se detecta *miedo* 55
Figura 55 Momento en el que recuadran que se debe detectar *miedo* 55
Figura 56 Resultados numéricos del vídeo de referencia *miedo* 56
Figura 57 Imagen usada como referencia en vídeo de *Ira* 1 57
Figura 58 Resultado de *ira* - vídeo de referencia 1 57
Figura 59 Captura del vídeo donde se detecta *miedo* 58
Figura 60 Mismo ejemplo que figura 45 58
Figura 61 Momento donde se debería detectar *ira* y se detecta *indiferencia* 59
Figura 62 Momento donde se detecta *ira* 60
Figura 63 Resultados numéricos de *ira* - vídeo de referencia 1 60
Figura 64 Imagen usada como referencia para el vídeo de Ira 2 61
Figura 65 Resultado de *ira* - vídeo referencia 2 61
Figura 66 Momento del vídeo de referencia de *ira* 2 donde se produce *sorpresa* 62
Figura 67 Momento del vídeo de referencia de *ira* 2 donde se produce *ira* 62
Figura 68 Momento en el que se detalla que se debe detectar *ira* y se detecta *ira* 63
Figura 69 Resultados numéricos de ira - vídeo de referencia 2 63
Figura 70 Resultado de *indiferencia* vídeo de referencia 64
Figura 71 Resultados numéricos vídeo indiferencia 65
Figura 72 Imagen usada como referencia en el vídeo de *Sorpresa* 65

Figura 73 Resultado vídeo de referencia *sorpresa* 66
Figura 74 Momento en el que se debe detectar *sorpresa*-Vídeo de referencia *sorpresa* 66
Figura 75 Momento en el que se detecta *indiferencia*-Vídeo referencia *sorpresa* 67
Figura 76 Resultados numéricos vídeo de referencia *sorpresa* 67
Figura 77 Imagen usada como referencia en el vídeo de *repulsión* 68
Figura 78 Resultado de *repulsión* 68
Figura 79 Momento en el que se detecta *indiferencia* 69
Figura 80 Resultados numéricos vídeo referencia *repulsión* 69
Figura 81 Imagen de referencia actor 05 70
Figura 82 Resultado vídeo *enfado* actor 05 71
Figura 83 Instante en el que se detecta *ira* 71
Figura 84 Resultados numéricos vídeo *enfado* actor 05 72
Figura 85 Resultado vídeo *sorpresa* actor 05 72
Figura 86 Instante en el que se detecta *sorpresa* 73
Figura 87 Instante en el que se detecta *miedo* 73
Figura 88 Resultados numéricos actor 05 vídeo *sorpresa* 74
Figura 89 Actor 07 imagen de referencia 74
Figura 90 Resultado vídeo *enfado* actor 07 75
Figura 91 Instante en el que se detecta sorpresa 75
Figura 92 Resultados numéricos vídeo *enfado* actor 07 76
Figura 93 Resultado vídeo *sorpresa* actor 07 76
Figura 94 Resultados numéricos vídeo *sorpresa* actor 07 77
Figura 95 Resultado vídeo *miedo* actor 07 78
Figura 96 Resultados numéricos vídeo *miedo* actor 07 78
Figura 97 Captura de *indiferencia* 79
Figura 98 Captura de *ira* 79
Figura 99 Captura de *miedo* 80
Figura 100 Captura de *repulsión* 80
Figura 101 Captura de *sorpresa* 81
Figura 102 Resultado vídeo *enfado* actor 09 81
Figura 103 Resultado vídeo *sorpresa* actor 09 82
Figura 104 Resultado vídeo *miedo* actor 09 82
Figura 105 Resultado de *ira* para la actriz 02 83
Figura 106 Resultado vídeo *sorpresa* para la actriz 02 84
Figura 107 Resultado vídeo *miedo* para la actriz 02 84
Figura 108 Imagen usada como referencia fija en el experimento 85
Figura 109 Análisis de *miedo* con imagen fija 85
Figura 110 Resultado de *miedo* con imagen fija 86
Figura 111 Momento en el que se detecta *ira* vídeo 1 con imagen fija 86

Figura 112 Resultado de *ira* vídeo 1 con imagen fija 87
Figura 113 Resultado de *ira* vídeo 2 con imagen fija 87
Figura 114 Momento en el que se detecta *sorpresa* con imagen fija 88
Figura 115 Resultado *sorpresa* con imagen fija 88
Figura 116 Resultado *repulsión* con imagen fija 89
Figura 117 Tabla resumen de resultados de los vídeos de referencia 89
Figura 118 Resultados numéricos de los vídeos de la base de datos 90
Figura 119 Tabla resumen de resultados de los vídeos de referencia con imagen fija 90
Figura 120 Comparativa vídeos *miedo* 91
Figura 121 Comparativa vídeos *ira* 1 91
Figura 122 Comparativa vídeos *ira* 2 92
Figura 123 Comparativa vídeos *sorpresa* 92
Figura 124 Comparativa vídeos *repulsión* 93
Figura 125 Planes de pago de FACECODING 96
Figura 126 Ejemplo funcionamiento FACECODING 96
Figura 127 Funcionamiento *EMOplay* 97
Figura 128 EMOFOCUS 98

1 INTRODUCCIÓN

No cuentes los días, haz que los días cuenten.

- Muhammad Ali -

1.1 Organización del documento

El documento se organiza en siete capítulos:

En el presente capítulo introducimos la organización del documento, el objetivo y el alcance.

En el Capítulo 2 se detalla el contexto del proyecto diferenciado en dos bloques: un primer apartado sobre estudios psicológicos y un segundo acerca de las herramientas software usadas para el análisis de expresiones, donde se detalla la base de la que partimos, el software libre usado y la librería donde nos apoyamos, explicando algunas funciones básicas interesantes.

En el Capítulo 3 explicaremos qué cambios se van a realizar sobre el software libre para conseguir que realice las funciones de detección de expresiones deseadas. Además, analizamos los vídeos de referencia, dónde nos quedaremos con los detalles de cómo se produce cada emoción.

En el Capítulo 4 nos centramos en la explicación del código de nuestro programa, bloques fundamentales y funciones creadas.

En el Capítulo 5 se analiza la salida del programa sobre los vídeos de referencia y sobre una base de datos pública.

En el Capítulo 6 se exponen las conclusiones obtenidas y se proponen varias tareas a modo de trabajo futuro.

Por último, el Capítulo 7 es un anexo del código en C++ de las funciones creadas.

1.2 Objetivos y alcance

Este proyecto tiene como objetivo la creación de un software, que sea capaz de detectar respuestas involuntarias en el sujeto bajo estudio intentando identificar qué estado emocional refleja.

Se creará a partir del software libre *Openface* y realizaremos un interfaz para el manejo del software interactivamente.

Para ello, el programa recibirá dos archivos: una imagen del modelo a estudiar, con expresión facial neutra, que tomará como referencia y un vídeo del mismo modelo.

Se comparará la imagen de referencia del sujeto analizado con cada uno de los fotogramas del vídeo, para ser capaces de determinar si se ha producido alguna expresión de emoción en algún instante del mismo.

El software será capaz de discriminar entre miedo, ira, sorpresa, indiferencia y repulsión.

Parte del proyecto se desarrolló mientras cursaba las prácticas en la empresa FAICO (Fundación Andaluza de Imagen, Color y Óptica). La verdad de referencia sobre cada expresión fue proporcionada por dicha empresa.

El software fue desarrollado en marzo de 2017.

2 CONTEXTO DEL PROYECTO

2.1 Estudios psicológicos

Las emociones constituyen un aspecto fundamental de la vida humana.

Uno de los primeros en interesarse por este tema e investigar acerca de ello fue Charles Darwin (1809-1882) que en el año 1872 [1] explicaba la importancia de las expresiones y cómo estas afectan de la misma manera a todos los individuos sin importar el género o la raza. Además, clasificó los tipos de expresiones parecidas en distintas categorías como sorpresa y asombro o pena y tristeza.

En la Figura 1 vemos una imagen de un sujeto en el momento de una expresión, la cual parece ser de miedo, tomada del libro de Darwin.

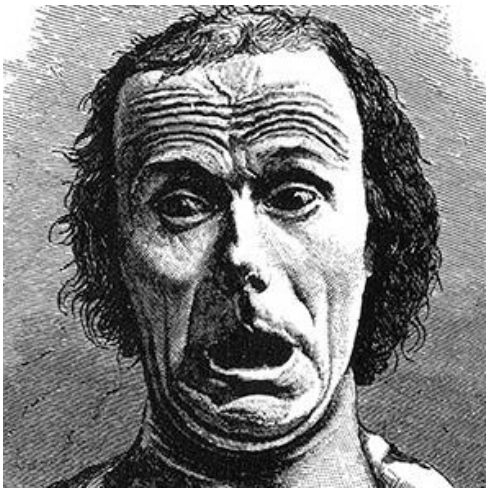

Figura 1 Charles Darwin incluyó varios retratos realizados por el fotógrafo médico Guillaume Benjamin Duchenne en su libro [1]

Siguiendo la línea temporal cabe destacar al psicólogo Albert Mehrabian [2] que en 1968 realizó una serie de estudios y experimentos donde afirmaba que solamente el 7% del potencial de la comunicación recae en las palabras. Un 38% estaría determinado por el tono de la voz y el resto, un 55%, se transfiere a través de las expresiones faciales y corporales. Esto es conocido como la regla 7-38-55.

En la Figura 2 lo podemos ver gráficamente.

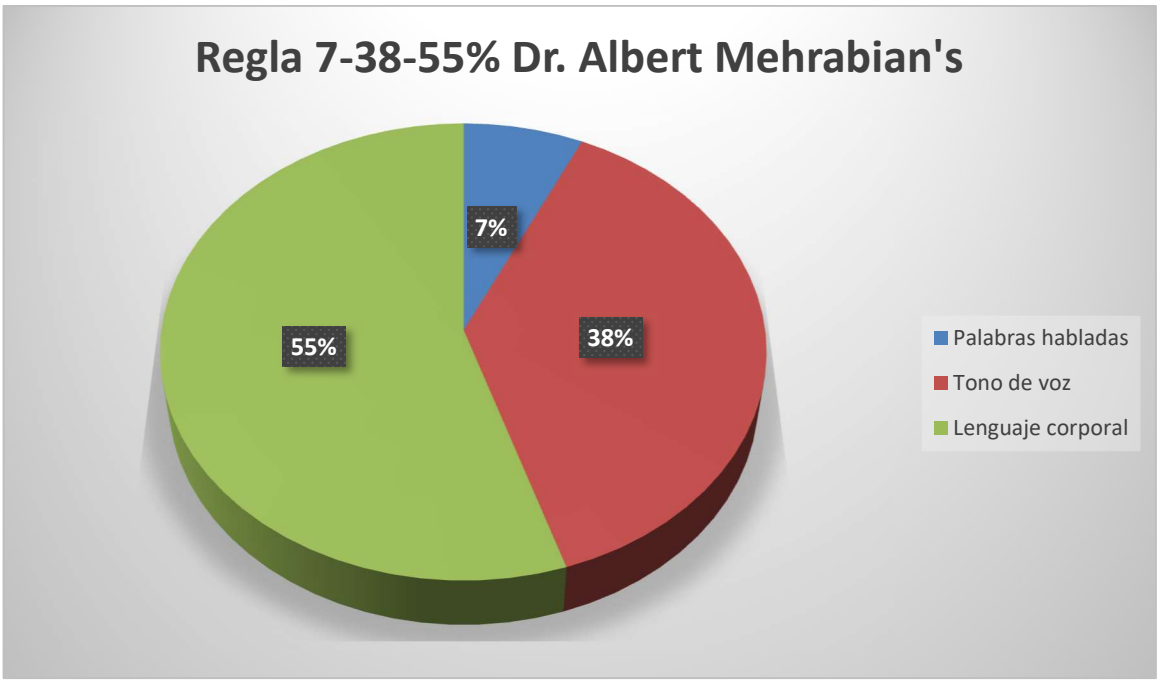

Figura 2 Regla 7-38-55

De nuevo en el siglo pasado, en la década de 1970, tiene lugar uno de los hitos más importantes en el estudio de las expresiones faciales. Ekman y Friesen [3] definieron seis categorías básicas de expresiones: *Anger* (Ira o enfado), *Disgust* (Asco), *Happiness* (Felicidad), *Fear* (Miedo), *Sadness* (Tristeza), and *Surprise* (Sorpresa) que proporcionan la base para todos los estudios que se desarrollan hoy en día en el reconocimiento automático de expresiones faciales. Continuando con la teoría de Darwin, en 1971 estos dos psicólogos realizaron un estudio sobre personas procedentes de distintas culturas. Llegaron a la conclusión de que las expresiones faciales eran prácticamente idénticas en todas ellas, y a pesar de que algunos investigadores, como Russell [4], dudaban de ello, se mostraron fuertes evidencias [5] [6] de que era cierto.

Actualmente es un hecho establecido que las expresiones faciales son universales.

En la Figura 3 se representan las expresiones de sorpresa, miedo, disgusto, ira, felicidad y tristeza, como las clasificaron Ekman y Friesen.

Figura 3 Expresiones de sorpresa, miedo, disgusto, ira, felicidad y tristeza por Ekman y Friesen [3]

En el año 2000 Parrott quiso ir más allá de las seis emociones básicas, definidas anteriormente por Ekman y Friesen. Llegó a identificar hasta 136 estados emocionales [7]. Algunos de estos estados con su clasificación están recogidos en la Figura 4.

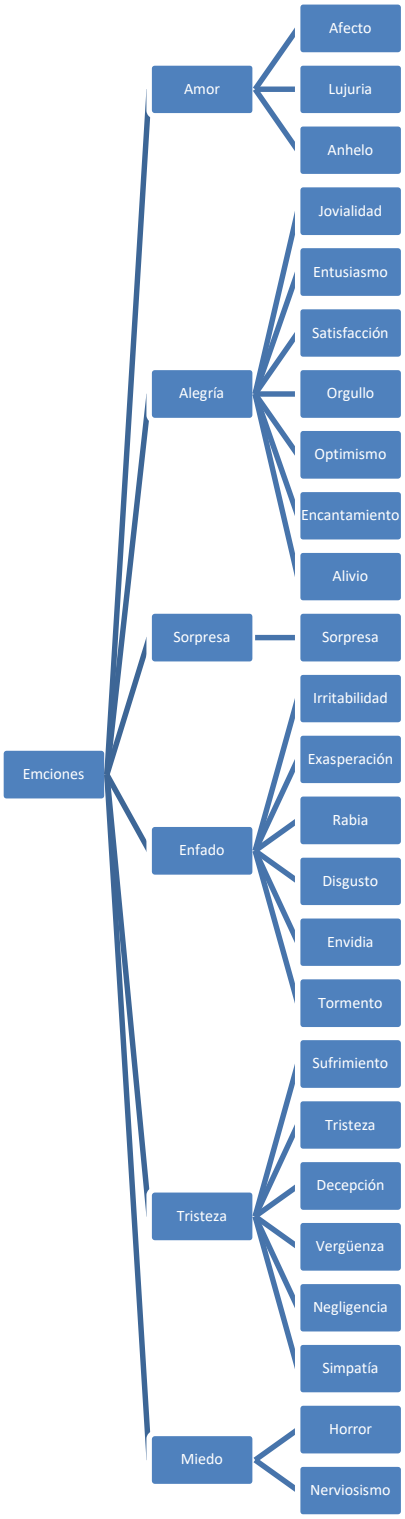

Figura 4 Clasificación de emociones por W. G. Parrott

Actualmente existen numerosos trabajos que intentan reconocer otras emociones además de las definidas por Ekman y Friesen. Como bien recoge Vinay Bettadapura [8] en el interesante estudio del arte que realizó en 2012, nos interesan desde expresiones en la cual la persona está posando (*posed expressions*) hasta micro expresiones (expresiones momentáneas e involuntarias) pasando por las expresiones espontáneas (*spontaneous expressions*), que son expresiones del día al día.

En los últimos años las investigaciones se están centrando en el reconocimiento de estas expresiones espontáneas, aunque no es un proceso sencillo. Requiere rapidez a la hora detectar la expresión y una gran precisión para analizar cuál es realmente la emoción que se está expresando.

Cada vez hay sistemas que reconocen con mayor ratio de eficiencia las emociones, aunque a veces se produce confusión incluso entre algunas de las seis emociones prototipo. Sebe et al. [9] por ejemplo dictaminaron que existe la posibilidad de que se confundan las expresiones de ira con asco o la de miedo con sorpresa.

El software protagonista de este proyecto es capaz de discriminar entre las siguientes emociones:

- Ira
- Indiferencia
- Repulsión
- Sorpresa
- Miedo

Podríamos añadir otras emociones, pero esto podría provocar un peor funcionamiento del software.

En el Capítulo 6.3 hablamos más a fondo de esto.

2.2 Herramientas software

Hasta finales del siglo XX los estudios relacionados con las emociones eran realizados por psicólogos.

Posteriormente, con los importantes avances en el campo de la informática, empezamos a tener cada vez ordenadores más potentes y se empezaron a utilizar estos en el estudio de las emociones.

En los siguientes apartados vamos a hablar de las herramientas disponibles a fecha de 2017 para así compararlos con el software del proyecto.

2.2.1 Herramientas de reconocimiento de emociones

En este apartado vamos a hablar de algunos programas de reconocimiento facial que encontramos en 2017.

2.2.1.1 SHORE

Ha sido desarrollado sobre las gafas del proyecto *Glass* de *Google*.

El software capta los rostros y los enfoca y enmarca como haría un sistema de reconocimiento facial. La clave es una librería de código propietario llamada *SHORE* desarrollada en el Instituto Fraunhofer, aunque existen funciones similares de diversos fabricantes, en código abierto y en varias plataformas.

Lo primero que calcula es la edad de la persona (con un margen de error de más/menos 7 años) y la probabilidad de que sea hombre o mujer (algo en lo que no parece tener dudas, dado que acierta el 94% de las veces).

Al resto de factores también se asignan unos porcentajes, entre ellos los denominados enfado, felicidad, tristeza

y sorpresa. Todo esto lo puede hacer tanto para una o varias personas que estén en el mismo encuadre.

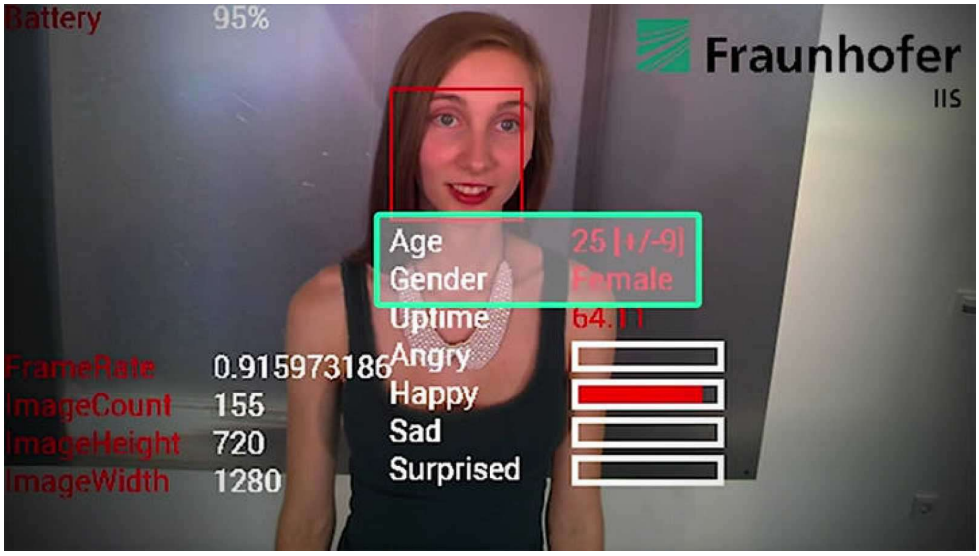

Figura 5 Funcionamiento software SHORE

Para conseguir estos resultados el sistema SHORE se basa en el entrenamiento con más de 10.000 caras evaluadas por personas de carne y hueso: cuanto más se parezcan los parámetros de la cara a los de otras personas en ciertas actitudes (por ejemplo, enfado o sorpresa), más probable es que ambas emociones coincidan.

2.2.1.2 Emotion Research Lab

Es un software que está más relacionado con nuestro proyecto ya que se basa en el análisis de los movimientos musculares con un algoritmo que pasa a identificarlos como felicidad, sorpresa, miedo, disgusto, tristeza y enfado.

En 2016 apareció en Antena 3 donde analizaron las expresiones de cada uno de los políticos de nuestro país.

Este software trabaja mostrando los porcentajes de cada emoción en el instante real y luego viendo los máximos determina un estado emocional.

A continuación, vemos una captura del momento del programa donde se presenta la herramienta:

Figura 6 Emotion Research Lab apareciendo en Antena 3

2.2.1.3 FAICO

Es la empresa que me contrata en 2017 y quiere realizar un software del estilo de los anteriores presentados.

Este software que vamos a crear no se basa como la mayoría de los softwares existentes en un sistema de codificación facial, si no que la verdad de referencia usada para optimizar el programa son los vídeos analizados en el Capítulo 3.2.

2.2.2 Herramientas de procesado de imagen para caras

Haciendo una distinción con las herramientas anteriores, que son herramientas para el reconocimiento de emociones, en este apartado vamos a hablar de la herramienta *OpenFace* cuya funcionalidad es únicamente el reconocimiento facial.

2.2.2.1 OpenFace

Nace en 2015, es una implementación de Python y Torch de reconocimiento facial con redes neuronales profundas basada en el artículo de CVPR 2015 *FaceNet*: *A Unified Embedding for Face Recognition and Clustering por Florian Schroff, Dmitry Kalenichenko y James Philbin[*11].

OpenFace se apoya en una biblioteca libre de visión artificial llamada *OpenCV*.

Nuestro objetivo es modificar *OpeFace* para que pase de ser una herramienta de reconocimiento facial a una herramienta de reconocimiento de emociones.

OpenFace nos permite trabajar con webcam, vídeos e imágenes.

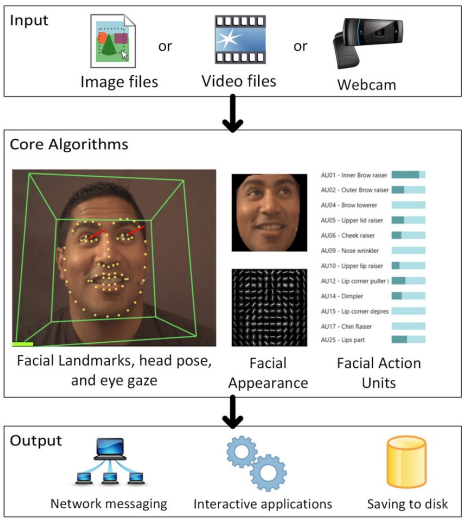

Figura 7 Entradas que permite *OpenFace*[11]

El funcionamiento de *OpenFace* es sencillo, ante una imagen de entrada el programa ejecuta un algoritmo que nos representa sobre la cara detectada unos puntos de referencia, el seguimiento de la vista y la orientación de la cabeza. Lo vemos en la Figura 8.

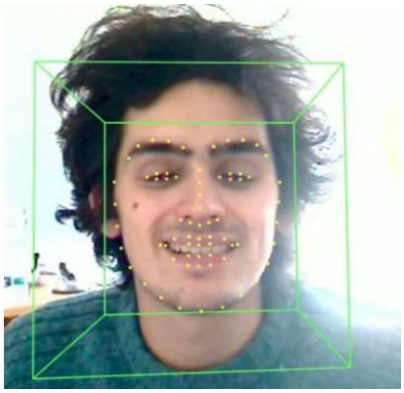

Figura 8 Ejemplo de salida de *OpenFace*[11]

2.2.2.2 OpenCV

Como hemos expresado anteriormente *OpenCV* es la librería en la que se apoya *OpenFace*.

Originalmente fue desarrollada por Intel. *OpenCV* significa *Open Computer Vision*.

Desde que apareció su primera versión alfa en el mes de enero de 1999, se ha utilizado en una gran cantidad de aplicaciones, y hasta 2020 se la sigue mencionando como la biblioteca más popular de visión artificial [12].

Figura 9 Logo de *OpenCV*[12]

Aplicaciones destacadas de *OpenCV*:

- Características 2D y 3D
- Estimación de pose de cámara
- Reconocimiento facial
- Reconocimiento de gestos
- Interacción persona-computadora
- Robótica móvil
- Comprensión de movimientos
- Reconocimiento de objetos
- Segmentación
- Estereoscopía
- *Structure from motion* (*SFM*)
- *Tracking*
- Realidad aumentada

Está totalmente desarrollado en C++, lenguaje en el que trabajaremos.

OpenCV contiene implementaciones de más de 2500 algoritmos. Además, está disponible de forma gratuita para fines comerciales y académicos.

Esta gran biblioteca tiene interfaces para múltiples lenguajes, incluidos Python, Java y C++.

OpenCV trabaja con imágenes utilizándolas como matrices que contienen pixeles de puntos de datos.

Se puede utilizar *OpenCV* para realizar operaciones simples con imágenes como:

- Abrir y guardar imágenes
- Dibujar formas simples en imágenes
- Escribir en imágenes

Estas son operaciones básicas necesarias para crear una base antes de avanzar y poder utilizar todas las funciones avanzadas que OpenCV ofrece.

2.2.3 Subir y mostrar una imagen con *OpenCV*

En la imagen que veremos a continuación podemos observar un ejemplo de subir y mostrar una imagen en *OpenCV* haciendo uso de la función *imread* y *imshow.*

```
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>

#include <iostream>

using namespace cv;

int main(int argc, char** argv) {
    // We'll start by loading an image from the drive
    Mat image = imread("image.jpg", CV_LOAD_IMAGE_COLOR);

    // We check that our image has been correctly loaded
    if(image.empty()) {
        std::cout << "Error: the image has been incorrectly loaded." << std::endl;
        return 0;
    }

    // Then we create a window to display our image
    namedWindow("My first OpenCV window");

    // Finally, we display our image and ask the program to wait for a key to be pressed
    imshow("My first OpenCV window", image);
    waitKey(0);

    return 0;
}
```

Figura 10 Código para subir y mostrar imagen en *OpenCV*

2.2.4 Dibujar formas simples

En las siguientes imágenes podemos observar el código en C++ para crear formas simples.

En primer lugar, crearemos una matriz llamada *image* que se usará para dibujar formas sobre ella. La matriz creada será simplemente una imagen en negro.

Después, crearemos líneas, rectángulos y círculos sobre la matriz en negro.

Finalmente mostraremos la matriz modificada.

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc.hpp> // drawing shapes
#include <iostream>


int main( int argc, char** argv )
{
    // First create a black image.
    cv::Mat image(500,500, CV_8UC3, cv::Scalar(0,0,0));

    // Check if the image is created successfully.
    if( !image.data ){
        std::cout <<  "Could not open or find the image" << std::endl ;
        exit(EXIT_FAILURE);
    }
```

Figura 11 Código para la creación la matriz *image*

```
//#####################(  Draw Line  )##########################
cv::Point p1(100,100), p2(200,100);
cv::Scalar colorLine(0,255,0); // Green
int thicknessLine = 2;

cv::line(image, p1, p2, colorLine, thicknessLine);
```

Figura 12 Código para la creación de una línea

```
//####################(  Draw Circle  )########################
// unfilled circle
cv::Point centerCircle1(250,250);
int radiusCircle = 30;
cv::Scalar colorCircle1(0,0,255);
int thicknessCircle1 = 2;

cv::circle(image, centerCircle1, radiusCircle, colorCircle1, thicknessCircle1);

// filled circle
cv::Point centerCircle2(400,100);
cv::Scalar colorCircle2(0,100,0);

cv::circle(image, centerCircle2, radiusCircle, colorCircle2, CV_FILLED);
```

Figura 13 Código para la creación de los círculos

```
//####################(  Draw Rectangle  )#######################
// unfilled
cv::Point p3(400,400), p4(450,450);
cv::Scalar colorRectangle1(0,0,255);
int thicknessRectangle1 = 3;

cv::rectangle(image, p3, p4, colorRectangle1,thicknessRectangle1);

//   filled
cv::Point p5(100,400), p6(150,450);
cv::Scalar colorRectangle2(255,0,255);

cv::rectangle(image, p5, p6, colorRectangle2, CV_FILLED);
```

```
//##################( Draw Shapes on Image )######################
cv::namedWindow( "Display window", cv::WINDOW_AUTOSIZE );
cv::imshow( "Display window", image );
```

Figura 14 Código para mostrar la matriz ahora modificada

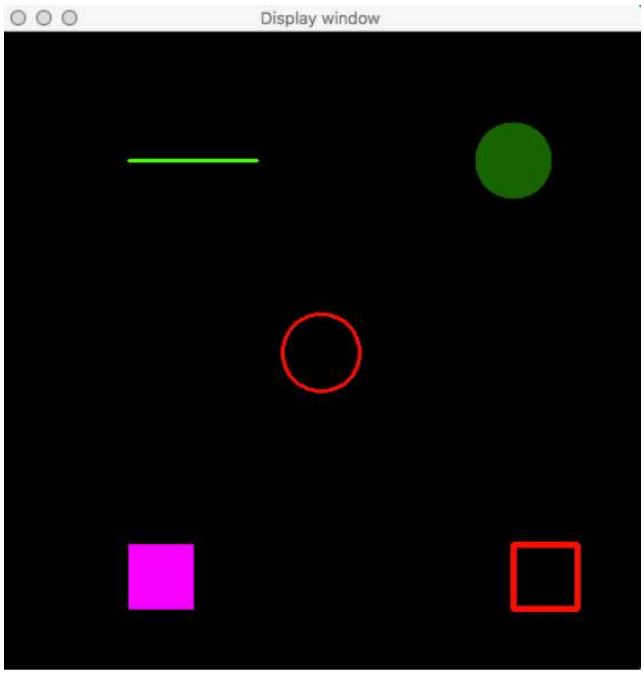

Figura 15 Salida de la función *imshow*

Esta biblioteca nos ofrece muchísimas funciones más, pero esta pequeña introducción ya nos sirve para identificar del software libre el instante en el que se representen los puntos en el rostro del sujeto bajo estudio. Esos puntos que representa *OpenFace* no son más que círculos rellenos de un determinado tamaño en una posición determinada.

2.2.5 Lenguaje de programación utilizado

El lenguaje de programación utilizado en el proyecto es C++.

Ha sido el lenguaje escogido por diferentes motivos:

1. OpenFace permite trabajar en C++ y Phyton
2. La librería OpenCV está programada en C++
3. Conocimientos en el lenguaje del programador.

Para el desarrollo de la interfaz gráfica utilizamos Visual Basic de Microsoft.

El entorno de desarrollo utilizado para todo ha sido Visual Studio 2013.

2.2.6 HW + SW en producción ¿Que se necesita para poner esto a funcionar?

Para comenzar a trabajar en nuestro proyecto es necesario:

1. Equipo con Windows
2. Software Microsoft Visual Studio
3. Instalación en el equipo de la librería OpenCV
4. Descargar el software libre OpenFace

Una vez tenemos todo esto, podemos realizar la prueba de modificar OpenFace y compilarlo de nuevo.

Si la librería esta correctamente instalada en el equipo no tendríamos ningún problema.

Para un usuario que necesite trabajar con el programa basta que tenga el ejecutable del mismo con el contenido que se genera en la carpeta *Release,* la cual contiene las librerías necesarias para la ejecución del programa.

3 IDEA INICIAL Y REFERENCIA

Antes de entrar en detalles de nuestra herramienta, es importante hablar sobre la idea de las modificaciones que se van a realizar sobre el software libre, OpenFace, y analizar los vídeos de referencia pasados por la empresa.

3.1 Idea inicial para transformar OpenFace en un software de análisis de emociones

Comenzamos estudiando el software *OpenFace*, analizando qué podemos usar de él y qué podemos obviar.

Como hemos comentado anteriormente, trabaja con imágenes, vídeos y webcam. El objetivo de nuestro software no es trabajar con vídeos en tiempo real, por lo que sólo usaremos la parte de código donde trabaja con vídeos e imágenes. Sería interesante para futuras líneas de trabajo poder recuperar la opción de la detección por cámara web.

Además, nos podemos quedar únicamente con la parte del código que representa los puntos en el rostro de la persona bajo análisis. No nos interesa la representación del seguimiento de la vista ni el recuadro de orientación con el que trabaja el software original.

Esos puntos, como hemos visto en la función para representar un círculo sobre una imagen, están centrados en una posición determinada. La idea inicial es trabajar con la distancia física que hay entre los distintos puntos que representa el software.

Si la distancia entre dos puntos que están representados en la boca, del sujeto bajo estudio, aumenta respecto a la distancia que tenían originalmente el software reconozca que se ha abierto la boca y si de lo contrario esta distancia disminuye reconozca que por ejemplo se han apretado los labios.

Para ser capaces de comparar la distancia que hay en el instante actual con la distancia que tenían originalmente se le pasará al software una referencia. Ésta será una imagen del sujeto que vamos a estudiar en el vídeo con una expresión neutral.

Para evitar que se produzca un aumento de la distancia entre puntos cuando se amplía la imagen del vídeo, buscaremos dos puntos los cuales se mantengan casi fijos a la hora de producirse una expresión.

Cuando en un determinado momento del vídeo se enfoque de más cerca al actor, crecerá la distancia entre todos los puntos de manera proporcional. Esta distancia será conocida gracias a los dos puntos usados como referencia. Esto lo usaremos para evitar que se detecten emociones erróneamente cuando se amplía el vídeo.

Por ejemplo, al hacer enfocar de cerca la cara de una persona los puntos de la boca tomarán mayor distancia respecto a los de una imagen de referencia de un plano más lejano. El programa podría detectar que se esté produciendo sorpresa. La distancia entre los puntos que se mantienen fijos, los usados como referencia, también aumenta, por lo que usaremos este valor para contrarrestar este efecto y que no se detecte ninguna emoción.

Lo podemos ver con más detalles en el Capítulo 7 del documento donde está definida la distancia entre esos dos puntos como una variable llamada *escala*.

Por último, crearemos una interfaz gráfica para trabajar con los vídeos dinámicamente y que permita al usuario añadirle nuevas características que puedan interesar sobre el software inicial.

3.2 Vídeos de referencia

En este capítulo vamos a analizar los vídeos usados como verdad de referencia. El objetivo es ver como muestran las emociones las personas que aparecen en el vídeo, para usarlo como referencia para nuestro proyecto.

El análisis que se realiza en este capítulo es sobre los vídeos que usamos, aún no interfiere para nada nuestro software, de él empezaremos a hablar en el Capítulo 4.

Estos vídeos se reproducen siguiendo un mismo patrón y están modificados por la empresa. Esta modificación consiste en repetir el momento en el que se produce la expresión de la siguiente manera: Sale un personaje, se produce un determinado momento donde realiza una expresión, al final del vídeo se repite ese momento varias veces, recalcándonos cuál es la expresión que nos interesa. En la última repetición, cuando va a finalizar el vídeo se amplía la imagen en el instante que se produce la respuesta emocional, y se recuadra la expresión que realiza el actor. Por último, se escribe lo que se debe detectar.

En los siguientes subapartados vamos a analizar estos vídeos y vamos a mostrar una captura del momento en el que se produce la expresión y como hemos explicado se recuadra la expresión del actor.

3.2.1 Miedo

El lenguaje corporal del miedo se manifiesta en las micro expresiones faciales. Las cejas ligeramente levantadas, el ceño tenso y la boca entreabierta son señales inequívocas de que el temor está dentro de una persona.

La expresión de miedo puede ser confundida fácilmente con la de sorpresa, pero en nuestro caso, los vídeos empleados como verdad de referencia son muy distintos. Lo veremos en el siguiente apartado donde analizamos el vídeo de sorpresa.

¿Cómo expresa miedo el actor de mi vídeo de referencia? Lo vemos en la Figura 16, dónde podemos observar cómo nos recuadran el momento en el que la actriz abre la boca y levanta las cejas con los ojos muy abiertos.

El software detectará miedo cuando se cumplan las siguientes condiciones:

- Cejas levantadas
- Boca abierta

No usamos como condición que se mantengan los ojos abiertos, para evitar que cuando el sujeto parpadea se filtre la detección de esta emoción.

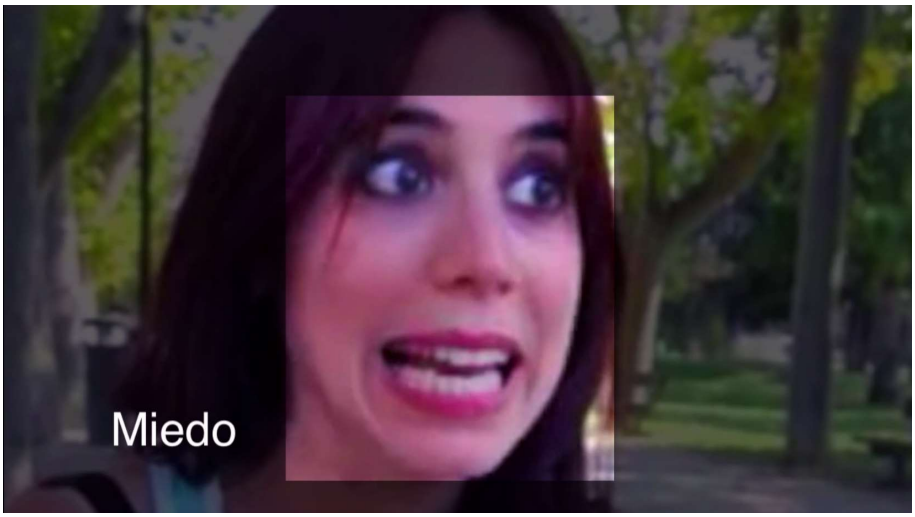

Figura 16 Momento exacto en el que se produce *miedo*

3.2.2 Sorpresa

Como hemos hablado en el apartado anterior, esta expresión puede ser confundida con la de miedo, pero en el vídeo de referencia el actor se muestra muy poco expresivo y lo único que hace es abrir la boca al sorprenderse.

Además, en la Figura 17 podemos ver cómo en este caso, la empresa, nos recuadra únicamente la boca.

Por lo que la diferencia entre la expresión de la emoción de miedo y sorpresa será únicamente la posición de las cejas.

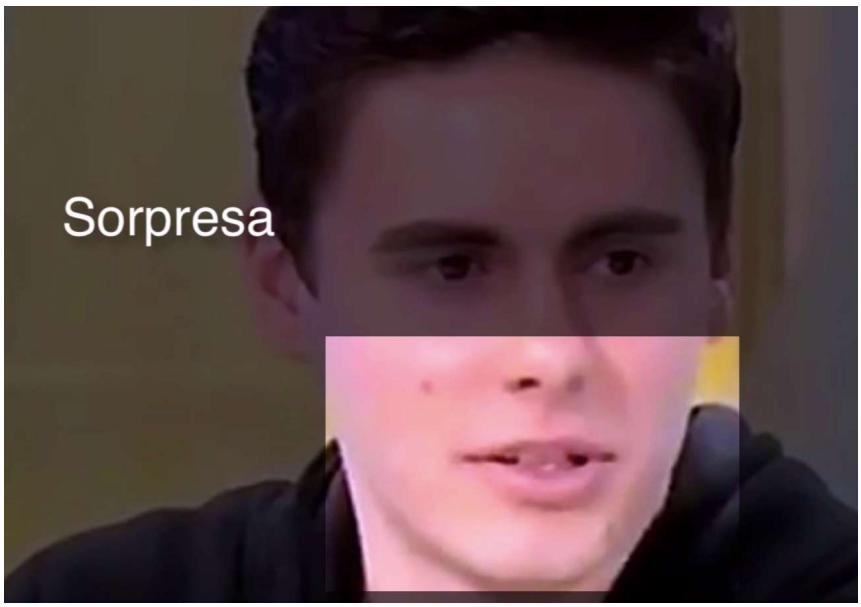

Figura 17 Momento exacto en el que se produce *sorpresa*

3.2.3 Ira

Para la expresión de ira nos pasan dos vídeos para emplearlos como verdad de referencia. Vamos a ver qué tienen en común para poder sacar conclusiones y definir cómo detectará *ira* el software.

En las Figuras 18 y 19 vemos el momento en el que se quiere detectar *ira*.

Figura 18 Momento exacto en el que se produce *ira* - Vídeo referencia 1

Figura 19 Momento exacto en el que se produce *ira* - Vídeo referencia 2

La conclusión que sacamos es que cuando experimentan la emoción de ira aprietan los labios, y aunque no se recuadra, mantienen la mirada por lo que no cierran los ojos.

Al igual que anteriormente vimos cómo la expresión de sorpresa y de miedo tenían en común que el actor abría la boca, vamos a ver, en los siguientes apartados, cómo las expresiones de ira, indiferencia y repulsión también están relacionadas.

3.2.4 Indiferencia

Como podemos observar en la Figura 20 el recuadro del vídeo de referencia se vuelve a centrar sólo en la boca

del actor.

Si nos quedamos sólo con el gesto de la boca, no podemos diferenciar la expresión que se produce cuando se experimenta indiferencia de la que se produce cuando se experimenta ira.

Si vamos más allá y nos fijamos en los ojos, podemos observar cómo Fernando Alonso baja la mirada y caen sus párpados.

Para nuestro software la disimilitud entre ira e indiferencia será el que se mantengan los ojos abiertos o se cierren levemente.

Figura 20 Momento exacto en el que se produce *indiferencia*

3.2.5 Repulsión

Como apreciamos en la Figura 21 el modelo tiene los labios apretados, los ojos más cerrados y las cejas más bajas.

Si se cumplen estas tres condiciones el software detectará repulsión.

Si se cumple que el actor tiene los labios más apretados, los ojos más cerrados, pero no se cumple la última condición, el software detectará indiferencia.

Si de lo contrario sólo se cumple la condición de labios apretados el software detectará ira.

Figura 21 Momento exacto en el que se produce *repulsión*

4 SOFTWARE DEFINITIVO

Este capítulo comienza con un resumen de todas las funciones que ofrece nuestro software.

A continuación, nos detendremos en cada una de éstas para analizarlas con más detalles desde el punto de vista de la programación.

Creamos una interfaz gráfica desde cero, para añadirle funciones que nos parecen interesantes desde el punto de vista del análisis de expresiones.

El archivo solución del proyecto de *visual studio,* llamado *ExpresionesFaico.sln,* está compuesto por muchos ficheros, de los cuales sólo uno ha sido modificado respecto al software libre original, y otro creado nuevo para la interfaz gráfica.

Por lo tanto, nuestro proyecto se basa en:

A. Modificar el fichero perteneciente a *OpenFace, LandmarkDetectorUtils.cpp*
B. Creación de un fichero desde cero, que será la interfaz gráfica y cuerpo de nuestro programa: *Expresiones.h*.

Expresiones.h contendrá toda la programación del interfaz y del procesamiento del vídeo y de la imagen.

LandmarkDetectorUtils.cpp contendrá toda la programación de la representación de los puntos en el vídeo y todo lo relacionado con la detección de expresiones.

4.1 Presentación del programa y funciones

En este capítulo vamos a introducir nuestro software, sin explicar nada a fondo, para hacernos una idea de todo lo que ofrece y de cómo trabaja. En capítulos posteriores entraremos en más detalle de cada función o por ejemplo los puntos y parámetros usados por el software para detectar una expresión.

El software recibirá una imagen, que será usada como referencia de la expresión neutral del sujeto, y un vídeo.

Para la imagen de referencia, en la primera llamada, se calculan las distancias entre los puntos elegidos y se almacenan en variables que se mantendrán fijas y sólo cambiarán cuando se seleccione otra imagen de referencia.

Para el vídeo, se realiza lo mismo pero para cada fotograma. El número de fotogramas lo obtenemos con funciones de *OpenCV*. Las variables donde se almacenan las distancias cambian para cada fotograma.

Para cada fotograma se compararán las distancias entre los puntos de la imagen y el vídeo. En función de la diferencia entre estas distancias, detectará si se ha producido alguna expresió o no.

En el apartado 4.3.15 veremos en más profundidad los puntos entre los que se calcula esta distancia y la difrencia que se tiene que producir, entre la imagen de referencia y el fotograma del vídeo, para que se llegue a detectar.

La imagen de referencia es importante elegirla correctamente. Tenemos que buscar una imagen del actor con una expresión neutral. Elegirla de forma errónea usando, por ejemplo, una imagen donde el actor pose con la boca abierta, hace que el programa detecte de forma errónea que se han apretado los labios cuando el actor simplemente cierre la boca.

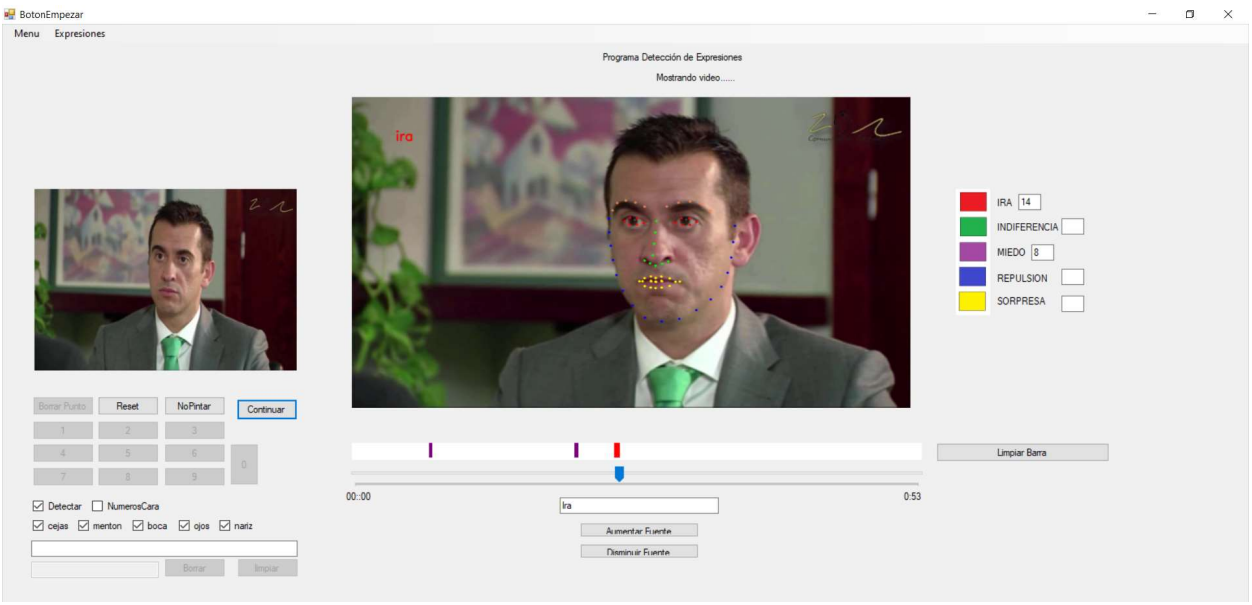

Figura 22 Presentación de la interfaz gráfica

En la Figura 22 podemos ver una captura de nuestra interfaz gráfica en funcionamiento. A la izquierda representa la imagen de referencia, en la parte central se reproduce el vídeo, los otros botones de la interfaz gráfica serán explicados en capítulos posteriores. Cuando el programa detecta que se ha producido una expresión, escribe el texto en rojo con el nombre de la expresión en la esquina superior izquierda del vídeo. Además, representa en un recuadro que está sobre el control del seguimiento del vídeo una franja del color correspondiente a la emoción detectada.

También lleva un control del número de fotogramas detectados, en este caso lleva detectados 14 fotogramas de *ira* y 8 de *miedo*.

4.1.1 Función de borrado de puntos y números en la cara

El programa nos ofrece la opción de borrar puntos interactivamente. Por ejemplo, si estamos analizando el vídeo de un actor y queremos centrarnos en el comportamiento únicamente de los puntos de las cejas y los ojos, podemos anular los sectores de puntos del mentón, nariz y boca.

Además del borrado por sectores, podemos eliminar un solo punto o todos a la vez, si nos apetece que el programa detecte y ver el vídeo original sin ser modificado

Por otro lado, tenemos la función de escribir la numeración que tienen estos puntos.

Si no se ven correctamente tenemos también una función de aumentar o disminuir la fuente del texto.

4.1.2 Funciones relacionadas con la modificación de los factores de una expresión

Para que el software detecte que se ha abierto la boca, la distancia entre los puntos de la boca de la imagen de referencia y del fotograma del vídeo tiene que ser superior por ejemplo a un 30%. Para este caso el factor de boca abierta será 1,3. La obtención de estos factores se explica más a fondo en el Capítulo 4.3.15, donde se habla de la función encargada de esto.

Si en futuros trabajos se quiere mejorar este software y ver cómo funciona si usamos un factor superior o inferior tenemos la opción de modificarlo interactivamente con unas barras de control.

En el momento que se quiere volver a los valores predeterminados del sistema podemos volver pulsando un botón.

4.1.3 Otras funciones

Aparte de las descritas anteriormente destacamos las siguientes:

- Tenemos la función de que el programa deje de detectar en un determinado instante. Esto es interesante cuando hay cambios de plano en el vídeo y participan otros sujetos. Evita el problema de que se puedan mezclar emociones no deseadas.
- Por otro lado, tenemos la opción de limpiar cuando nos interese el recuadro superior al control de seguimiento del vídeo donde se representan las franjas del color de la expresión detectada. Pulsando este botón el recuadro volverá a quedar en blanco.
- Por último, encontramos un botón de pausa, continuar, cancelar y salir.

4.2 Expresiones.h

Es el código de la interfaz gráfica que maneja el programa.

Como vimos en el punto anterior, el programa tiene varias opciones que podemos ir activando o desactivando a medida que se va reproduciendo el vídeo a analizar. Cada vez que pulsamos un botón de la interfaz gráfica se genera un evento, es decir, una llamada a una función que hace los cambios adecuados en las variables del programa para conseguir lo deseado.

Conseguimos realizar estos cambios sin interferir en la reproducción del vídeo porque son procesos que se ejecutan en segundo plano. Estos procesos en segundo plano se ejecutan en un bloque del programa llamado *backgroundWorker1* donde está copiado gran parte del código del software libre, que se encarga de todo el procesamiento de imágenes y vídeo y llama a las funciones de los otros ficheros.

En los siguientes subapartados vamos a desarrollar cada uno de los eventos que se generan al pulsar los botones de la interfaz gráfica y algunas modificaciones del bloque principal del software libre. Todo los que explicamos a continuación es código nuevo programado, de no ser así explicamos que es lo que hemos añadido sobre un código ya existente.

4.2.1 Declaración de variables y modificaciones

Se declaran las siguientes variables al inicio:

```
bool primerclick = true;
bool pause = false;
bool borrarpuntos = true;
bool done = false;
bool cambio = false;
bool bloquear = false;
double numeroborrar;
```

```
int Contador = 0;
bool capturar = false;
int numeroFramesVideo;
int fps;
```

- Variable *primerclick*: De tipo *bool*. La usamos para saber si es la primera vez que le damos a empezar el vídeo para inicializar y limpiar el *trackbar*.
- Variable *pause*: De tipo *bool* e inicializada a *false*. Cuando la ponemos a *true* se queda en un bucle y el programa se queda pausado.
- Variable *borrarpunto*: De tipo *bool*. Se usa también para salir de un bucle. El programa se quedaría pausado mientras le decimos que punto queremos borrar y al pulsar el botón de borrarlo cambia el valor de la variable y el programa sale del bucle.
- Variable *done*: Se pone a *true* cuando finaliza el programa.
- Variable *cambio*: Si hemos pulsado que queremos borrar un número se activaría *cambio*. Entraría en una serie de sentencias *if* para comprobar que se ha introducido un número correcto (0-68) y si es así llamamos a la función de *LandmarkDetector* para borrar ese punto.
- Variable *bloquear*: Se usa junto con *pause* para lo mismo.
- Variable *numeroborrar*: Almacena el número que hemos seleccionado para borrar.
- Variable *contador*: Almacena la posición del control de seguimiento del vídeo.
- Variable *capturar*: Cuando hacemos *click* con el ratón sobre el control de seguimiento se pone a *true* y se consigue ir al instante de vídeo seleccionado.
- Variables *numeroFramesVideo* y *fps*: Almacenan el número de fotogramas del vídeo y el número de fotogramas por segundo. Se usa para conocer la duración del vídeo y saber cuándo se ha finalizado.

Estas variables definidas son útiles para el manejo del interfaz y algunas son utilizadas como banderas para hacer llamadas a las funciones del otro fichero.

4.2.2 Botón *Menu*

Como vemos en la Figura 23 al pulsarlo se despliegan cuatro opciones:

-Seleccionar modelo: Para seleccionar la imagen de referencia.

-Seleccionar video: Para seleccionar el vídeo a analizar.

-Cancelar: Cancela el proceso y permite seleccionar otra imagen o vídeo.

-Salir: Sale del programa y se cierra la interfaz.

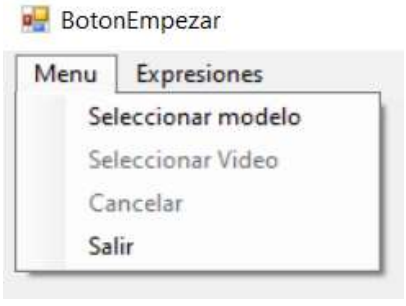

Figura 23 Desplegable del botón *Menu*

4.2.2.1 Botón *Seleccionar modelo*

Se abre una ventana de Windows para la búsqueda de la imagen que servirá como referencia.

Filtra por formatos de imagen y al seleccionarla, la representa en un recuadro a la izquierda de mi interfaz y se llama a la función *Landmarkdetector:ponern2a0*, su uso será explicado en el capítulo 5.

Se habilita el botón *Seleccionar Video*.

4.2.2.2 Botón *Seleccionar Video*

Se abre una ventana de Windows para la búsqueda del vídeo que se comparará con la imagen de referencia.

Filtra por formatos de vídeo y al seleccionarlo lo representa en un recuadro en la parte central del interfaz.

Se hace uso de funciones para determinar el número de fotogramas que lo componen y así conocer la duración del vídeo. Esto es útil para el control de seguimiento y saber cuándo ha finalizado.

Se habilita el botón *Empezar*.

4.2.2.3 Botón *Cancelar*

Se habilita una vez empezado el vídeo y se usa para parar el proceso.

Pone a *true* una variable llamada *primerclick* que se utiliza para diferenciar la primera vez que se pulsa el botón *Empezar*.

4.2.2.4 Botón *Salir*

Al ser pulsado cierra la interfaz gráfica para salir del programa.

4.2.3 Botón *Empezar/Pausa/Continuar*

La primera vez que es pulsado, la variable *primerclick* de tipo booleana toma el valor *true*. Al ser pulsado el botón cambia su texto a *Pausar* y comienza el funcionamiento del programa *backgroundWorker1*.

Cuando se pulsa el botón *Pausar* cambia su texto a *Continuar* y el programa queda en un bucle infinito mientras este botón no cambie el texto.

Cuando se pulsa el botón *Continuar* cambia el texto a *Pausar* y hace que el programa salga del bucle anterior y siga con su funcionamiento normal.

4.2.4 Botón *Limpiar Barra*

En un determinado momento, si nos interesa limpiar el recuadro superior al control de seguimiento, pulsaríamos este botón y como vemos en la Figura 24 las franjas verdes representadas hasta ese momento se borrarían.

Como vimos en las funciones de *OpenCV* lo que se realiza es volver a representar ese recuadro en blanco.

Figura 24 Franjas verdes representadas sobre recuadro encima del control de seguimiento

4.2.5 Botón *Borrar punto*

Deshabilita el propio botón. Habilita todos los botones del teclado numérico y el botón *borrar*.

Como vemos en la Figura 25 salta un cuadro de diálogo de Windows donde nos preguntan si queremos borrar un punto. Si seleccionamos *Sí* las variables *pause* y *bloquear* las pone a *true*, el programa se queda pausado y se habilita el teclado numérico.

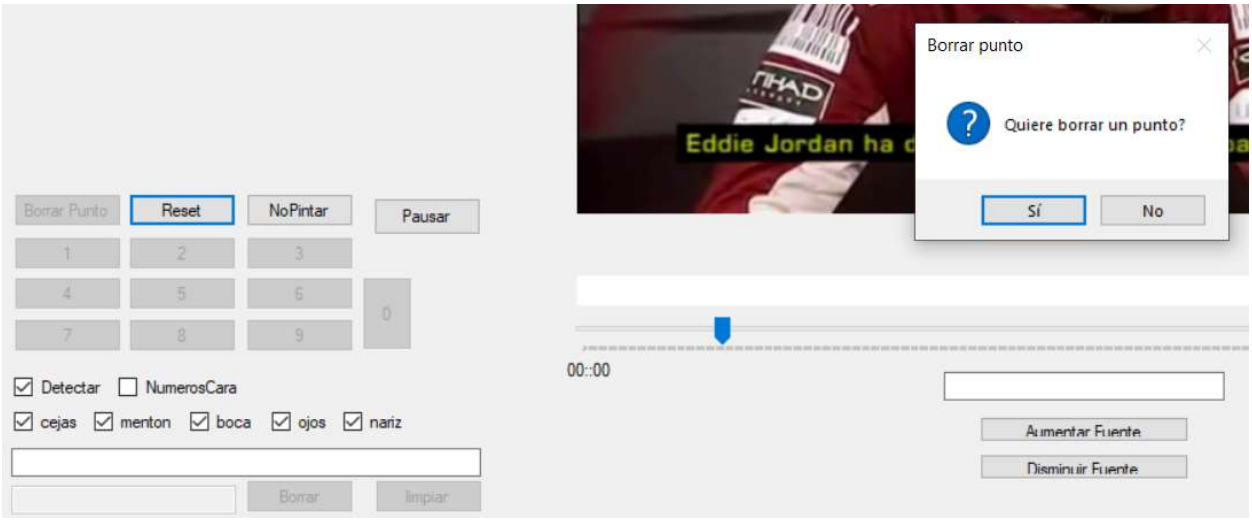

Figura 25 Ejemplo de cuando se ha pulsado "Borrar punto"

4.2.6 Botones 0-9

Al pulsar uno de ellos se va añadiendo al cuadro de texto sombreado en amarillo en la Figura 26.

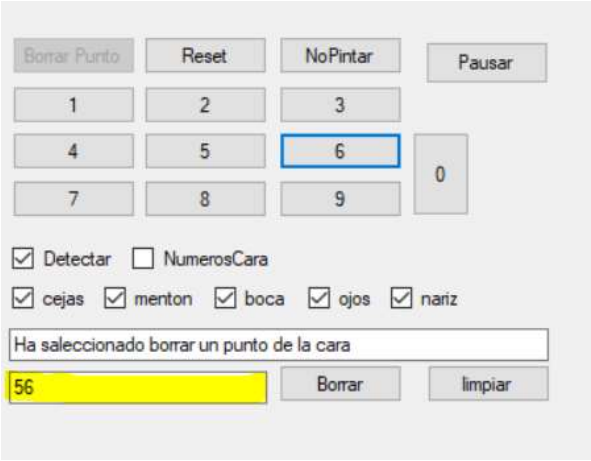

Figura 26 Teclado numérico y *checks*

4.2.7 Botón *Borrar*

Pone a *false* la variable *borrarpuntos*. El número introducido en el cuadro de texto lo almacena en la variable

numeroborrar. Por último, en el cuadro de texto se escribe “Número borrado”.

4.2.8 Botón *Limpiar*

Se borra lo que haya escrito en el cuadro de texto.

4.2.9 Botón *Reset*

Llama a la función *resetear* de *LandmarkDetector* la cual pone todos los cuadros de comprobación del programa como estaban inicialmente.

4.2.10 Botón *No pintar*

Desmarca todos los cuadros de comprobación. El resultado es que no representará nada ya que se desmarcan las opciones de representar el mentón, cejas, ojos, boca y nariz.

4.2.11 Botón *Aumentar Fuente* y *Disminuir Fuente*

Llama a las funciones de *LandmarkDetector* definidas como *aumentarfuente* y *disminuirfuente*.

El efecto causado es que aumenta el tamaño del número escrito sobre el punto del vídeo.

4.2.12 Comprobación de expresiones

Cuando son pulsados se llama a la correspondiente función de *LandmarkDetector*.

Estas funciones están definidas para que cuando sean llamadas, se multiplique el valor de la variable por -1.

Cuando vale 1 representa y al contrario no.

4.2.13 Botón expresiones

Al pulsar sobre esta opción se abre un desplegable con cada una de las expresiones.

Para cada emoción podremos modificar interactivamente el valor de cada uno de los factores.

Por ejemplo, para la emoción de indiferencia podremos cambiar el valor del factor de *Labios apretados* y el valor del factor de *Ojos cerrados.*

De esta manera podemos ver cómo se comporta la herramienta con otros valores.

Si cambiamos demasiados valores, siempre podemos devolverlos a el valor predeterminado.

4.3 LandmarkDetectorUtils.cpp

Este archivo es donde están definidas todas las funciones creadas y que luego serán llamadas al ir interactuando con la interfaz gráfica. En los siguientes subapartados de esta sección se definirán todas las funciones creadas o modificaciones de *LandmarkDetectorUtils*. También mostraremos un diagramada de flujo de cada una de ellas para comprender mejor su funcionamiento.

4.3.1 Definición de variables e inicialización

Las variables definidas son:

```
int n2;

std::string borra = "";
int seleccionado = 0;
int unico;
int numeroscara=-1;
int empezar = 1;
vector<cv::Point2d> vectorpuntos;
std::vector<int>borraunico(68,1);
int menton=1,ojos=1,cejas=1,boca=1,nariz=1;
int contind = 0, contmiedo=0, contira=0, contsorp=0, contrepul=0;
int ignora = 7;
double d1boca, d1cejaojo, dojoizq1, dlabio1, d1nariz;
double factorbocaAbierta=3.7;
double factorLabiosApretados=1.1;
double factorCejasBajadas=1.1;
double factorOjosCerrados=1.1;
double factorOjosAbiertos=1.3;
float fuente = 0.4;

int producemiedo, produceira, producerepulsion, produceindiferencia, producesorpresa;
```

Estas variables definidas son útiles para la detección de expresiones y la representación de los puntos en el vídeo.

A continuación, con la explicación de cada función veremos la utilidad de cada una de ellas.

4.3.2 Función *Pintasector*

Función de tipo *void*, se le llama con un *string* al que llamamos *borra*.

Su utilidad será que conmutará el valor de los enteros definidos para que el sector determinado se represente en mi programa o se elimine.

En la función *Draw* si algún sector tiene el valor -1, no entra en la representación.

En la Figura 27 mostramos el diagrama de flujo de esta función.

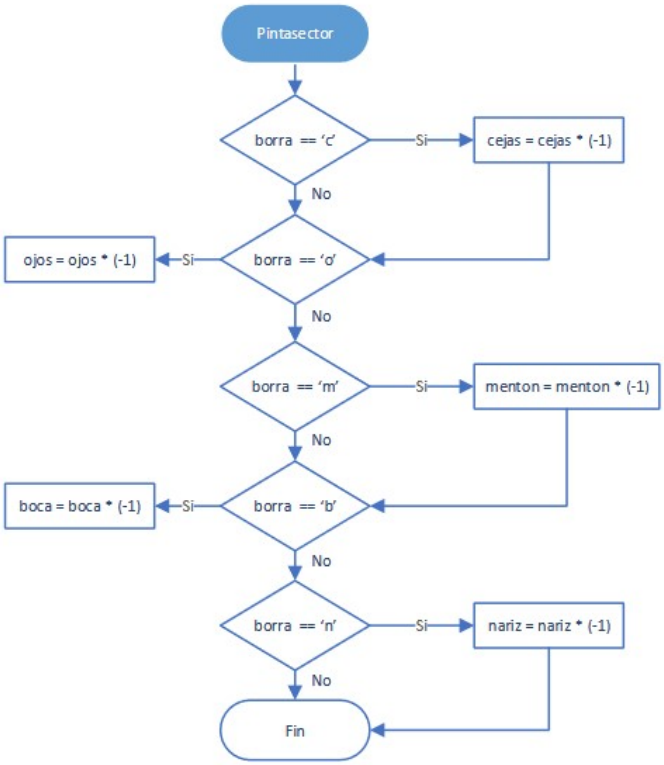

Figura 27 Diagrama de flujo *Pintasector*

4.3.3 Función *borraunpunto*

Función de tipo *void*, se le llama con un entero llamado *unico*.

Conmuta el valor de la posición *unico* del vector *borraunico* para no representar un punto determinado. *Borraunico* es un vector de 68 posiciones todas ellas con valor 1.

Al igual que en la función anterior, la función *Draw no dibujará en este caso el punto que tenga el valor -1.*

En la Figura 28 mostramos el diagrama de flujo de esta función.

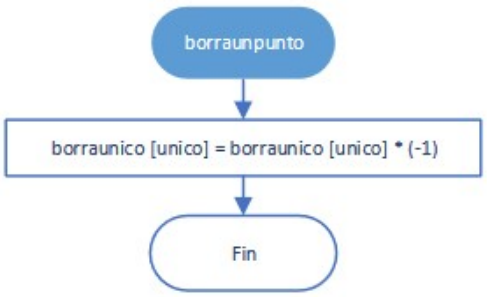

Figura 28 Diagrama de flujo *borraunpunto*

4.3.4 Función *resetear*

Función donde se vuelve a definir el vector *borraunico* para que vuelva a representar todo.

En la Figura 29 mostramos el diagrama de flujo de esta función.

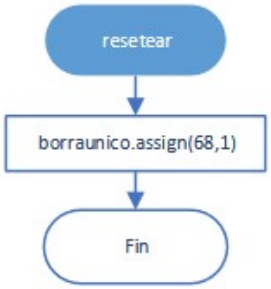

Figura 29 Diagrama de flujo *resetear*

4.3.5 Función *nopintarnada*

Define todos los sectores a cero para que no pinte nada.

En la Figura 30 mostramos el diagrama de flujo de esta función.

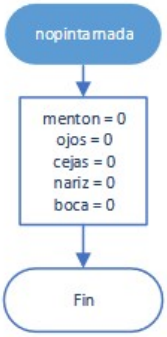

Figura 30 Diagrama de flujo *nopintarnada*

4.3.6 Función *detectar*

Conmuta el valor del entero *empezar*.

Cuando *empezar* vale 1 el programa detecta; cuando conmuta a -1 no detecta.

Esto es de utilidad en vídeos donde hay cambios de planos y salen otras personas a las que no nos interesa analizar. Desde el interfaz seleccionamos la opción correspondiente, se hace una llamada a la función detectar, se conmuta el valor de la variable empezar y el programa no entra en la detección en la función *calculaexpresiones*.

En la Figura 31 mostramos el diagrama de flujo de esta función.

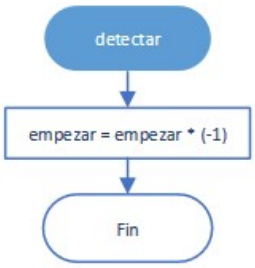

Figura 31 Diagrama de flujo *detectar*

4.3.7 Función *pintarnumeroscara*

Conmuta el valor del entero *numeroscara*.

Cuando *numeroscara* vale 1, la función *Draw* además de representar el punto, escribe sobre él el número correspondiente al orden en el que es representado. Inicialmente vale -1 y no se representa el número.

En la Figura 32 mostramos el diagrama de flujo de esta función.

Figura 32 Diagrama de flujo *pintarnumeroscara*

4.3.8 Función *distancia*

Función que es llamada con tres parámetros: un vector de puntos, y dos enteros.

Su utilidad es calcular la distancia entre los dos puntos especificados por los enteros de ese vector.

En la Figura 33 mostramos el diagrama de flujo de esta función.

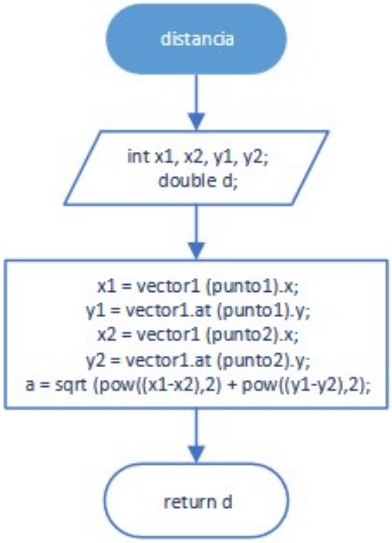

Figura 33 Diagrama de flujo *distancia*

4.3.9 Función *ponern2a0*

Cuando es llamada pone la variable n2 a 0.

Esta variable es utilizada para diferenciar la primera iteración del programa en la cual se hace la copia de todos los puntos de la imagen de referencia. Para las siguientes iteraciones el programa actúa sobre el vídeo.

En la Figura 34 mostramos el diagrama de flujo de esta función.

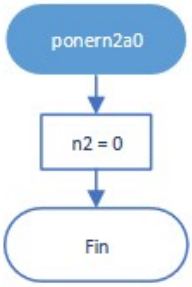

Figura 34 Diagrama de flujo *ponern2a0*

4.3.10 Funciones que detectan si se ha producido una expresión

En este apartado encontramos las siguientes funciones:

- *seProduceMiedo*
- *seProduceIra*
- *seProduceRepulsion*
- *seProduceIndiferencia*

- *seProduceSorpresa*

Se utilizan como *banderas* para comunicarnos con el programa principal de *Expresiones.h*.

En la Figura 35 mostramos los diagramas de flujo de estas funciones.

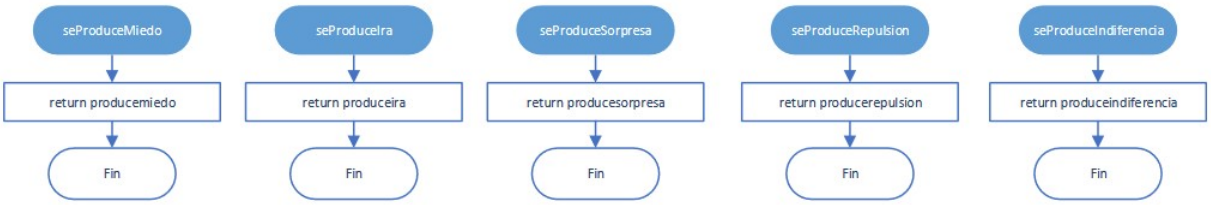

Figura 35 Diagramas de flujo de las funciones que indican si se ha producido una expresión

4.3.11 Función ponertodoacero

Función que evita que se produzcan dos expresiones a la vez ya que una vez que se produce alguna las vuelve a poner todas a cero.

En la Figura 36 mostramos el diagrama de flujo de esta función.

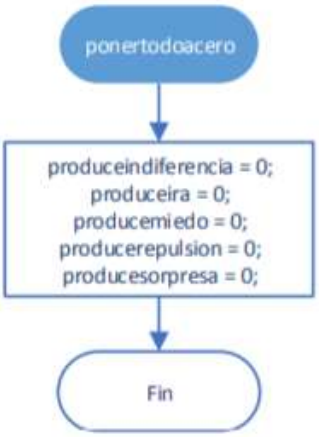

Figura 36 Diagrama de flujo *ponertodoacero*

4.3.12 Funciones que modifican el factor de las expresiones

En este apartado encontramos las siguientes funciones:

- *bocaAbiertaFuncion*
- *cejasBajadasFuncion*
- *labiosApretadosFuncion*
- *ojosAbiertosFuncion*

- *ojosCerradosFuncion*

Es llamada con una variable de tipo *double* definida como: *factor*.

Poniendo el ejemplo de la primera modificaría *factorbocaAbierta* y tomaría el valor de *factor*.

En la Figura 37 mostramos los diagramas de flujo de estas funciones.

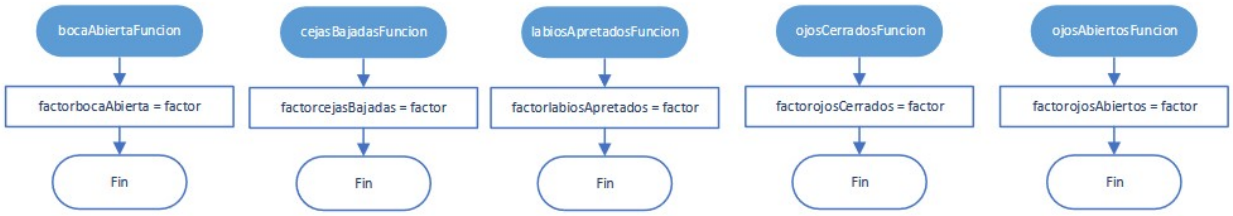

Figura 37 Diagramas de flujo de las funciones que modifican el factor de una expresión

4.3.13 Funciones para aumentar y disminuir fuente

La utilidad de estas funciones es aumentar o disminuir el tamaño del número representado cuando son llamadas.

En la Figura 38 mostramos los diagramas de flujo de estas funciones.

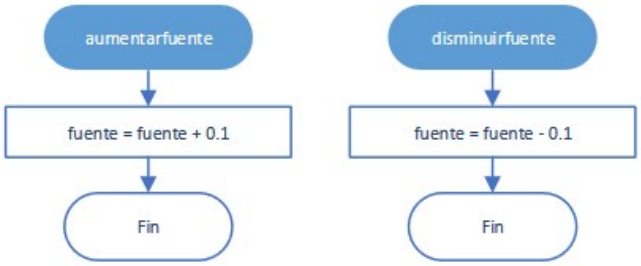

Figura 38 Diagramas de flujo de *aumentarfuente* y *disminuirfuente*

4.3.14 Función *Draw*

Función que pertenece al software libre encargada de poner en la imagen y en el vídeo los puntos sobre la cara del actor. En cada fotograma se llama a la función *Draw*.

Aunque pertenece al software libre, ha sido modificada para añadir nuevas características:

- Añadimos que dentro de ella se haga la llamada a la función *distancia*.
- En la primera iteración, cuando *n2* vale 0, se llama la función distancia con el vector que almacena los puntos de la imagen de referencia. Este vector no cambia.
- En el software original los puntos se iban representando en el fotograma recorriendo un bucle *for* desde

0 hasta *n* siendo *n* el número de puntos. Ahora se diferencian los puntos de la cara por sectores: mentón, nariz, boca, ojos y cejas.

- Para que cumpla con los requisitos de nuestro programa se le han añadido una serie de sentencias de tipo *if*. Si no se cumple alguna de estas sentencias, no entra en el bloque del *if* y no representa el punto o el sector de puntos.
    1. La primera comprobación es si la posición *i* del vector *borraunico* vale 1 entra en el bloque, si de lo contrario vale -1 no entra. Esto sirve para no representar un único punto. A continuación, se realizan las siguientes comprobaciones:
    2. ¿El valor de *i* está comprendido entre 0-16 y además la variable *mentón* vale 1? Entra en el bloque y dibuja el punto que corresponde al sector del mentón en color azul.
    3. ¿La variable *numeroscara* vale 1? Además, escribe sobre el punto el valor de *i.*

De esta forma conseguimos representar cada sector de un color, añadir los números sobre el punto y que se pueda no representar un sector determinado.

Los puntos se representan en el siguiente orden:

- Los puntos del mentón van desde el 0 al 16.
- Los puntos de las cejas van desde el 17 al 26.
- Los puntos de la nariz van desde el 27 al 35.
- Los puntos de los ojos van desde el 36 al 47.
- Los puntos de la boca van desde el 48 al 67.

Al finalizar, se realiza llamada a la función *calculaexpresiones* y se libera memoria del vector que almacena los puntos del fotograma actual.

En la Figura 39 mostramos el diagrama de flujo de esta función.

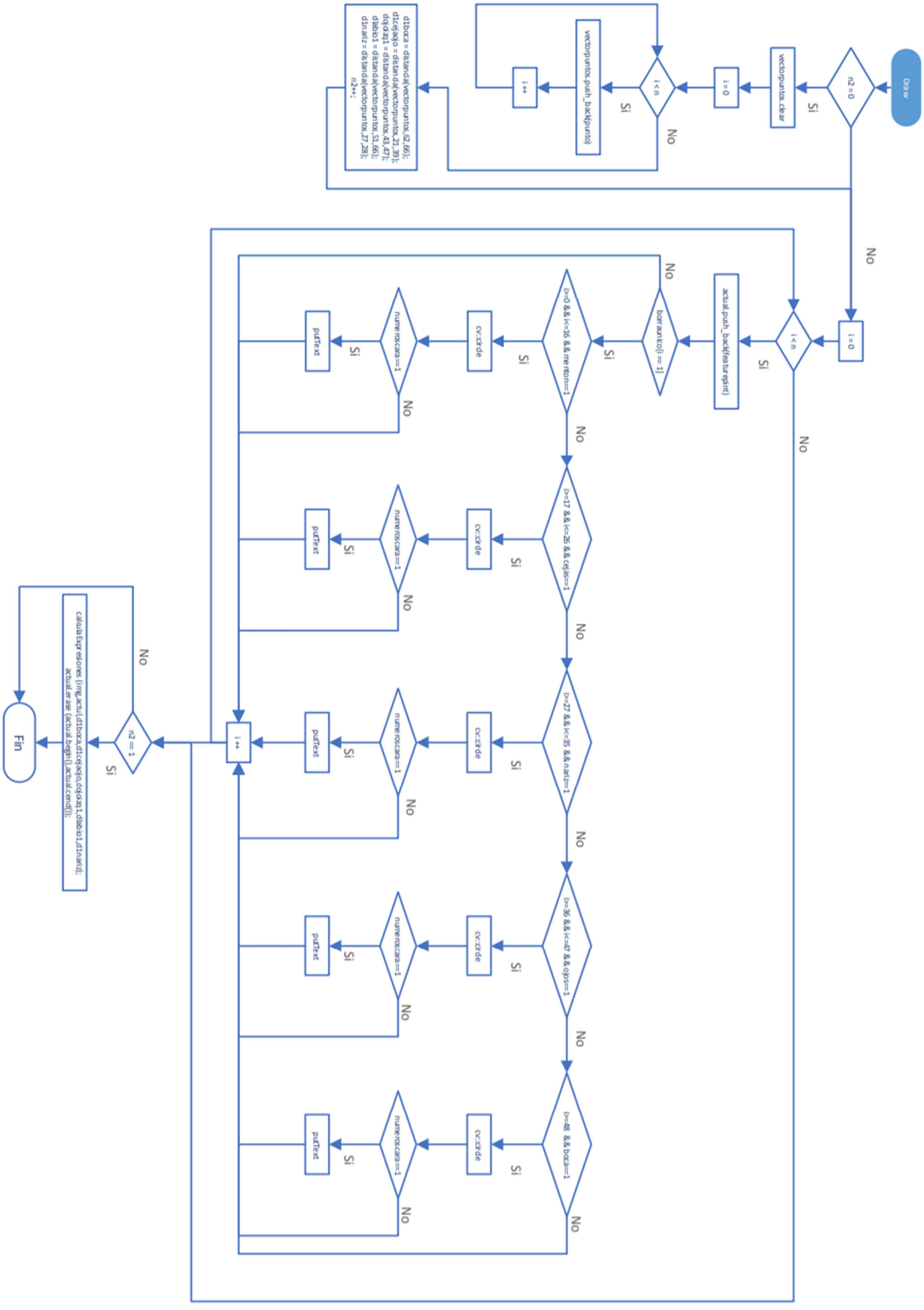

Figura 39 Diagrama de flujo de función *Draw*

4.3.15 Función *calculaexpresiones*

En esta función se calculan las distancias del vector que almacena los puntos del fotograma actual del vídeo.

A diferencia del vector que almacenaba los de la imagen de referencia, éste sí cambia con cada iteración.

Una vez que tenemos las distancias entre los puntos de la imagen de referencia y la distancia entre los del fotograma de vídeo actual podemos compararlas.

Si hay un cambio significativo entre las distancias de los puntos de la imagen de referencia y la actual, el programa detectará que se ha producido una expresión. A continuación, explicamos lo que tiene que suceder para que se llegue a detectar.

Las variables que se declararon fueron las siguientes:

```
double factorbocaAbierta=3.7;
double factorLabiosApretados=1.1;
double factorCejasBajadas=1.1;
double factorOjosCerrados=1.1;
double factorOjosAbiertos=1.3;
```

Estos valores se obtuvieron realizando pruebas con cada vídeo. Se iban modificando y se probaban. Finalmente, nos quedamos con los valores que funcionaban mejor para el conjunto de vídeos.

En el siguiente párrafo, vamos a poner un ejemplo de comprobación para que se llegue a detectar *miedo* o *sorpresa:*

Si la distancia entre los puntos de la boca del vector actual es superior a la distancia de los puntos de la boca del vector de referencia multiplicado por el factor *bocaAbierta* y por el factor *escala,* y además la variable *empezar* vale 1, quiere decir que se ha abierto la boca.

```
//boca abierta con escala
if ((d2boca > (factorbocaAbierta)*escala*d1boca) && (empezar == 1))
```

En este caso, además se comprobaría si el sujeto ha levantado las cejas.

Si se cumplen las dos condiciones aumentaría el contador de *miedo*.

Si de lo contrario, sólo se cumple la primera aumentaría el contador de *sorpresa*.

Se tiene que detectar la misma expresión durante 7 fotogramas seguidos para que se detecte la emoción.

Se define un contador de *miedo* que tiene que superar a la variable *ignora*, cuyo valor es 7. En el momento en el que el contador supera el valor 7, la variable *producemiedo* toma el valor 1 y se escribe el texto *Miedo* en el fotograma actual del vídeo.

En la Figura 40 mostramos el diagrama de flujo de esta función.

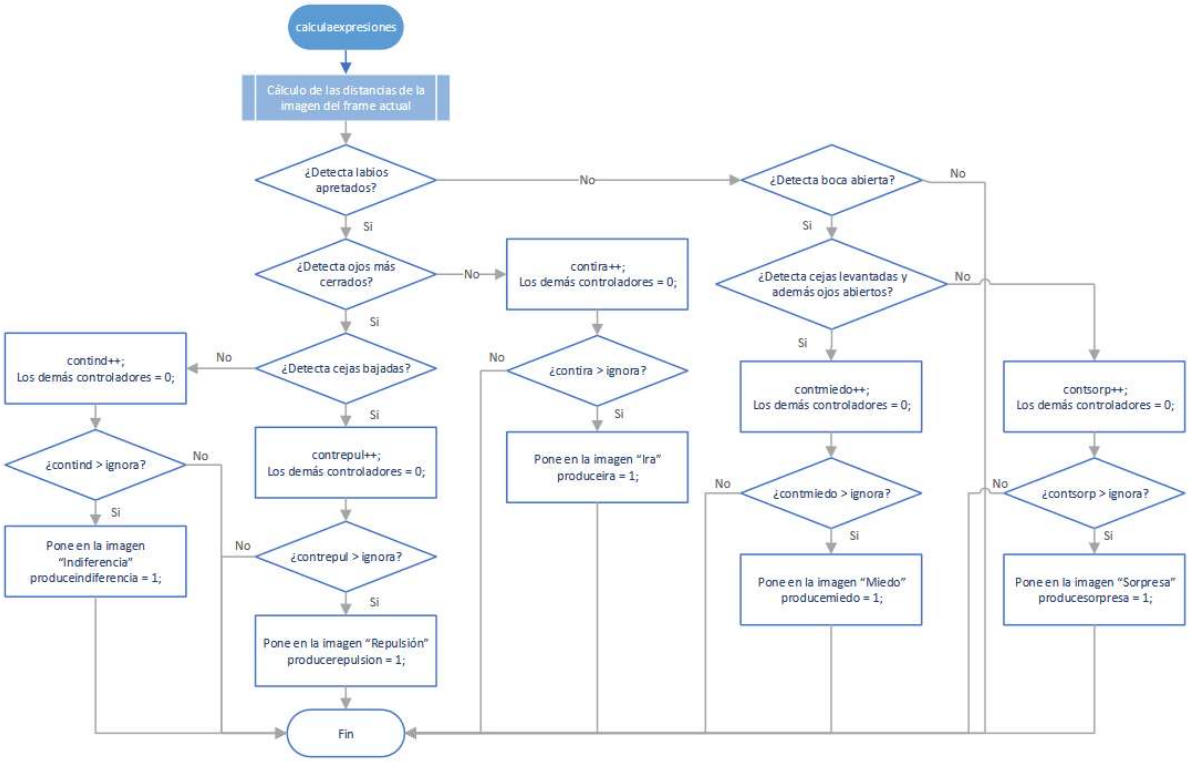

Figura 40 Diagrama de flujo *calculaexpresiones*

Vamos a explicar más a fondo cuales son los puntos usados para detectar cuándo se abre la boca, cuándo se levantan las cejas etc.

- La distancia entre los puntos 62 y 66 nos ayudará a determinar cuándo se ha abierto la boca.
- La distancia entre los puntos 51 y 66 nos ayudará a determinar cuándo se han apretado los labios.
- La distancia entre los puntos 21 y 39 nos ayudará a determinar cuándo se han levantado o bajado las cejas.
- La distancia entre los puntos 43 y 47 nos ayudará a determinar cuándo se han cerrado los ojos.
- La distancia entre los puntos 27 y 28, al ser dos puntos que no varía la distancia entre ellos cuando se produce alguna expresión, la usaremos como un factor de escala. Como hemos explicado en otros puntos anteriores nos ayuda a determinar cuándo se ha producido un enfoque en el vídeo o puede que sea el mismo actor el que se haya acercado a la cámara.

En las Figuras 41 y 42 mostramos la localización de cada punto, a partir de los cuales se detectarán las expresiones.

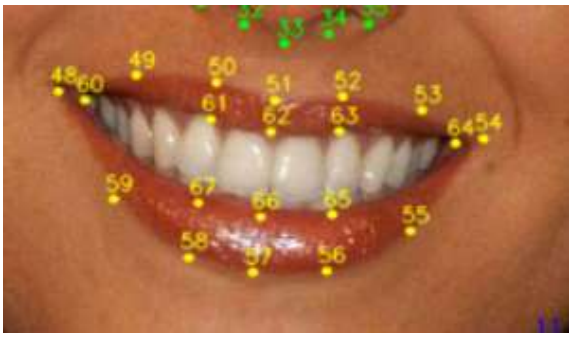

Figura 41 Puntos en la boca

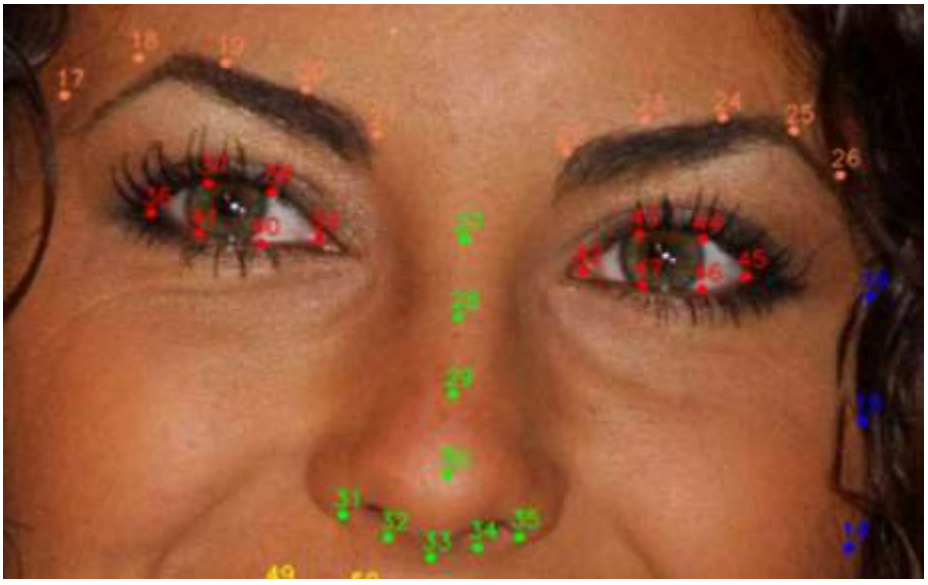

Figura 42 Puntos en los ojos, cejas y nariz

5 FUNCIONAMIENTO Y ANÁLISIS DE LOS RESULTADOS

En el siguiente apartado vamos a comentar como usar nuestro software, los pasos a seguir.

Luego entraremos en el análisis de los resultados de los vídeos de referencia.

Además, se han descargado vídeos de una base de datos de actores, que realizan en un determinado momento una expresión. Se analizarán algunos de ellos con detenimiento.

5.1 Guía de uso

A continuación, explicaremos una breve guía de como comenzar a utilizar nuestro software.

Empezamos abriendo nuestro ejecutable *ExpresioonesFaico.exe.*

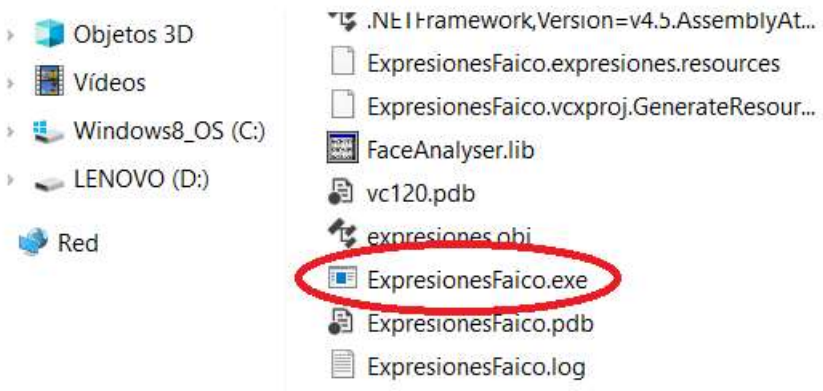

Figura 43 Guía de uso-Captura 1

Se nos abre una ventana de nuestro programa como podemos observar en la Figura siguiente:

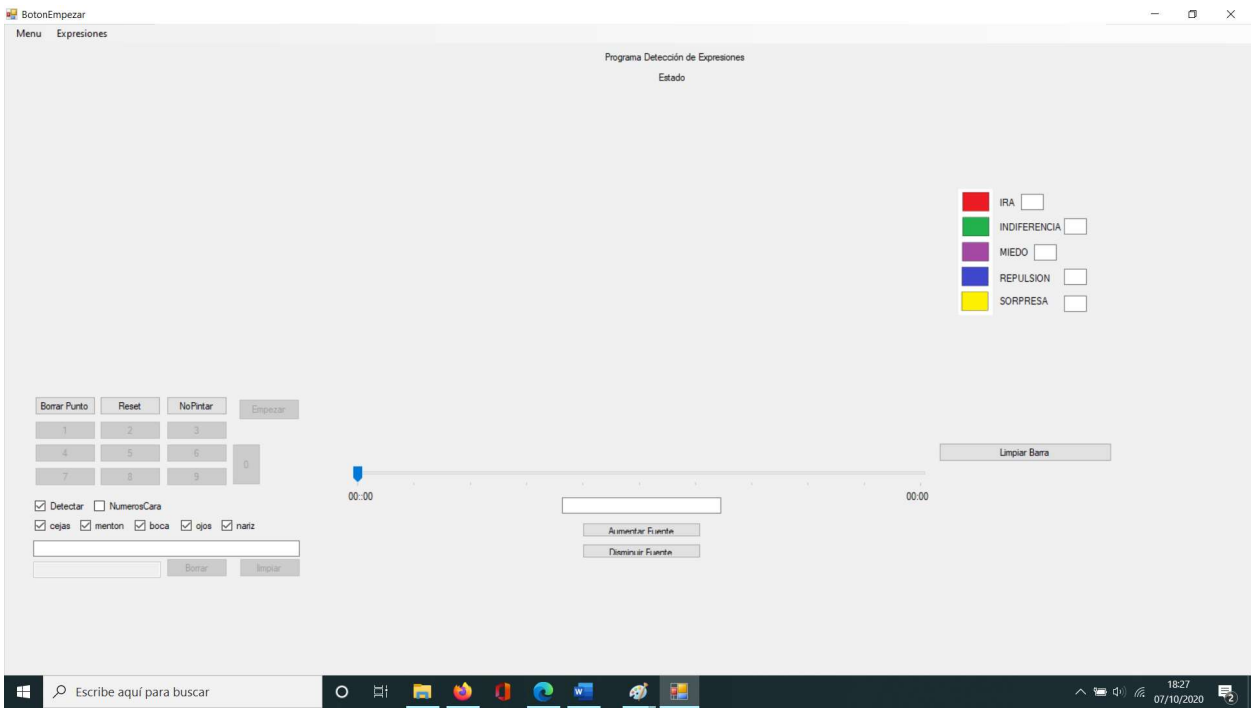

Figura 44 Guía de uso-Captura 2

Pulsamos sobre el botón *Menu* y en el desplegable volvemos a pulsar sobre la opción *Seleccionar modelo*.

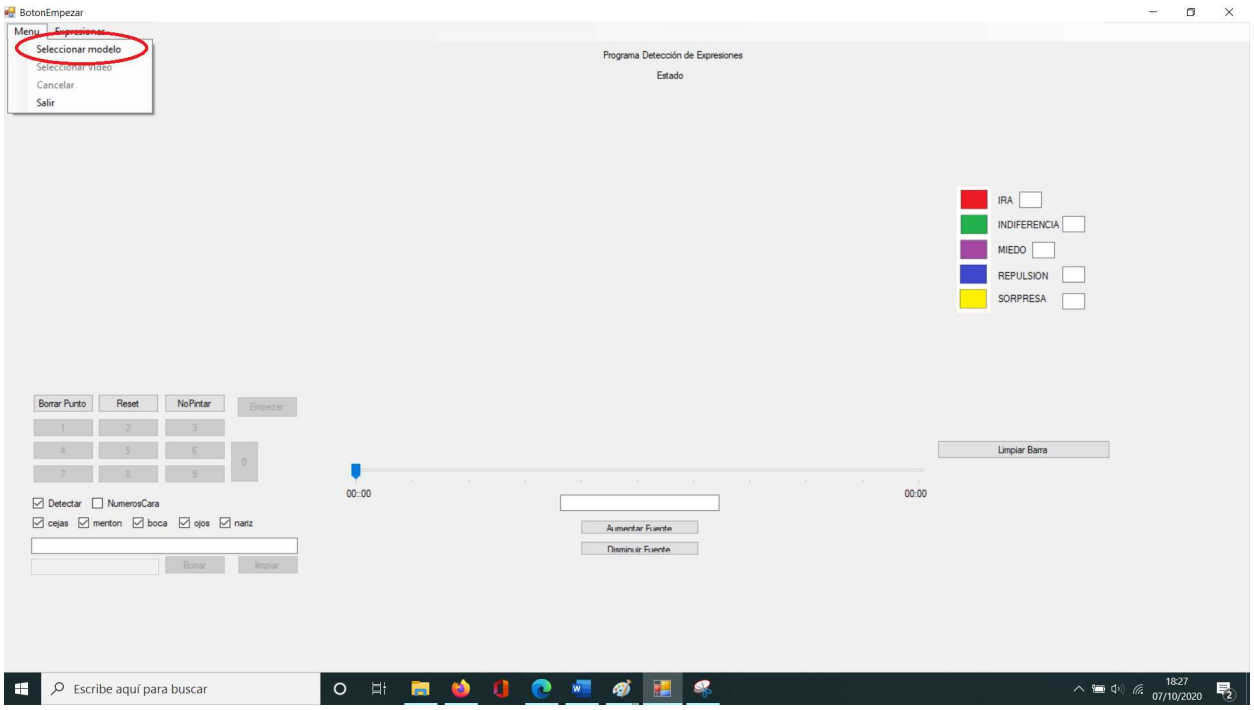

Figura 45 Guía de uso-Captura 3

Se nos abre una ventana de Windows y procedemos a buscar la imagen que queremos usar como referencia.

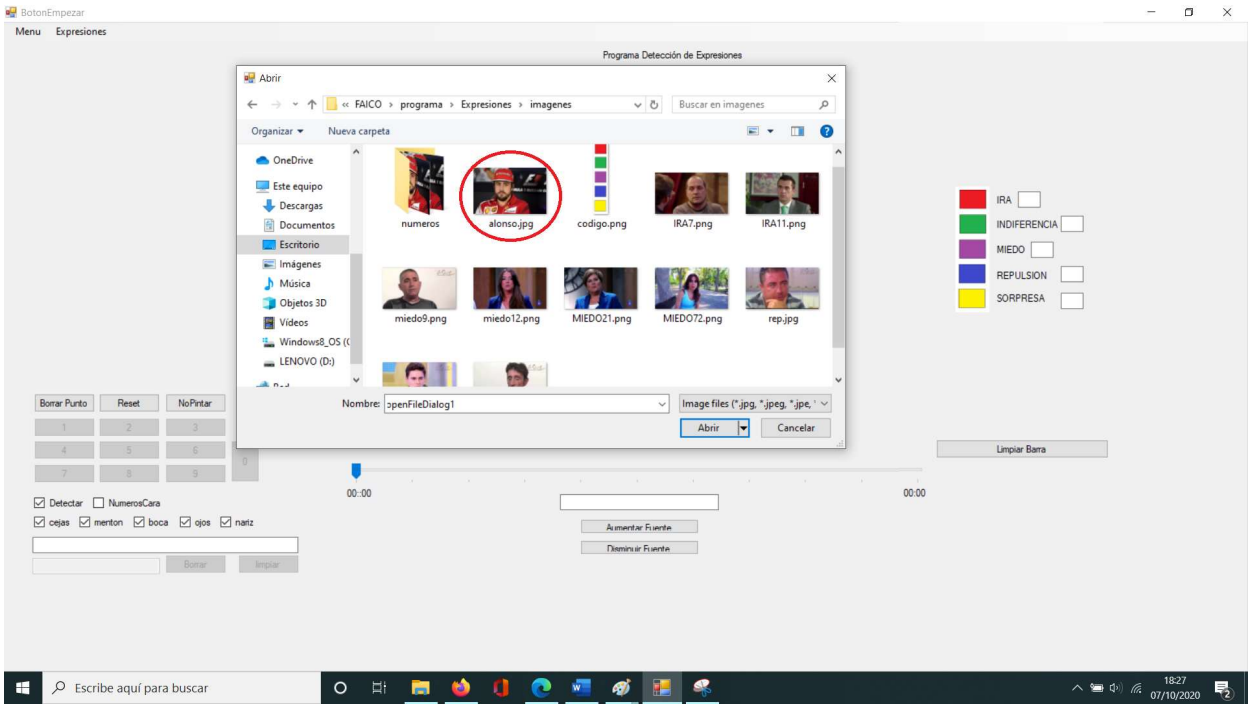

Figura 46 Guía de uso-Captura 4

La imagen se coloca a la izquierda del interfaz.

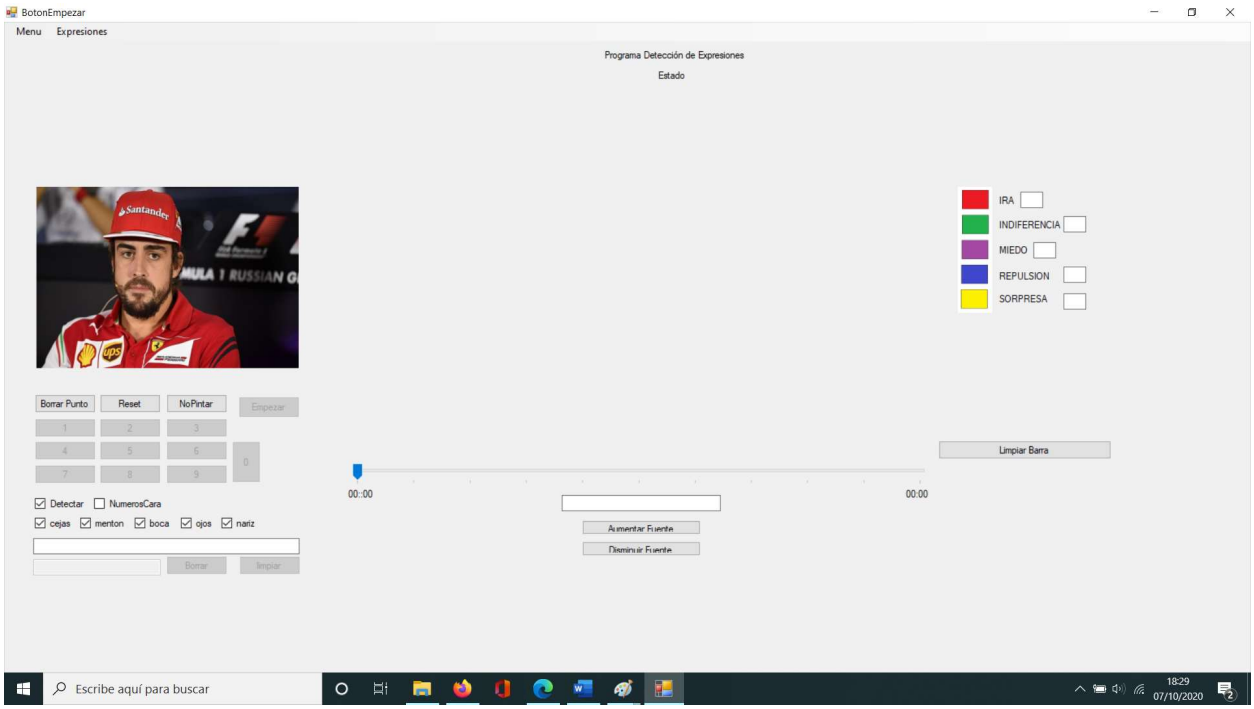

Figura 47 Guía de uso-Captura 5

Volvemos a pulsar el botón *Menu* para a continuación, pulsar sobre la opción *Seleccionar Video.*

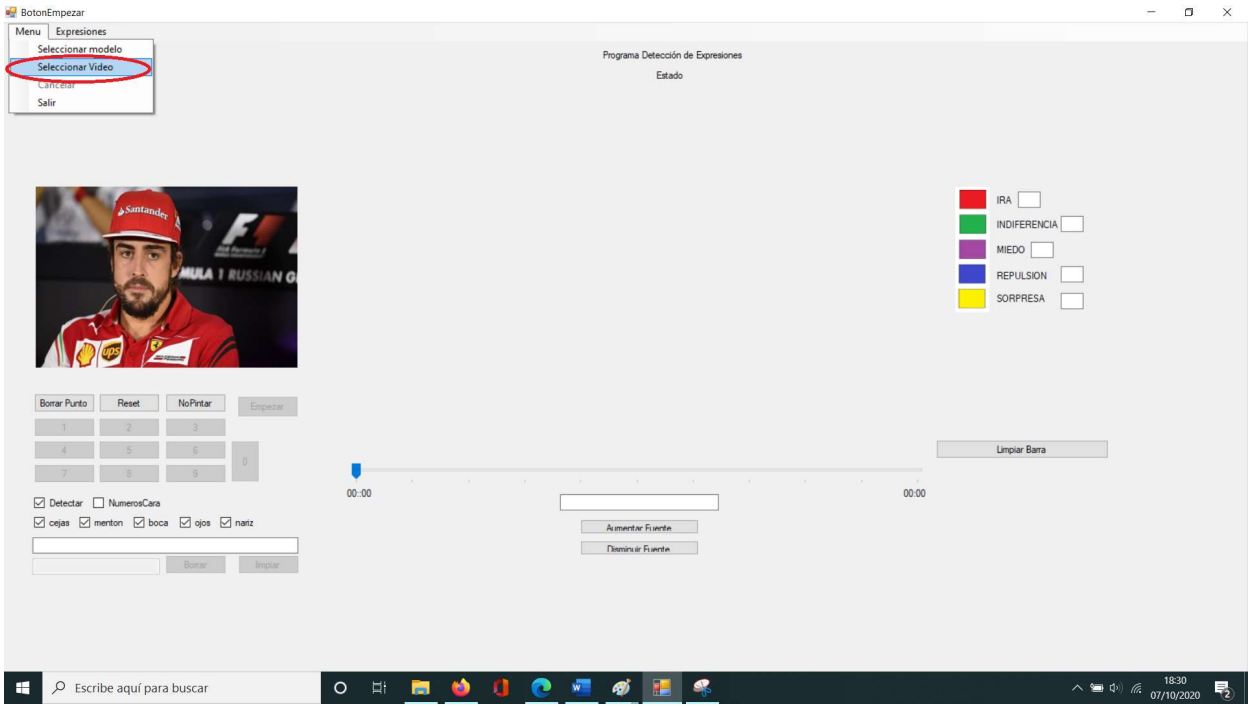

Figura 48 Guía de uso-Captura 6

De nuevo se abre una ventana de Windows para buscar el vídeo que deseamos usar.

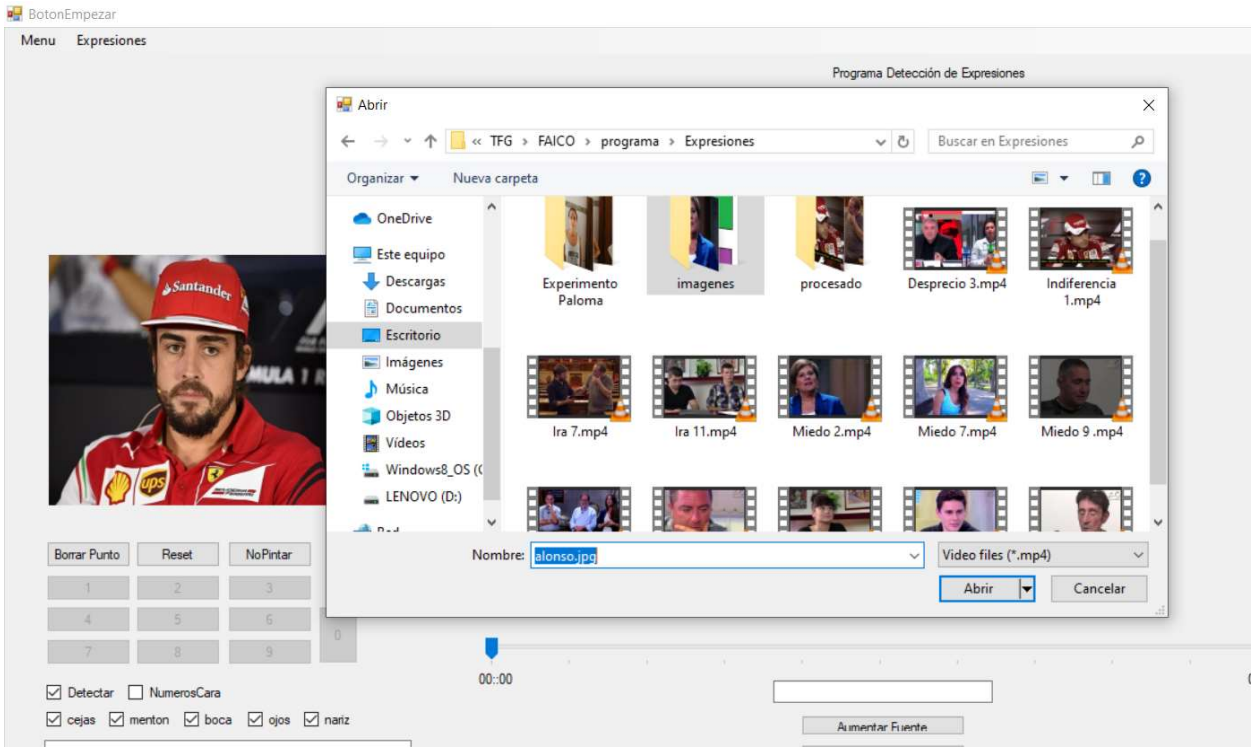

Figura 49 Guía de uso-Captura 7

Ahora el vídeo seleccionado se coloca en la parte central del interfaz.

Pulsamos el botón empezar y comienza nuestro software.

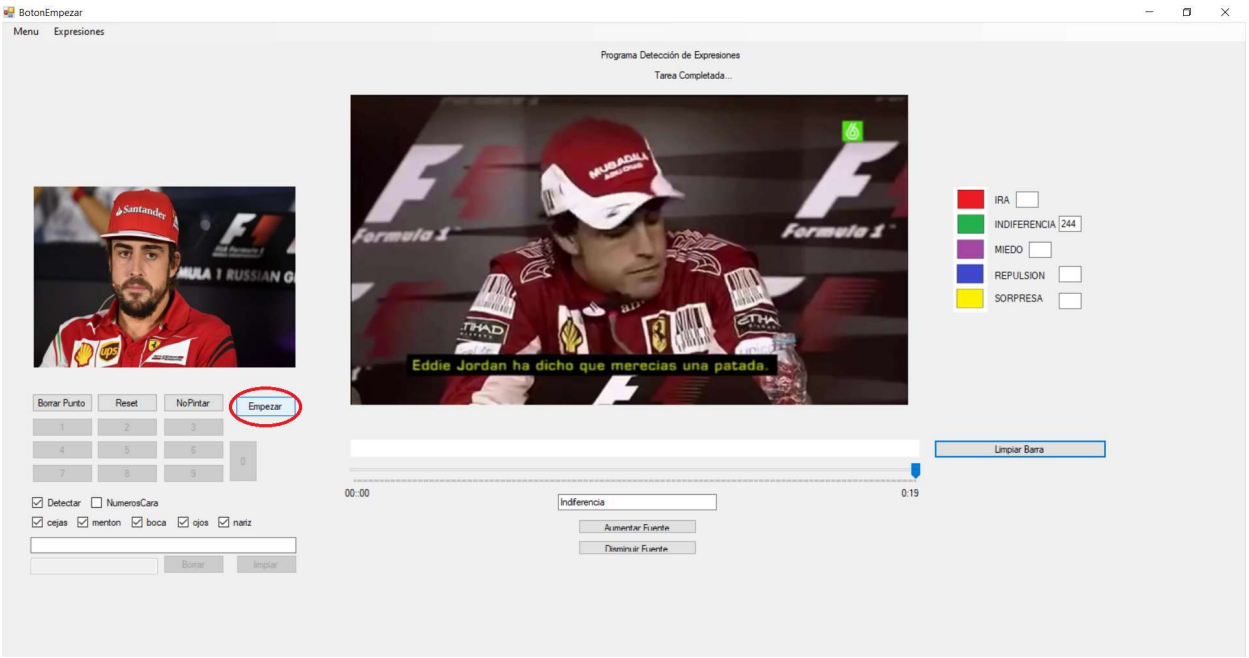

Figura 50 Guía de uso-Captura 8

Si no hacemos nada el programa finaliza y nos dice los fotogramas que se han detectado de cada emoción.

A medida que se está reproduciendo podemos usar los botones redondeados en la Figura para usar las características que ofrece el software.

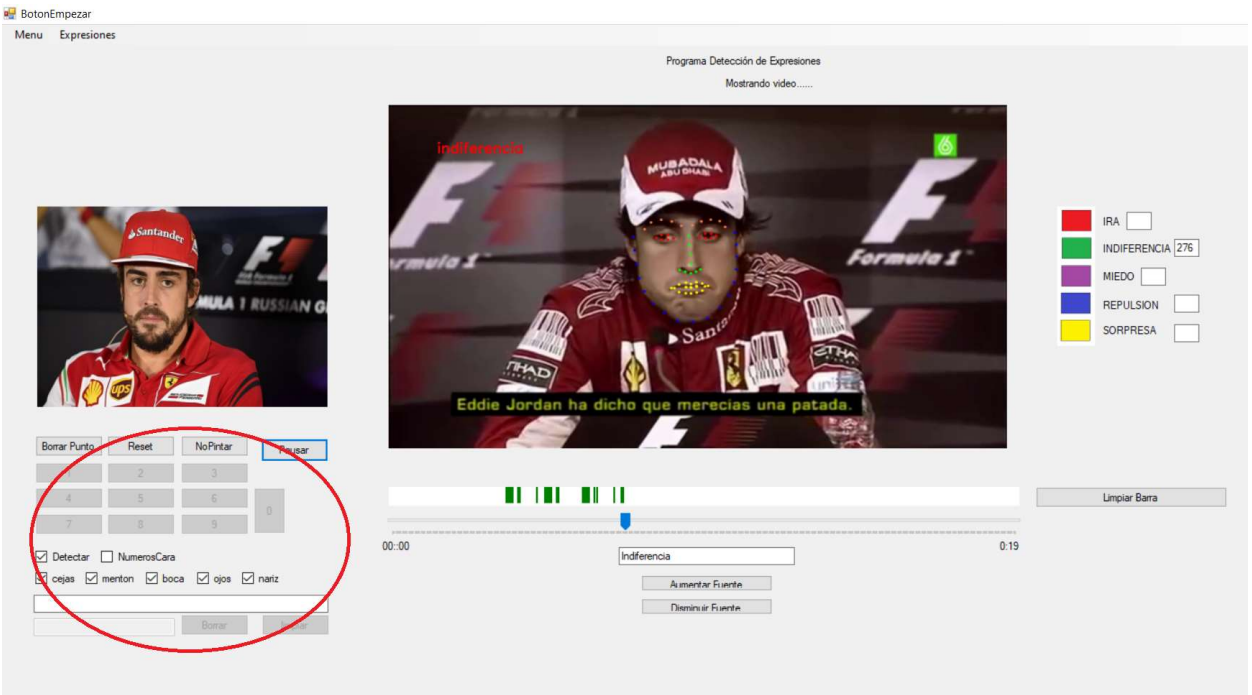

Figura 51 Guía de uso-Captura 9

5.2 Análisis de los vídeos de referencia

5.2.1 Análisis de *Miedo*

El resultado del vídeo es muy satisfactorio, se detecta en todo momento la expresión de miedo en el momento exacto. En la figura 52 observamos la imagen usada como modelo de referencia.

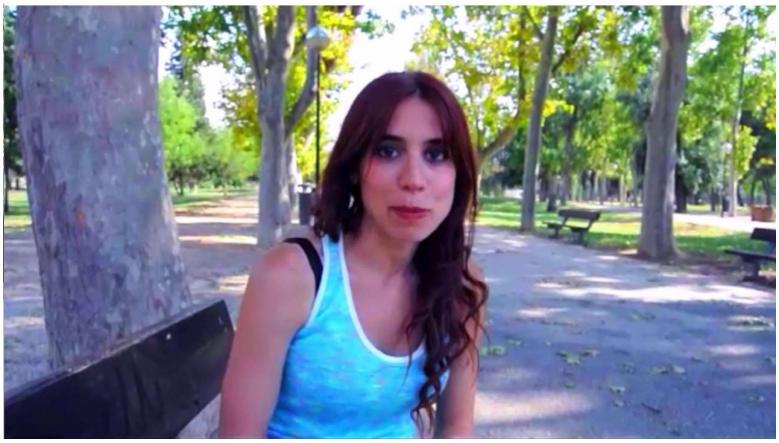

Figura 52 Imagen de referencia para el vídeo de *miedo*

En la Figura 53 observamos la salida final del programa.

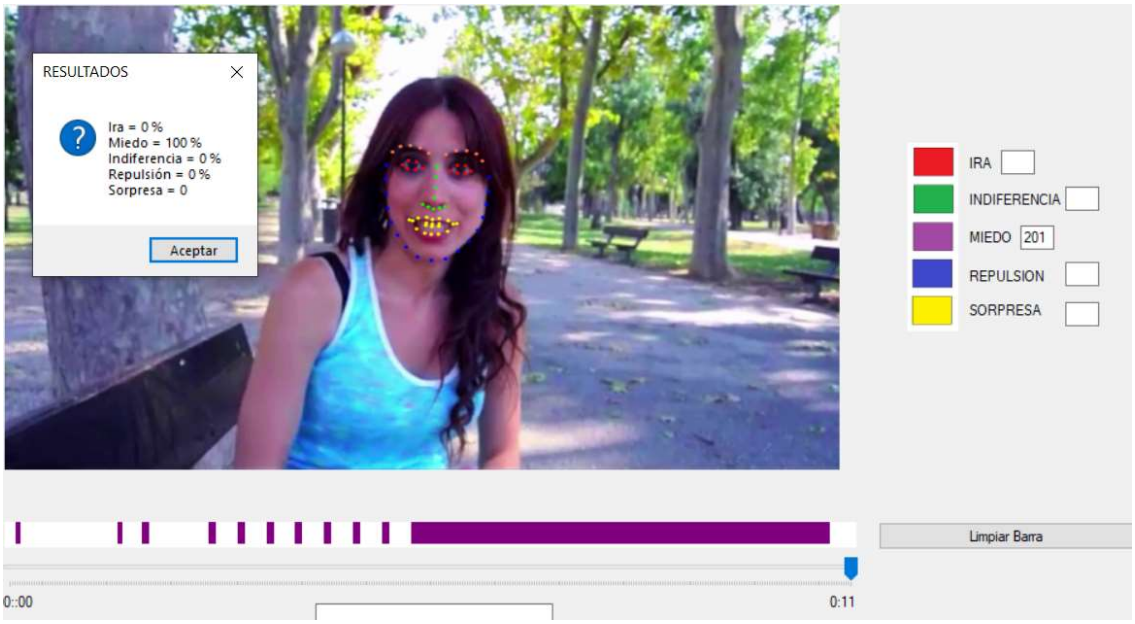

Figura 53 Resultado del vídeo de *miedo*

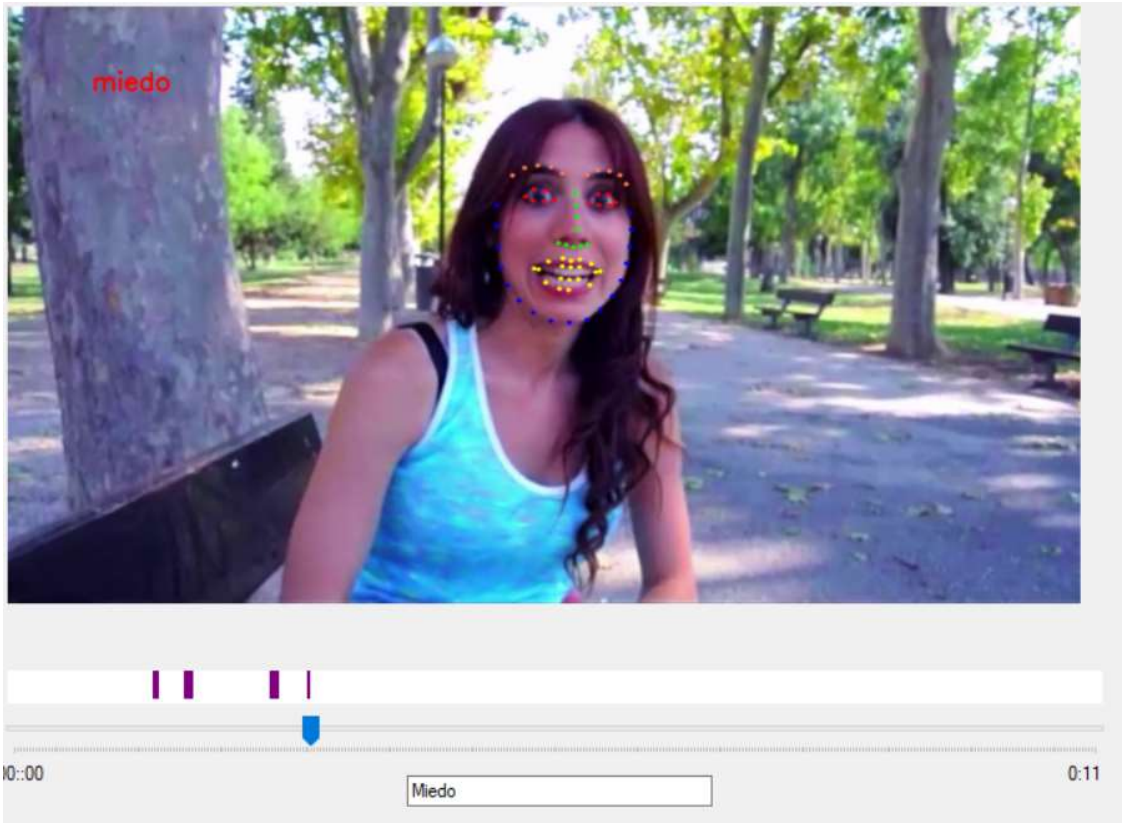

Figura 54 Un momento determinado donde se detecta *miedo*

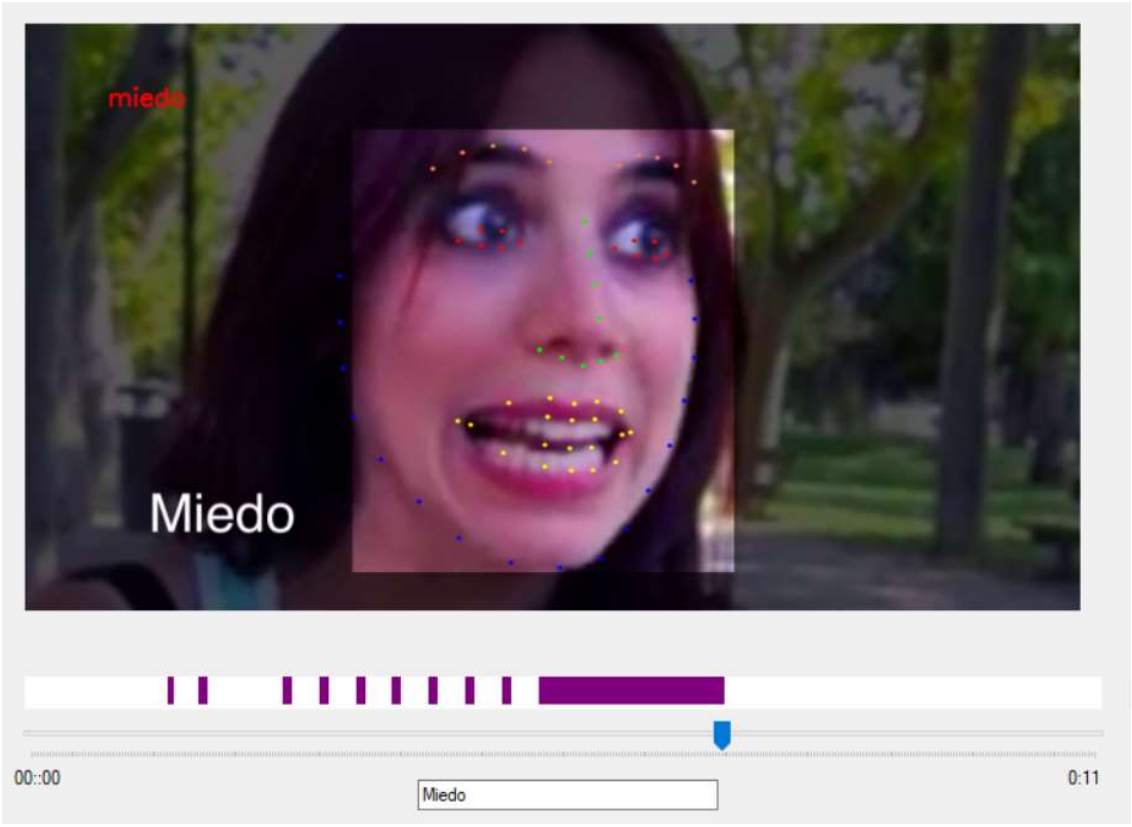

Figura 55 Momento en el que recuadran que se debe detectar *miedo*

En las Figuras 54 y 55 podemos observar distintos fotogramas donde se detecta miedo.

El programa da como resultado 201 fotogramas donde se ha detectado miedo. En ningún otro fotograma se ha detectado otra emoción. En la Figura 56 podemos observar el resultado gráficamente.

Figura 56 Resultados numéricos del vídeo de referencia *miedo*

5.2.2 Análisis de *Ira*

Para esta emoción analizamos dos vídeos de dos actores independientes.

En ambos casos se mezclan varias expresiones y a continuación vamos a analizar por qué sucede.

5.2.2.1 Vídeo de referencia de *Ira* 1

Empezamos mostrando en la Figura 57 la imagen usada como modelo.

Figura 57 Imagen usada como referencia en vídeo de *Ira* 1

En la Figura 58 vemos como se mezclan varios colores en el recuadro. Estos son morado, rojo y verde que corresponden a *miedo*, *ira* e *indiferencia*.

A continuación, analizaremos los fotogramas donde se ha detectado cada emoción.

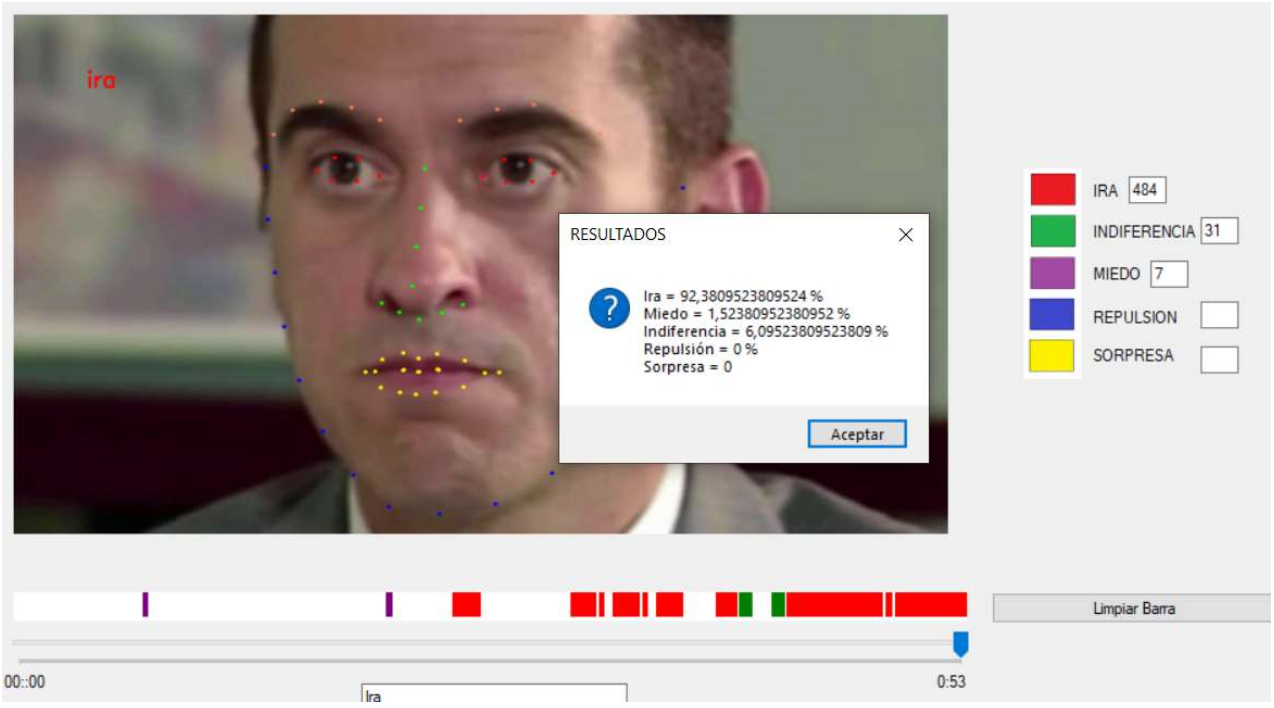

Figura 58 Resultado de *ira* - vídeo de referencia 1

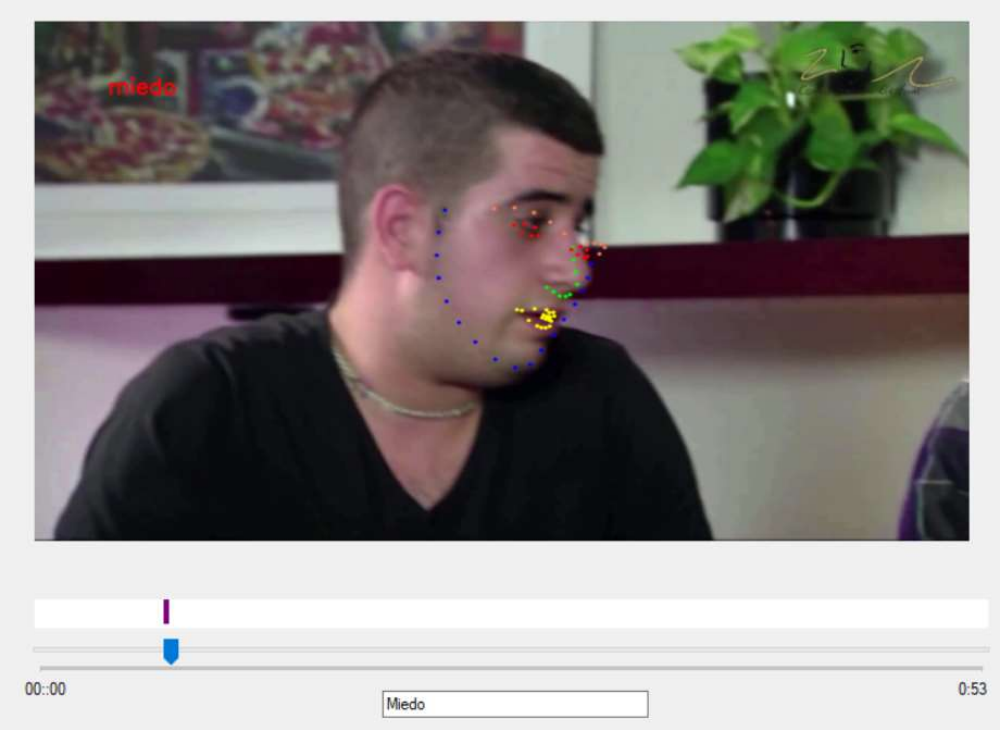

Figura 59 Captura del vídeo donde se detecta *miedo*

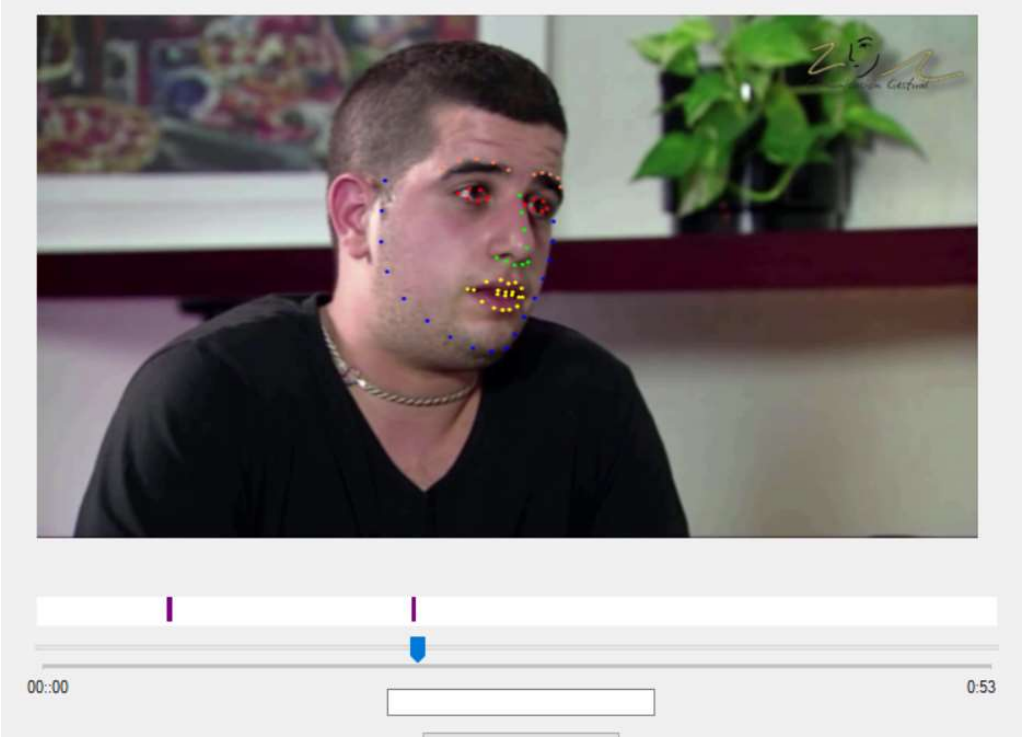

Figura 60 Mismo ejemplo que figura 45

Como vemos en las Figuras 59 y 60 los fotogramas donde se detecta *miedo* corresponden a un instante del vídeo donde cambia la toma y el sujeto a analizar. Además, en el caso de la Figura 46 el programa no está representando los puntos donde debería.

Al ser otro sujeto el que está bajo análisis y la imagen de referencia no corresponde con una de éste, el programa no funciona de manera óptima.

Para estos casos podemos hacer uso del botón para que no detecte durante estos intervalos.

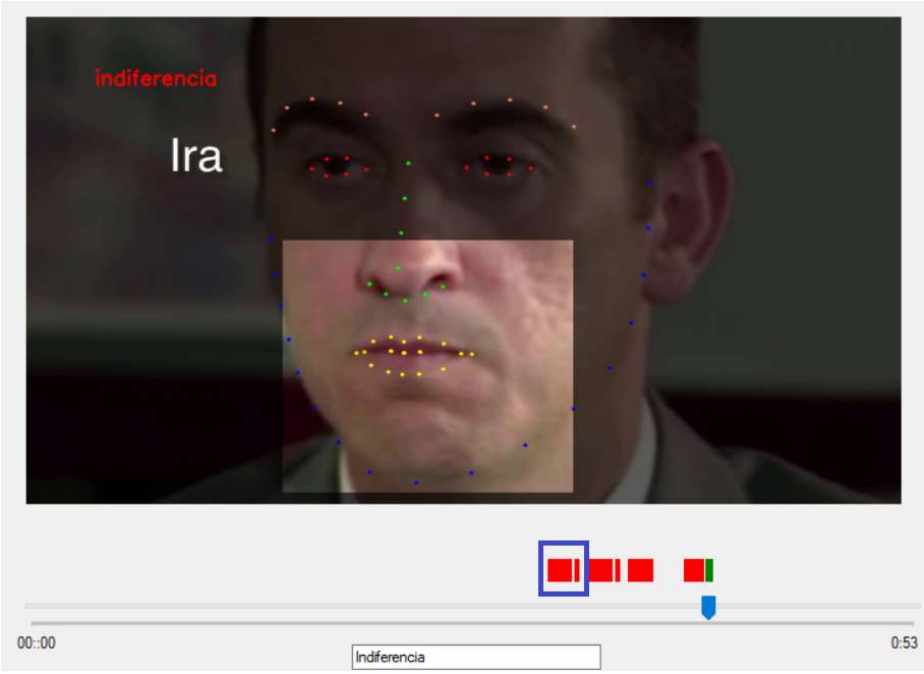

Figura 61 Momento donde se debería detectar *ira* y se detecta *indiferencia*

En la Figura 61 tenemos recuadrado en azul un instante donde el programa está detectando *ira* y durante un fotograma no detecta nada. Esto se debe a que el actor parpadea un instante y como cierra los ojos ya no se detecta *ira*.

Si nos fijamos en la misma figura, en el momento donde está pausado el vídeo, el actor cierra los ojos, pero se da la condición de que al ser a cámara lenta hay un número determinado de fotogramas seguidos en los que tiene los labios apretados y los ojos más cerrados, por lo que detecta *indiferencia*.

En la Figura 62 el actor vuelve a tener los ojos abiertos y de nuevo detecta *ira*.

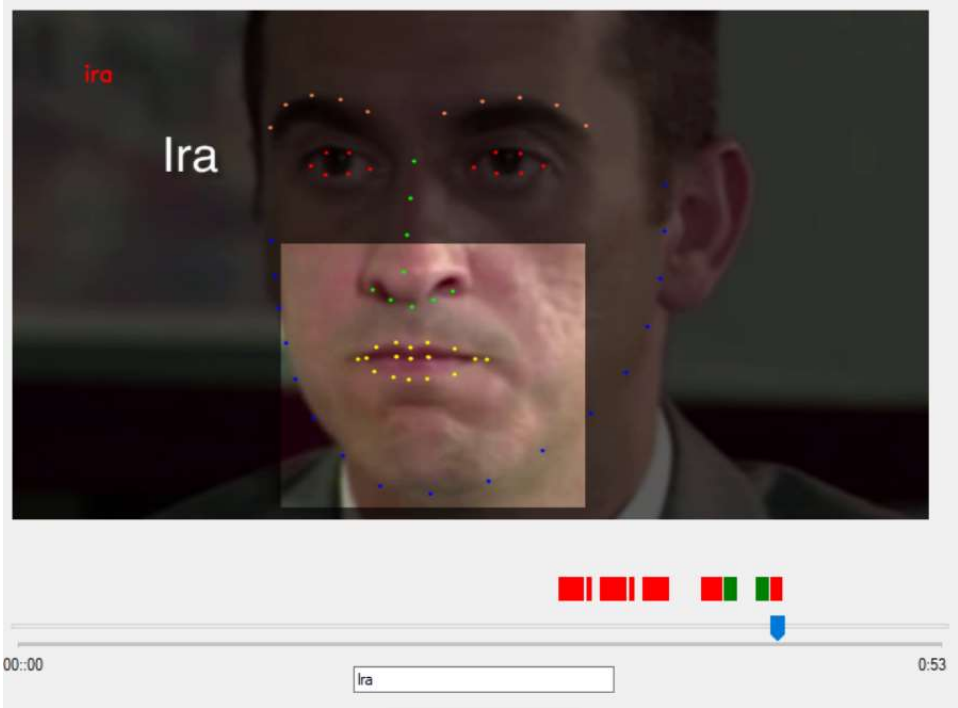

Figura 62 Momento donde se detecta *ira*

Se detectan 484 fotogramas de *ira*, 31 de *indiferencia* y 7 de *miedo*. Se representa gráficamente en la Figura 63.

Figura 63 Resultados numéricos de *ira* - vídeo de referencia 1

5.2.2.2 Vídeo de referencia de *Ira* 2

Comenzamos con la imagen usada como referencia en el vídeo de imagen de *Ira* 2.

Figura 64 Imagen usada como referencia para el vídeo de Ira 2

En la Figura 65 observamos la salida final del segundo vídeo de ira. En este vídeo de nuevo se vuelven a mezclar emociones, en este caso *sorpresa* e *ira*.

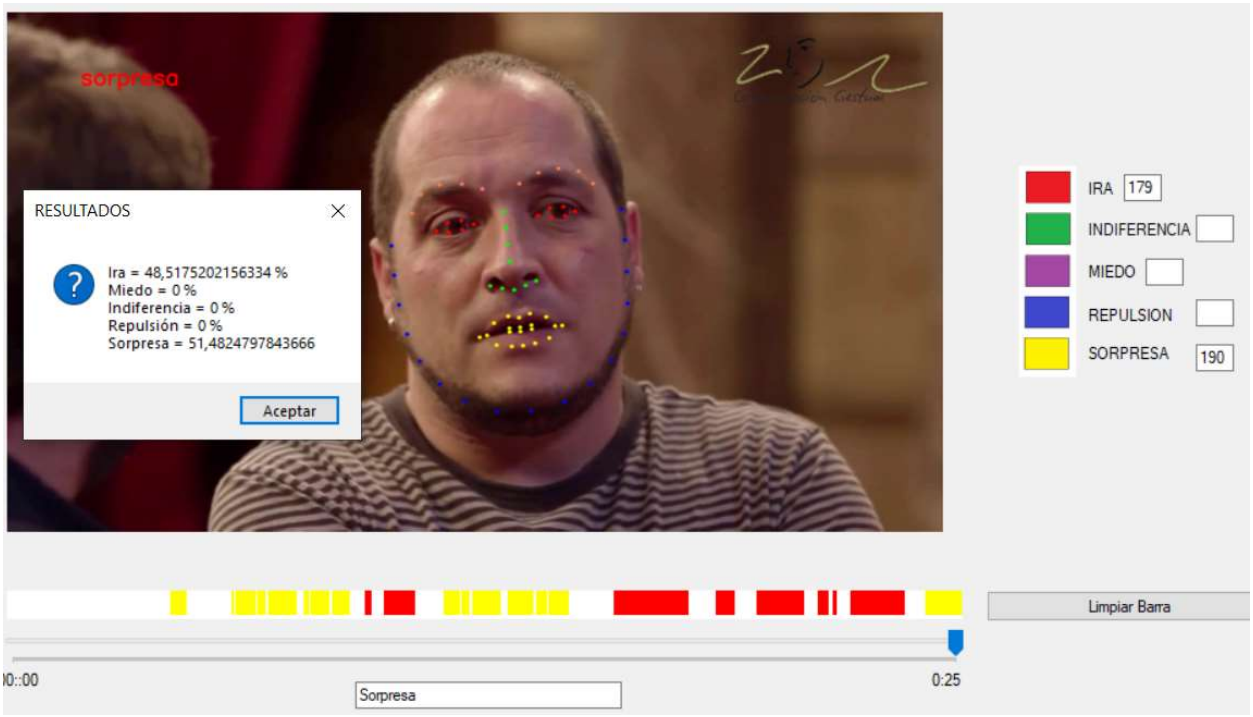

Figura 65 Resultado de *ira* - vídeo referencia 2

En la Figura 66 vemos una captura del vídeo en el momento exacto que se detecta *sorpresa*.

El sujeto en estudio permanece durante varios fotogramas seguidos con la boca abierta. Como se cumple esta condición se detecta sorpresa.

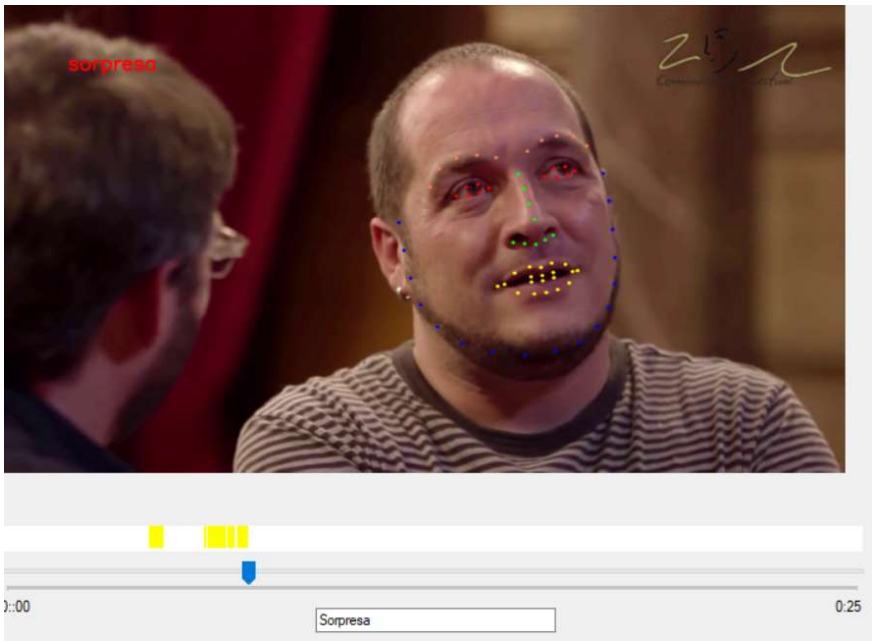

Figura 66 Momento del vídeo de referencia de *ira* 2 donde se produce *sorpresa*

En la Figura 67 vemos una captura de un momento determinado donde se detecta *ira.* El actor aprieta los labios con los ojos abiertos durante una serie de fotogramas seguidos y se cumple la condición para detectar *ira*.

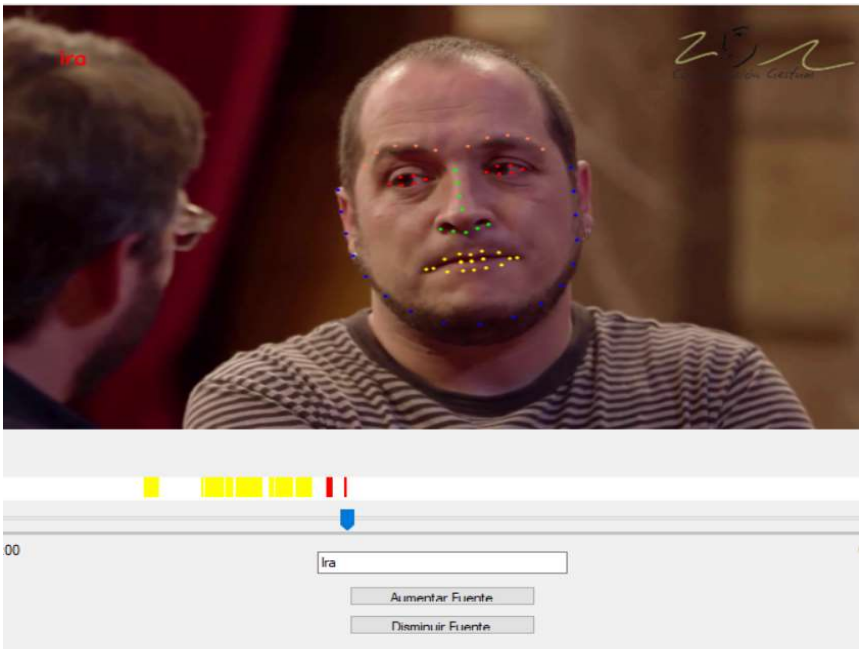

Figura 67 Momento del vídeo de referencia de *ira* 2 donde se produce *ira*

Por último, en la Figura 68 vemos el momento que nos recuadran donde se desea que se detecte *ira* y así lo hace el programa.

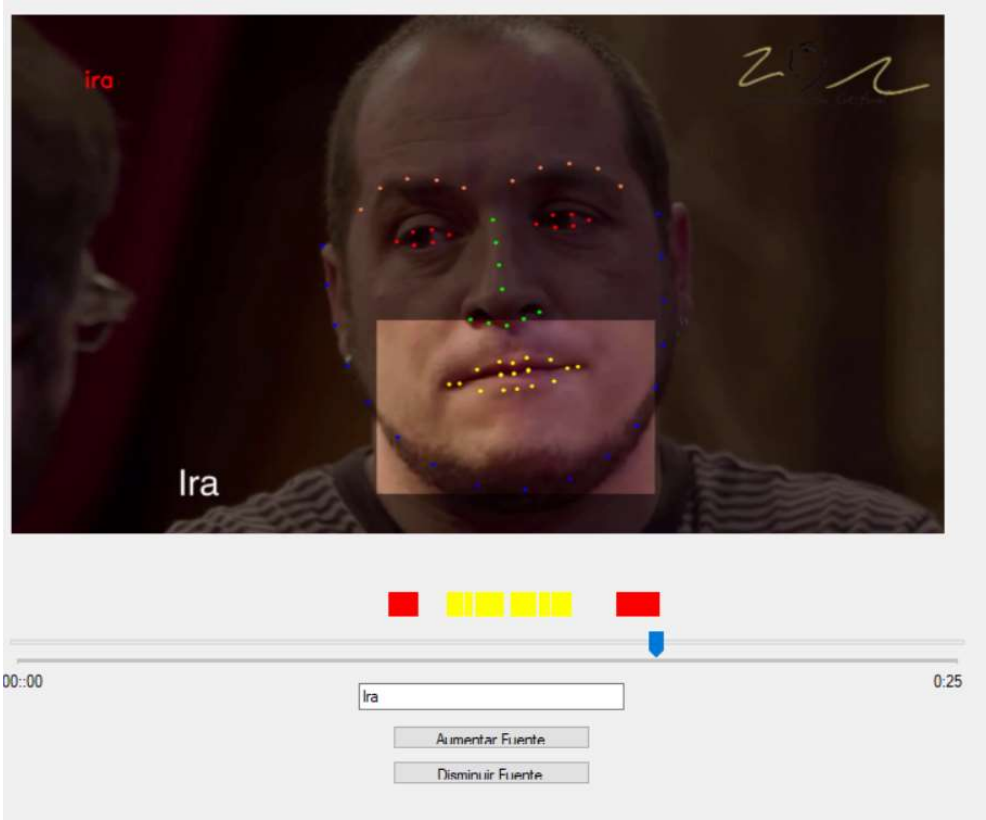

Figura 68 Momento en el que se detalla que se debe detectar *ira* y se detecta *ira*

Se detectan 179 fotogramas de *ira* y 190 de *sorpresa*. Se representa gráficamente en la Figura 69.

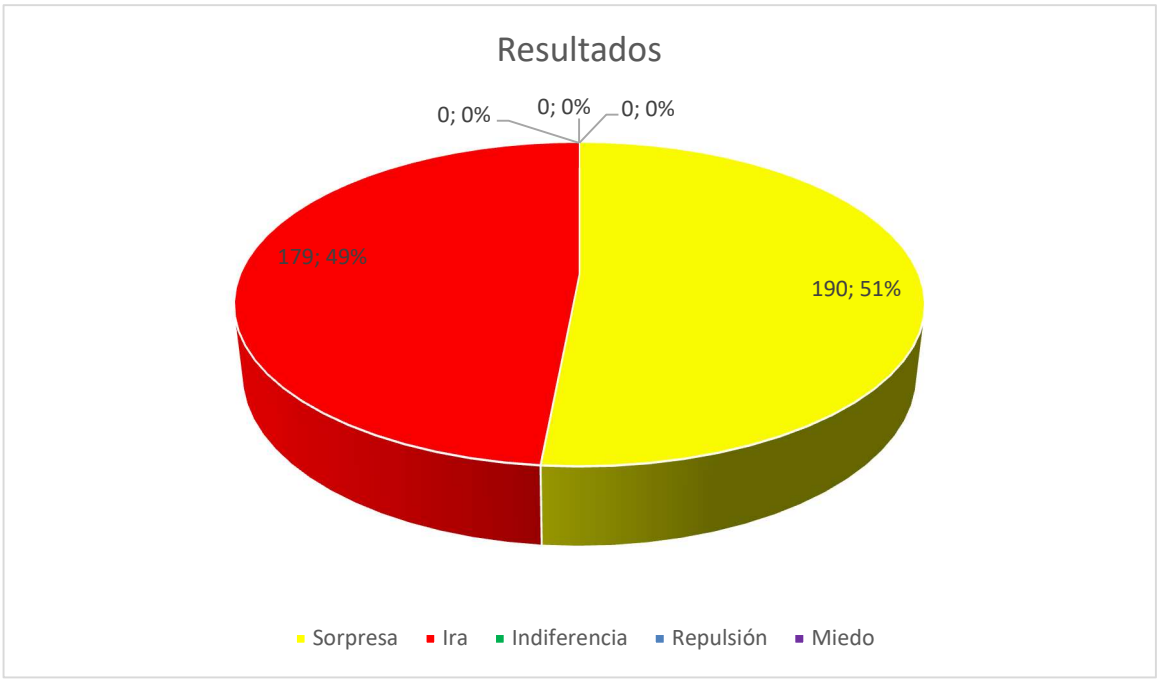

Figura 69 Resultados numéricos de ira - vídeo de referencia 2

5.2.3 Análisis de *Indiferencia*

El resultado del análisis de este vídeo es indiferencia en todo momento.

En la Figura 70 vemos el aspecto de la interfaz gráfica una vez finalizado el análisis. Podemos observar el recuadro superior al control de seguimiento del vídeo con sólo franjas verdes a intervalos.

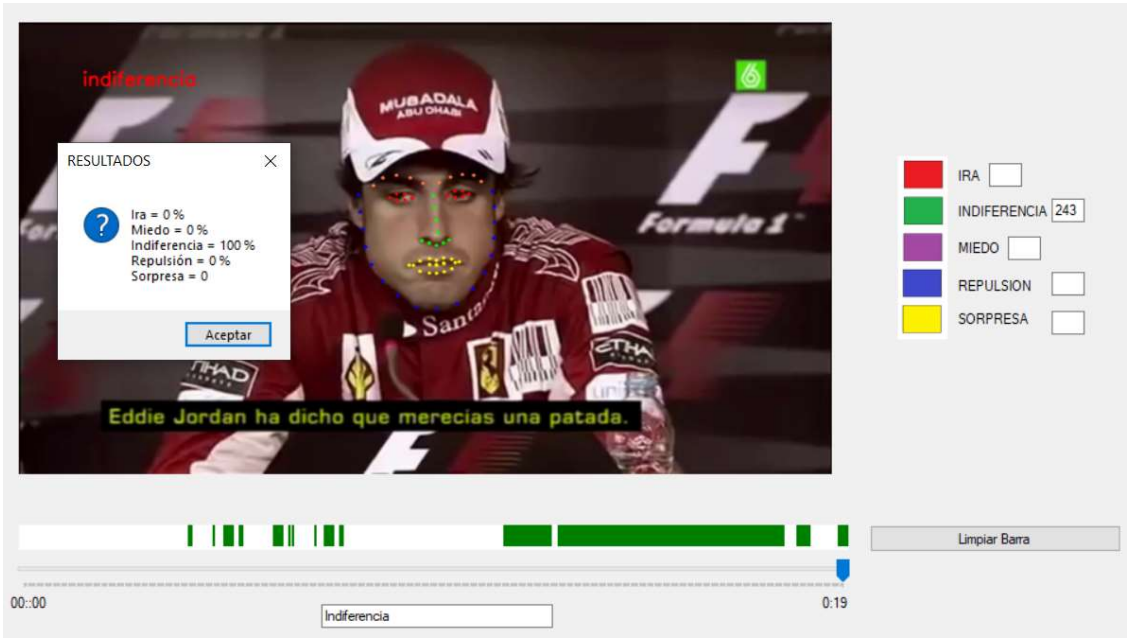

Figura 70 Resultado de *indiferencia* vídeo de referencia

Se detectan 243 fotogramas de indiferencia. En la Figura 71 se representa gráficamente.

Figura 71 Resultados numéricos vídeo indiferencia

5.2.4 Análisis de *Sorpresa*

Se detecta *sorpresa* e *indiferencia*. En la Figura 72 podemos ver la imagen usada como modelo en el vídeo de *sorpresa*.

Figura 72 Imagen usada como referencia en el vídeo de *Sorpresa*

En la Figura 73 podemos ver el resultado final.

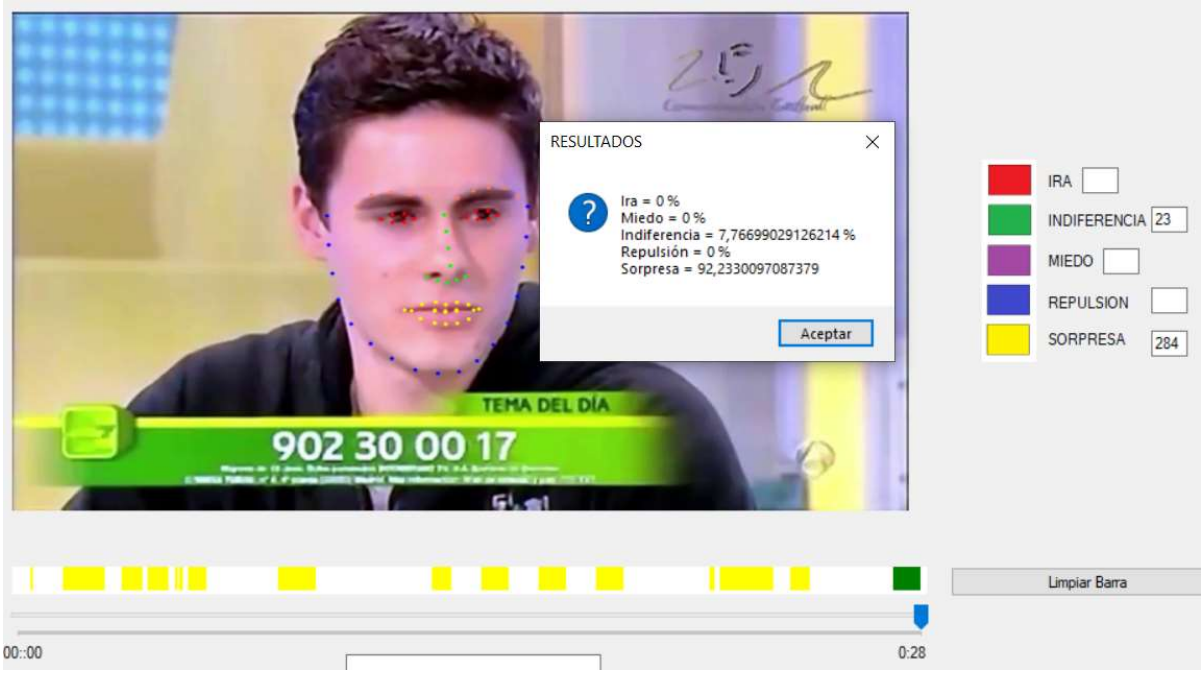

Figura 73 Resultado vídeo de referencia *sorpresa*

En la Figura 74 vemos el momento en el que se debe detectar *sorpresa* y así lo hace.

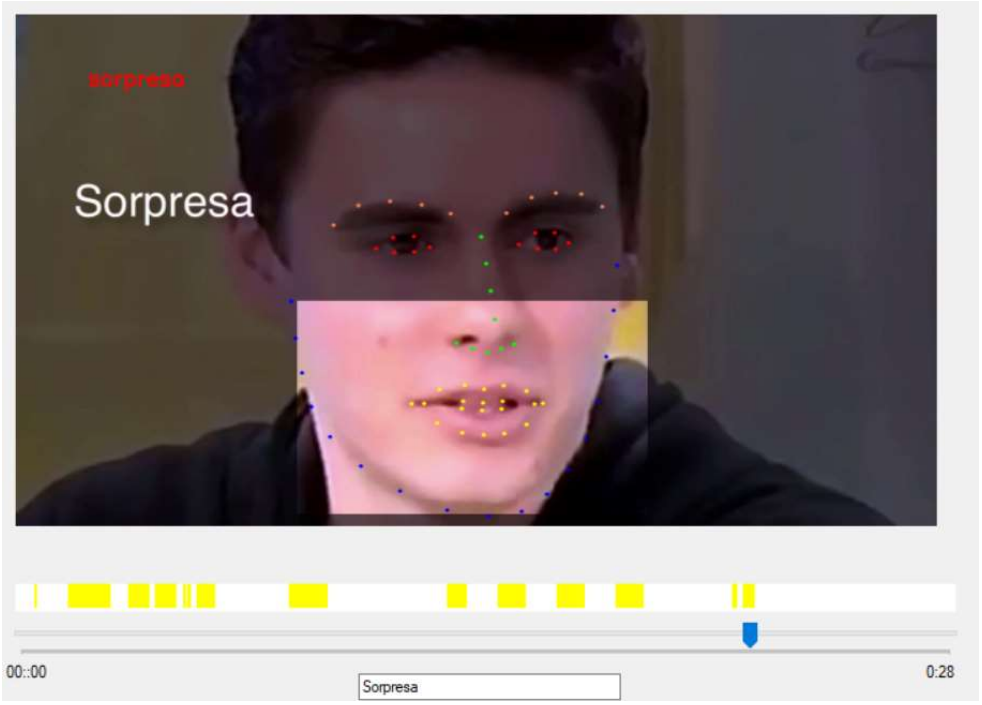

Figura 74 Momento en el que se debe detectar *sorpresa*-Vídeo de referencia *sorpresa*

En la Figura 75 vemos el momento en el que se detecta *indiferencia*, el sujeto bajo estudio cierra parcialmente los ojos y aprieta los labios.

Figura 75 Momento en el que se detecta *indiferencia*-Vídeo referencia *sorpresa*

Se detectan 284 fotogramas de *sorpresa* y 23 de *indiferencia*. Lo vemos gráficamente en la Figura 76.

Figura 76 Resultados numéricos vídeo de referencia *sorpresa*

5.2.5 Análisis de *Repulsión*

Es uno de los vídeos más complicados por la cantidad de expresiones que realizar el actor. Se aprietan los labios,

se cierran los ojos, se bajan las cejas etc. El programa va mezclando entre *ira*, *indiferencia* y *repulsión*.

En la Figura 77 vemos la imagen usada como referencia.

Figura 77 Imagen usada como referencia en el vídeo de *repulsión*

Podemos ver el resultado final en la Figura 78.

Vamos a analizar algún fotograma de manera individual para ver por qué el programa detecta la emoción en ese instante determinado.

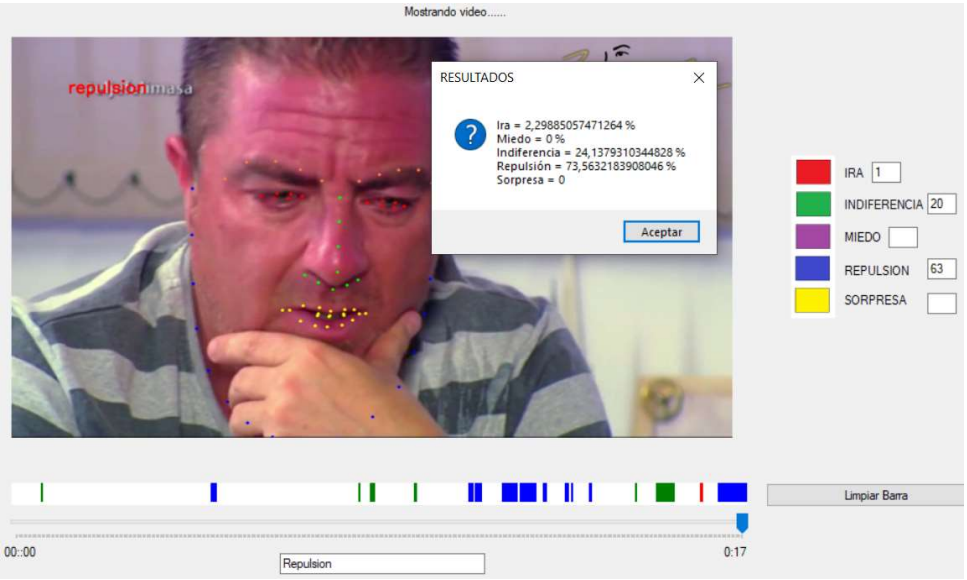

Figura 78 Resultado de *repulsión*

En la Figura 79 vemos una captura del vídeo donde se detecta indiferencia. El actor se mantiene con los ojos cerrados y con los labios apretados durante un número determinado de fotogramas y detecta indiferencia.

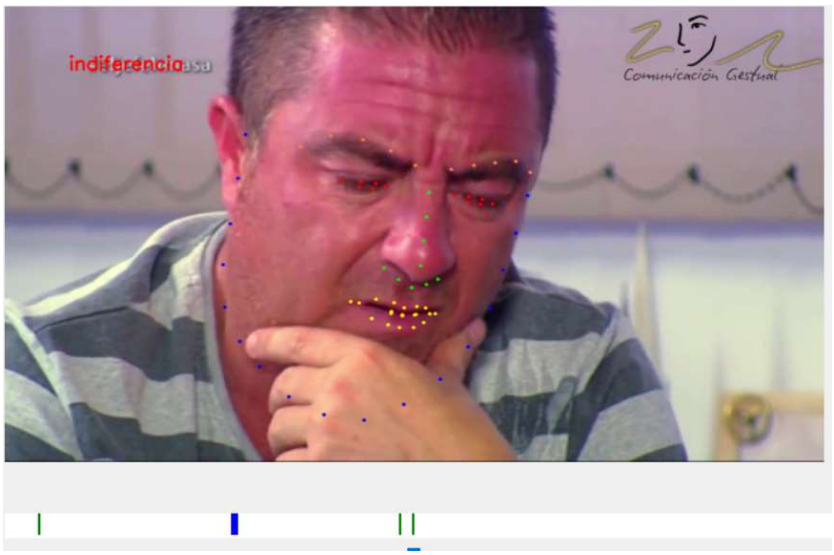

Figura 79 Momento en el que se detecta *indiferencia*

En la Figura 80 podemos ver gráficamente los resultados numéricos que arroja la herramienta.

Se detectan 63 *frames* de *repulsión*, 20 de *indiferencia* y 1 de *ira*.

Figura 80 Resultados numéricos vídeo referencia *repulsión*

5.3 Base de datos y análisis

Para tener otro punto de vista acerca de cómo funciona nuestra herramienta vamos a trabajar con una base de datos de vídeos de expresiones:

"*The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS)*" [13].

Esta base de datos contiene 7356 archivos con un tamaño total de 24,8 Gb.

Aparecen 24 actores realizando expresiones como tranquilidad, felicidad, tristeza, enfado, miedo, sorpresa y disgusto y cada una de estas la realiza en dos niveles de intensidad (normal y fuerte).

Aunque las expresiones no son exactamente las mismas que detecta el programa le vamos a pasar distintos vídeos para ver cómo actúa la herramienta.

Los vídeos están nombrados siguiendo un código.

Ejemplo: 01-01-05-01-01-02-05.mp4

Los dos primeros dígitos corresponden a la modalidad del vídeo *(01 = full-AV, 02 = video-only, 03 = audio-only*).

El siguiente par corresponde al canal vocal (*01* = hablado*, 02* = cantado).

El tercer par es el que más nos interesa, corresponde con la emoción que refleja el actor (01 = neutral, 02 = calma, 03 = felicidad, 04 = tristeza, 05 = enfado, 06 = miedo, 07 = disgusto, 08 = sorpresa).

El cuarto par es la intensidad de la emoción que se refleja (01 = normal, 02 = fuerte).

El quinto par corresponde a lo que dice el actor (01 = *"Kids are talking by the door", 02 = "Dogs are sitting by the door*").

El sexto par, la repetición (01 = primera repetición, 02 = segunda repetición).

Por último, el número del actor.

5.3.1 Actor 05

En la siguiente figura vemos la captura que vamos a tomar como imagen de referencia para este actor.

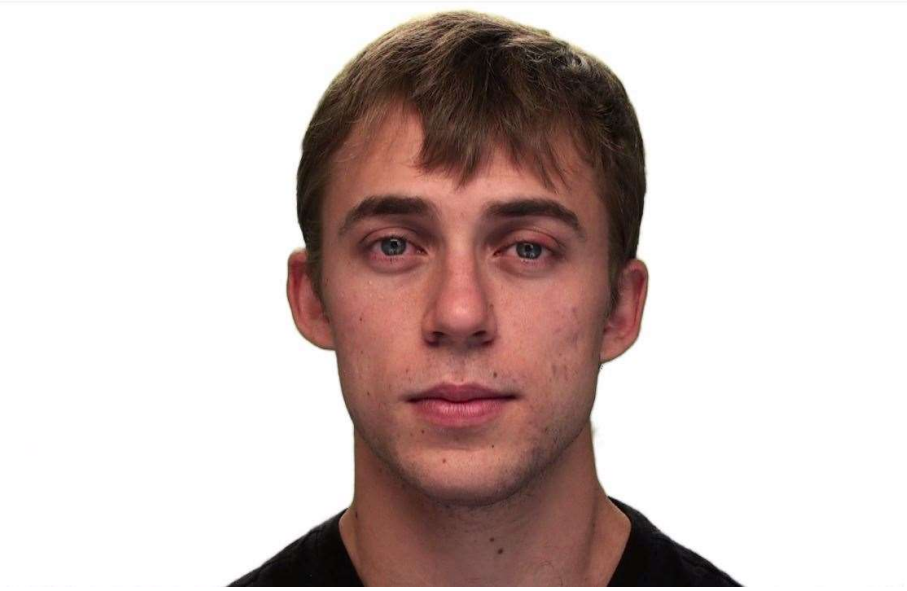

Figura 81 Imagen de referencia actor 05

5.3.1.1 Vídeo 01-01-05-01-01-02-05.mp4

Del título, observando el tercer par de dígitos, podemos determinar que es un vídeo relacionado con la emoción de enfado. En la Figura 82 podemos ver la salida de la herramienta. Detecta en la gran mayoría *sorpresa* debido a que habla a la vez que expresa esta emoción con la boca muy abierta.

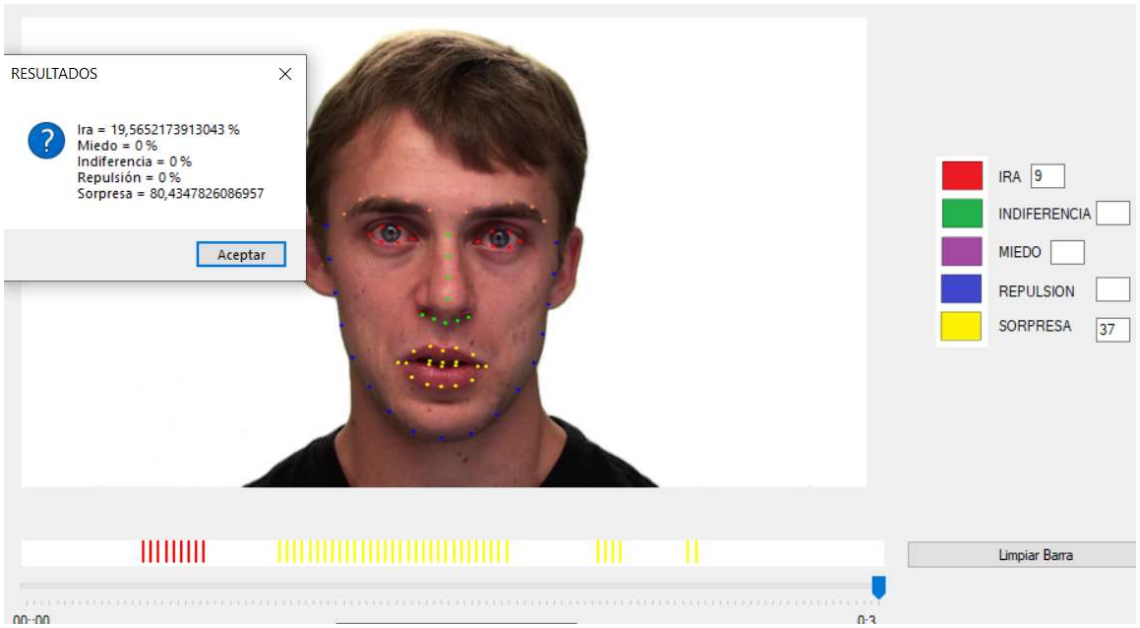

Figura 82 Resultado vídeo *enfado* actor 05

En la Figura 83 podemos ver que en el momento en el que el actor expresa ira de manera parecida a los vídeos usados como verdad de referencia la herramienta lo detecta fácilmente.

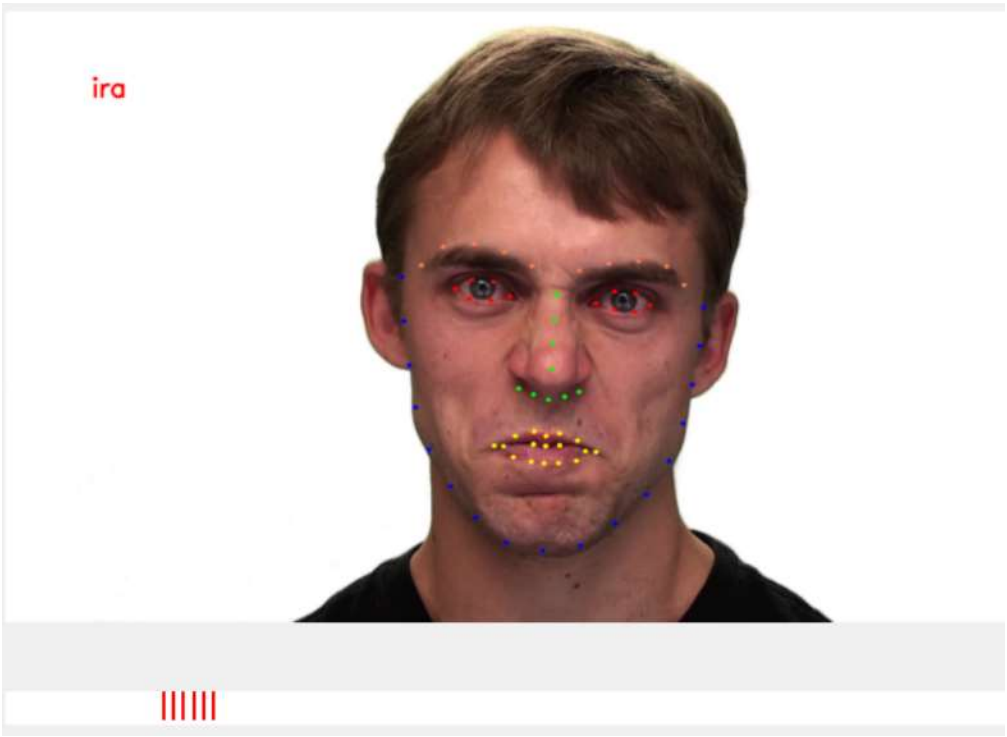

Figura 83 Instante en el que se detecta *ira*

A continuación, los resultados numéricos:

Figura 84 Resultados numéricos vídeo *enfado* actor 05

5.3.1.2 Vídeo 02-01-08-02-01-02-05.mp4

Vídeo de *sorpresa* para el actor 05.

En la Figura 85 vemos el resultado de la herramienta cuando le pasamos el vídeo del título.

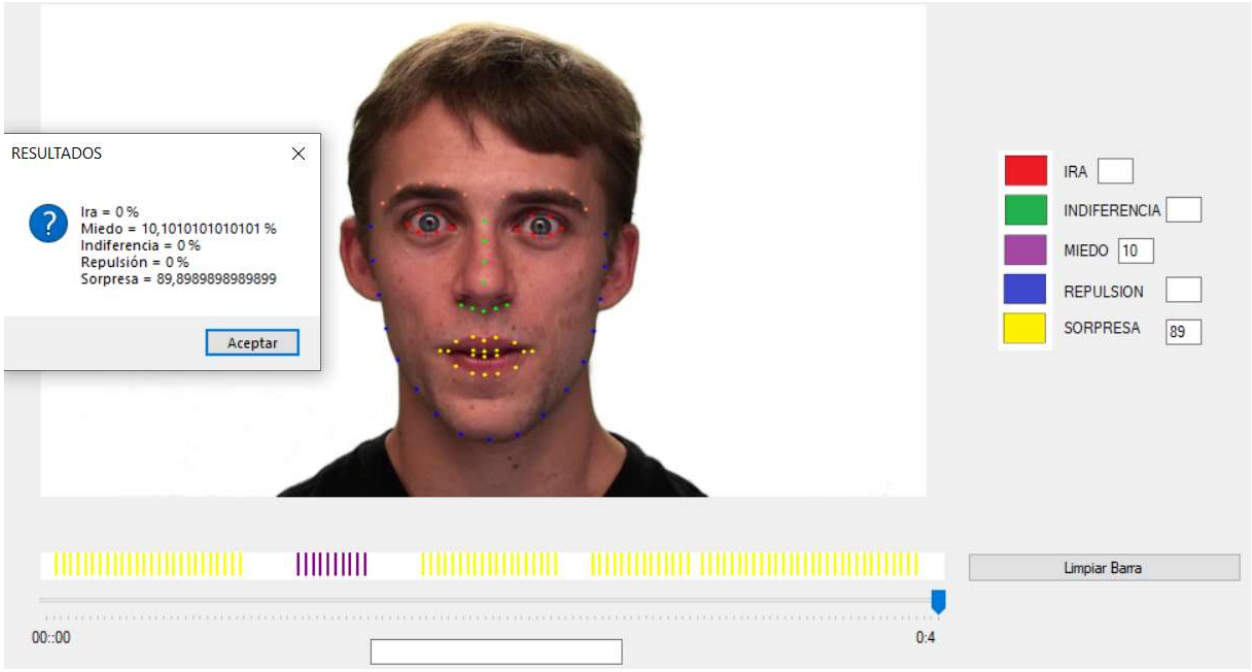

Figura 85 Resultado vídeo *sorpresa* actor 05

En las Figuras 86 y 87 podemos ver instantes en el que se detecta sorpresa y miedo. La diferencia entre estas dos imágenes es que el actor sube más las cejas cuando la herramienta está detectando miedo.

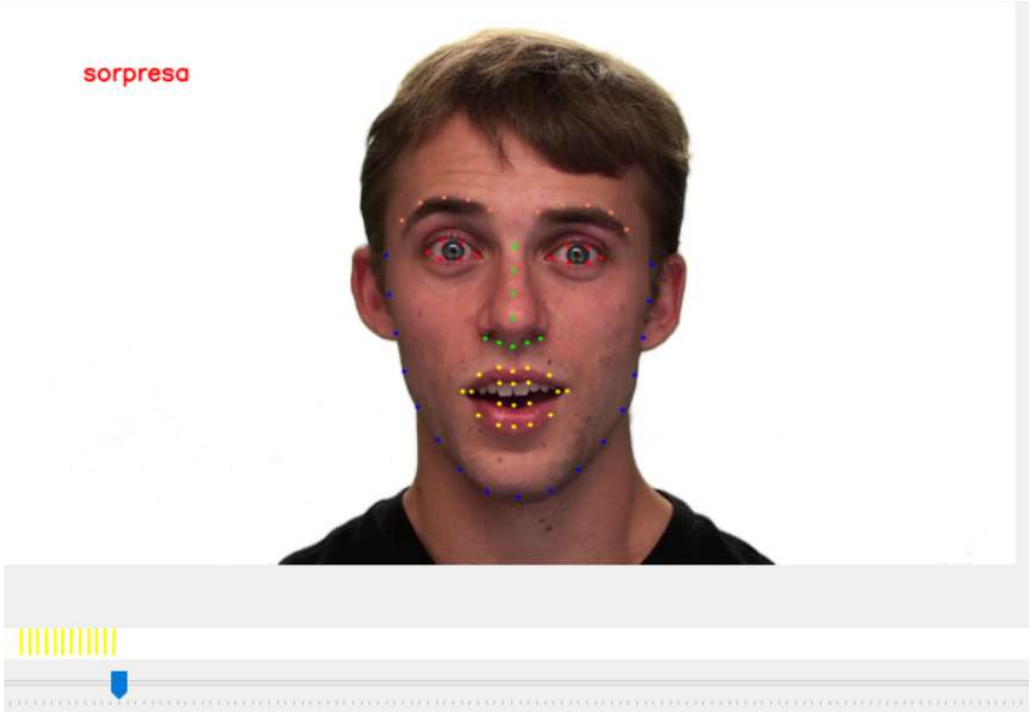

Figura 86 Instante en el que se detecta *sorpresa*

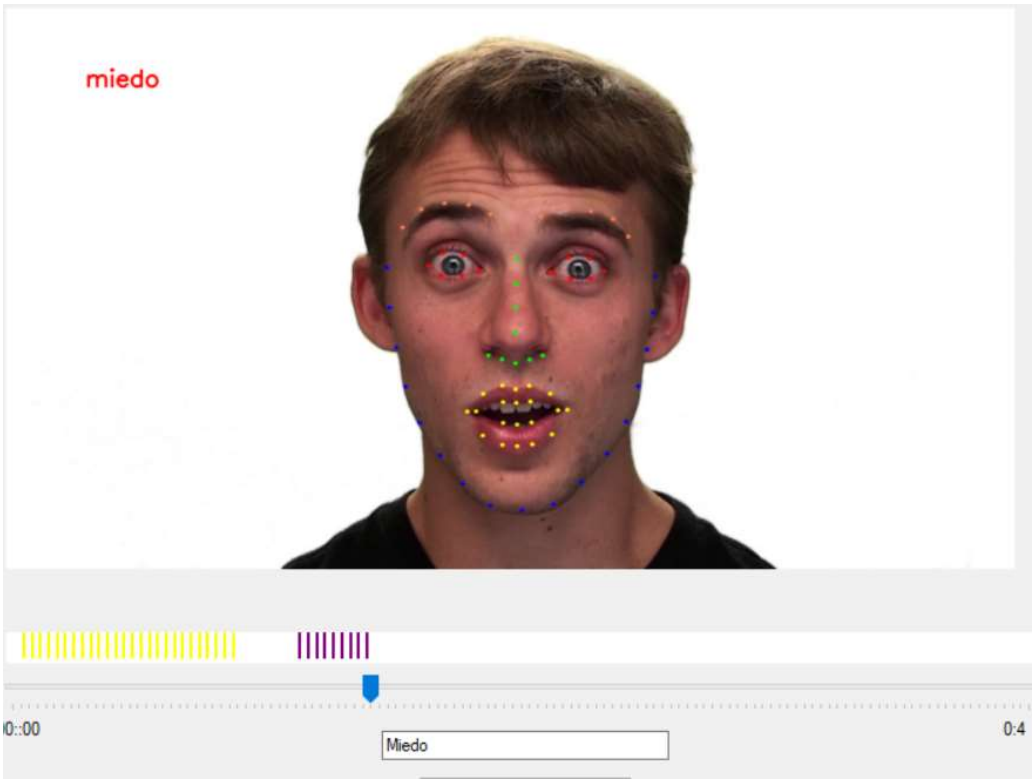

Figura 87 Instante en el que se detecta *miedo*

A continuación, vemos gráficamente los resultados numéricos.

Figura 88 Resultados numéricos actor 05 vídeo *sorpresa*

5.3.2 Actor 07

En la Figura 89 vemos la captura realizada para tomarla como imagen de referencia.

Figura 89 Actor 07 imagen de referencia

5.3.2.1 Vídeo 01-01-05-01-02-02-07.mp4

Durante el vídeo el actor muestra enfado gritando por lo que mantiene abierta la boca durante mucho tiempo y se detectan en muchos fotogramas *sorpresa*. En el momento en el que expresa enfado sin hablar el programa detecta *ira*. Lo vemos en la Figura 90:

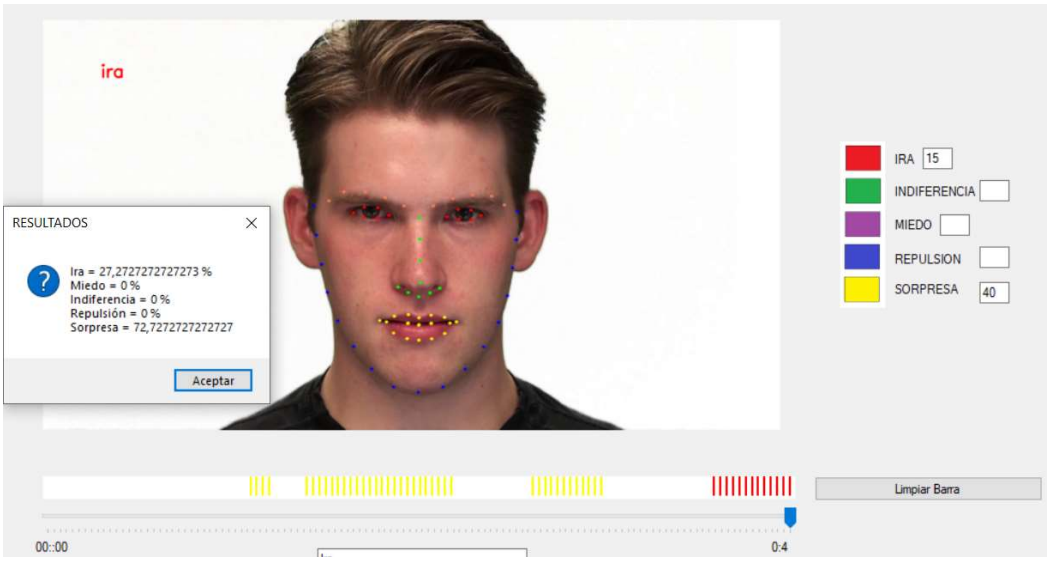

Figura 90 Resultado vídeo *enfado* actor 07

En la Figura 91 vemos un instante en el que se está detectando sorpresa debido a la boca abierta del actor:

Figura 91 Instante en el que se detecta sorpresa

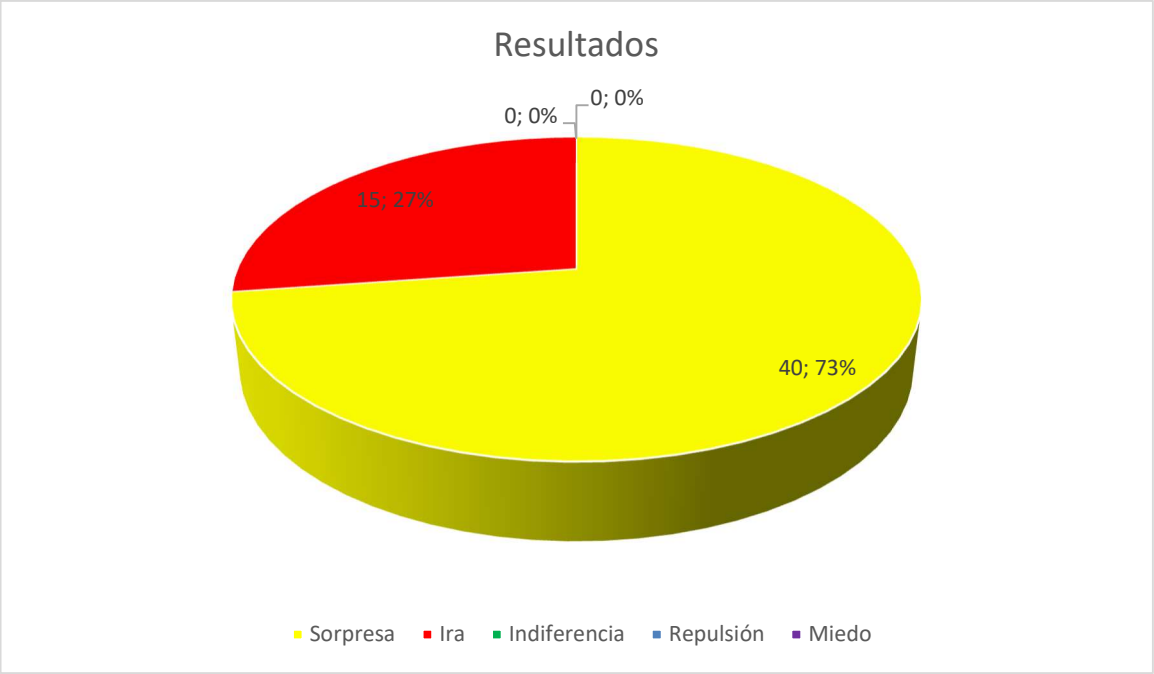

Figura 92 Resultados numéricos vídeo *enfado* actor 07

5.3.2.2 Vídeo 02-01-08-01-01-02-07.mp4

Del tercer par de dígitos del título del vídeo podemos obtener que el actor expresará la emoción de sorpresa.

Sin embargo, en la Figura 93 vemos como la herramienta lo único que detecta es *miedo* en todo momento.

Este actor expresa la emoción de sorpresa levantando las cejas por lo que se cumplen las dos condiciones para que se detecte miedo: Cejas levantadas y boca abierta.

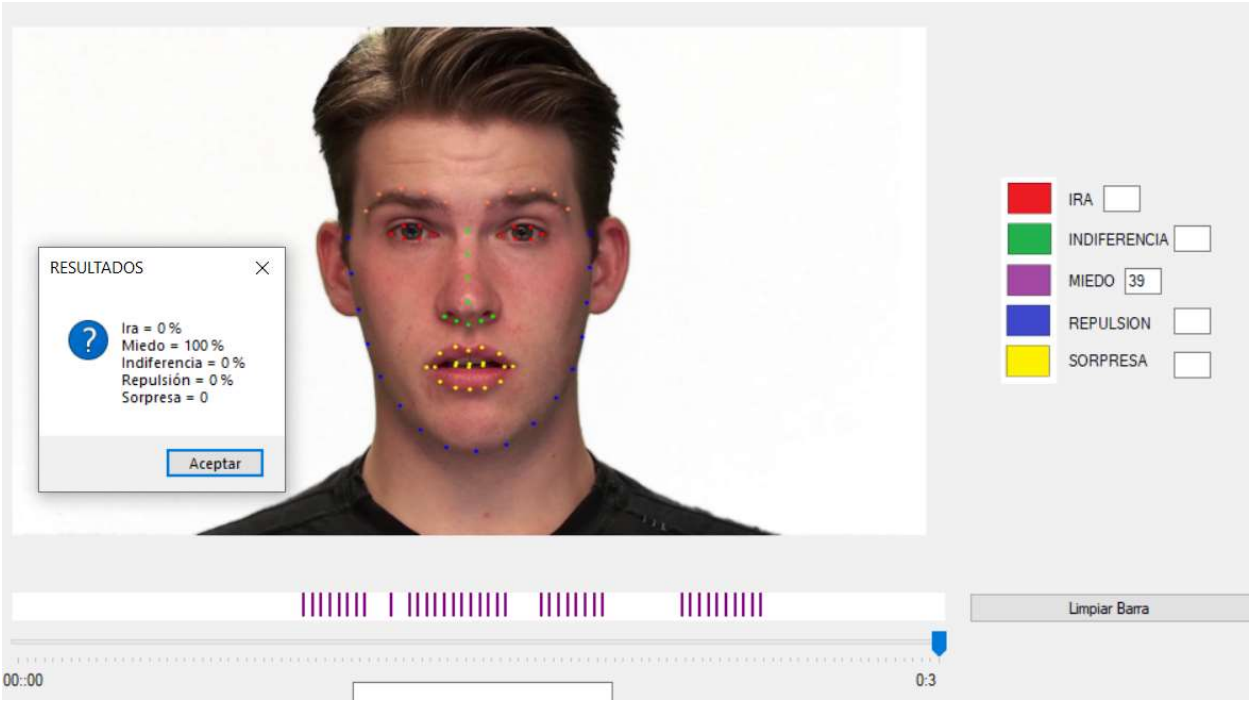

Figura 93 Resultado vídeo *sorpresa* actor 07

Resultados numéricos en forma de gráfica en la Figura 94.

Figura 94 Resultados numéricos vídeo *sorpresa* actor 07

5.3.2.3 Vídeo 01-01-06-02-02-01-07.mp4

En este caso el vídeo pasado es de la emoción de miedo. En la Figura 95 vemos como en este caso la herramienta funciona correctamente.

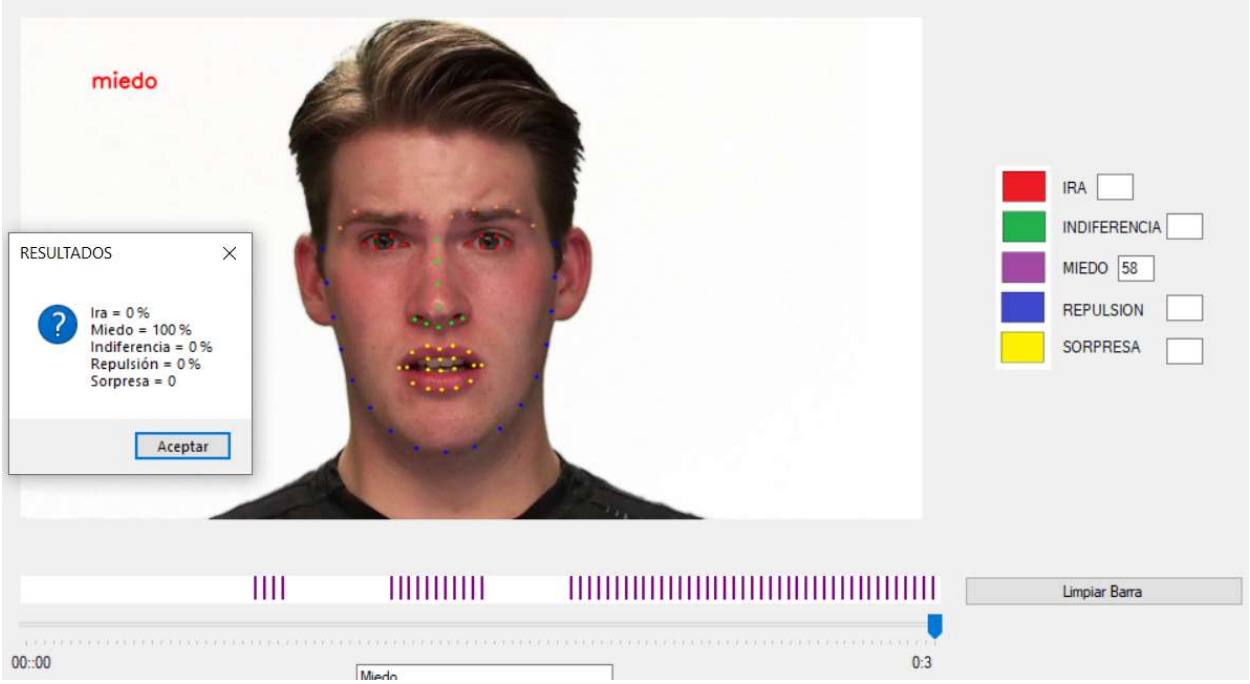

Figura 95 Resultado vídeo *miedo* actor 07

En la Figura 96 vemos gráficamente los resultados:

Figura 96 Resultados numéricos vídeo *miedo* actor 07

5.3.2.4 Experimento con el actor 07

Para comprobar que la herramienta funciona con otros vídeos y detecta todas las emociones con las que trabaja se ha realizado un experimento con el actor 07.

En distintos vídeos se han realizado capturas al actor.

En estas capturas el actor tiene una expresión que cumpliría con lo que el software entiende que es cada una de las emociones.

Se han convertido estas capturas a vídeo y se las hemos pasado a la herramienta.

En las Figuras siguientes podemos ver la salida para cada una de las emociones:

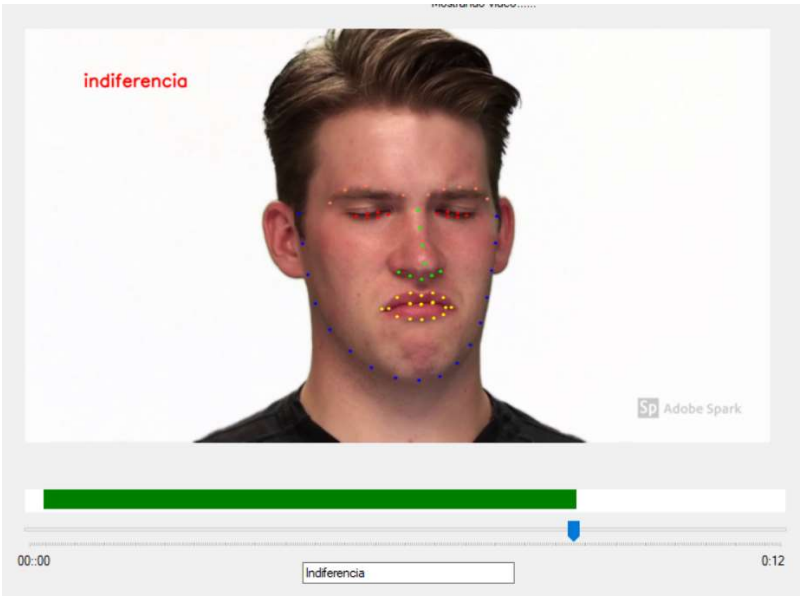

Figura 97 Captura de *indiferencia*

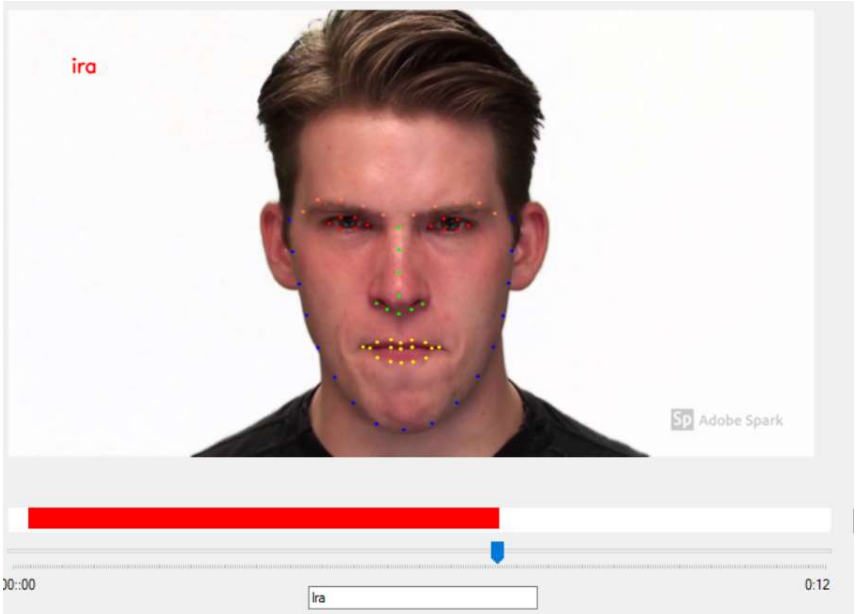

Figura 98 Captura de *ira*

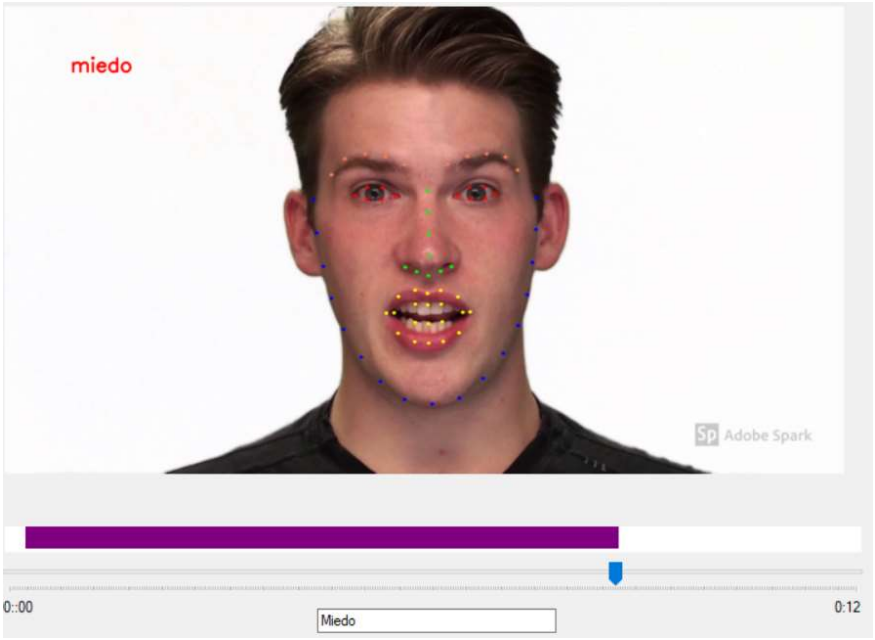

Figura 99 Captura de *miedo*

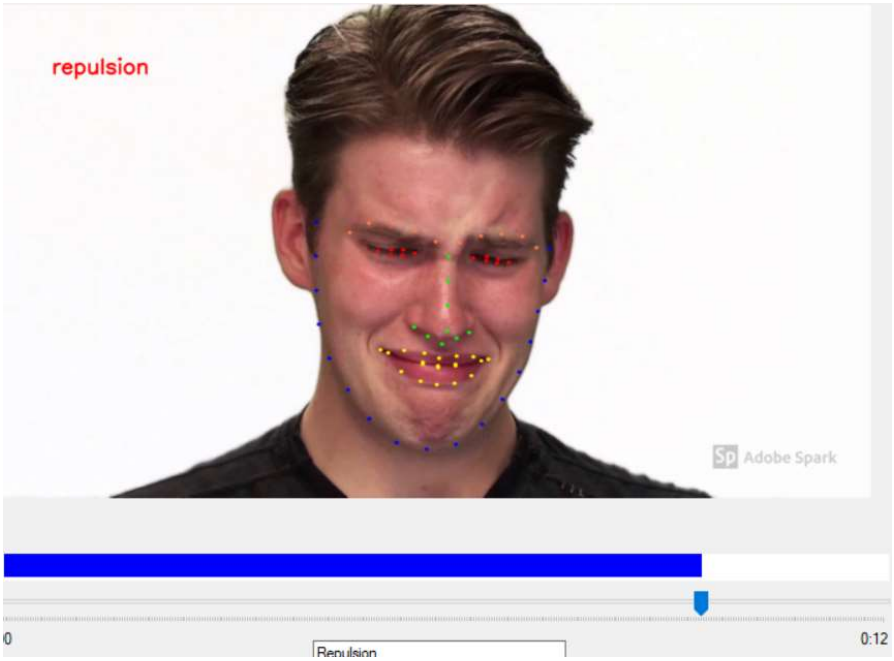

Figura 100 Captura de *repulsión*

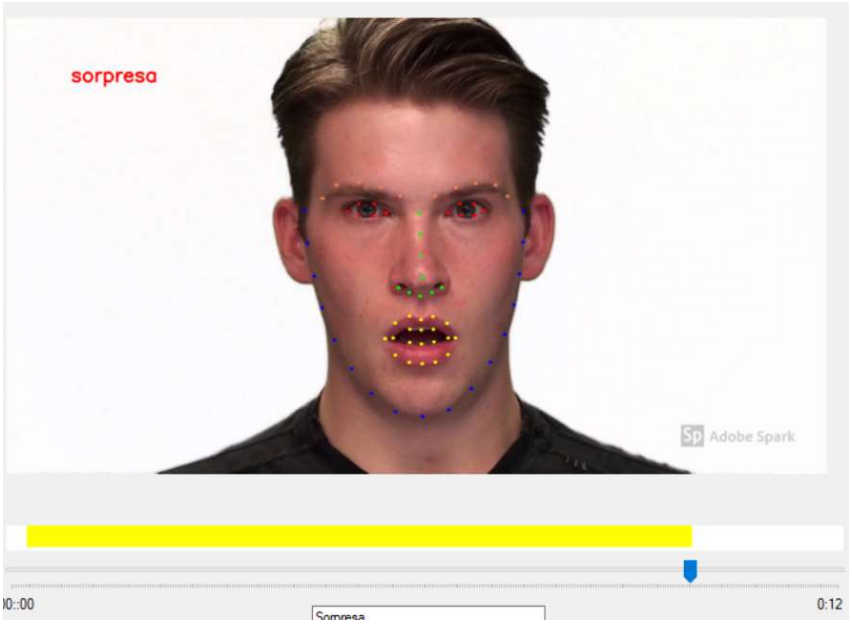

Figura 101 Captura de *sorpresa*

5.3.3 Actor 09

5.3.3.1 Vídeo 01-01-05-01-02-02-09.mp4

Vídeo de la emoción enfado (ira) para el actor 09.

El actor expresa ira mostrando los dientes y no apretando la boca.

El resultado es sorpresa en todo momento. Podemos observar la salida del programa en la Figura 102.

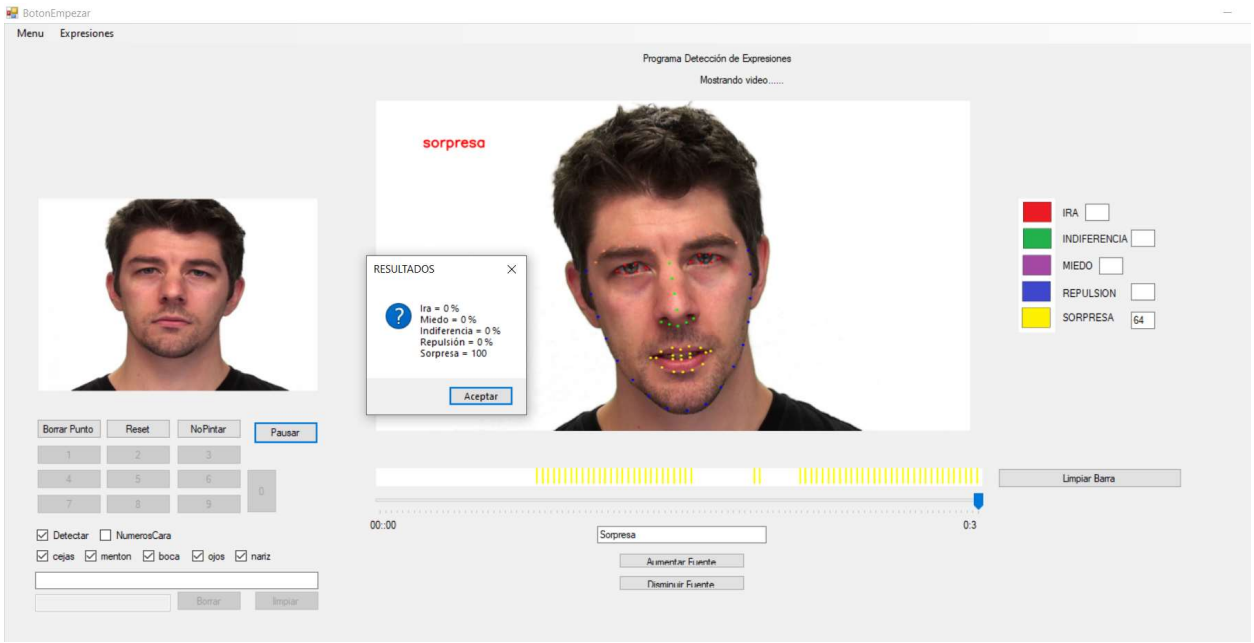

Figura 102 Resultado vídeo *enfado* actor 09

5.3.3.2 Vídeo 02-01-08-01-01-02-09.mp4

Vídeo de la emoción sorpresa para el actor 09. El resultado es sorpresa en todo momento.

En la Figura 103 podemos ver el resultado final.

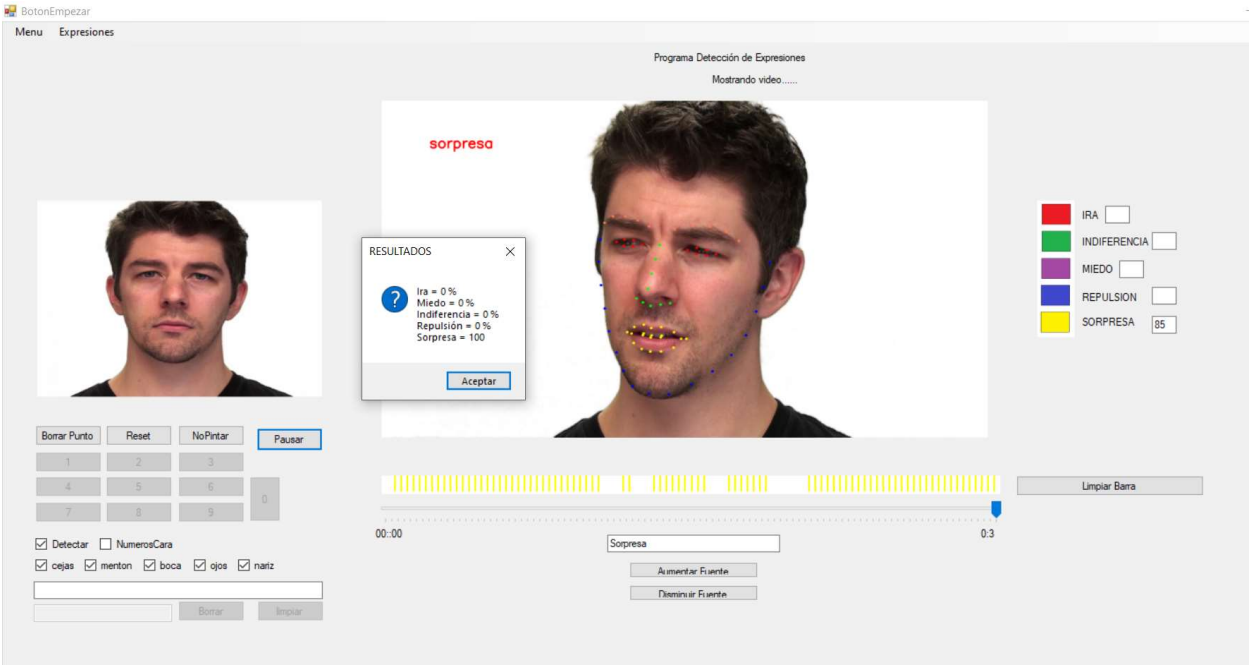

Figura 103 Resultado vídeo *sorpresa* actor 09

5.3.3.3 Vídeo 01-01-06-02-02-01-09.mp4

Vídeo de la emoción miedo para el actor 09. Aunque el actor levanta las cejas, no parece ser suficiente para que detecte miedo. El resultado es sorpresa en todo momento y lo podemos ver en la Figura siguiente.

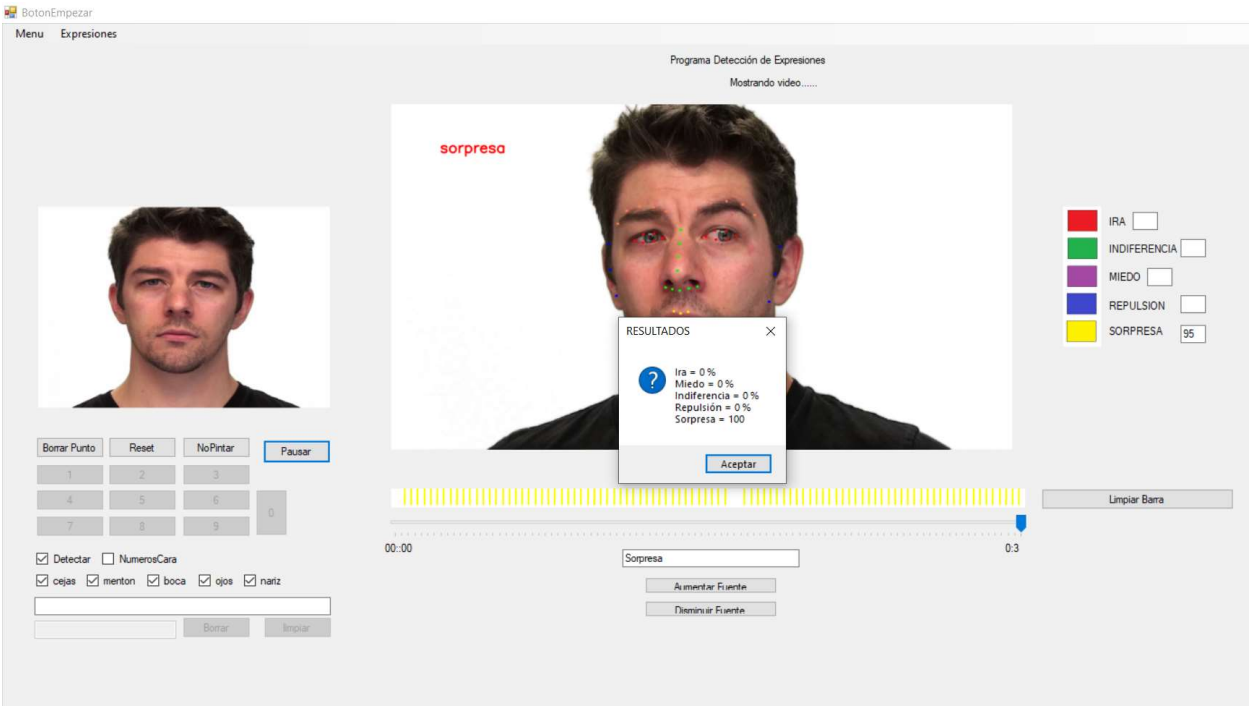

Figura 104 Resultado vídeo *miedo* actor 09

5.3.4 Actriz 02

5.3.4.1 Vídeo 01-01-05-01-02-02-02.mp4

Vídeo de ira para la actriz 02. Muestra ira frunciendo las cejas, pero no aprieta los labios.

El resultado es sorpresa cuando la actriz habla y abre la boca.

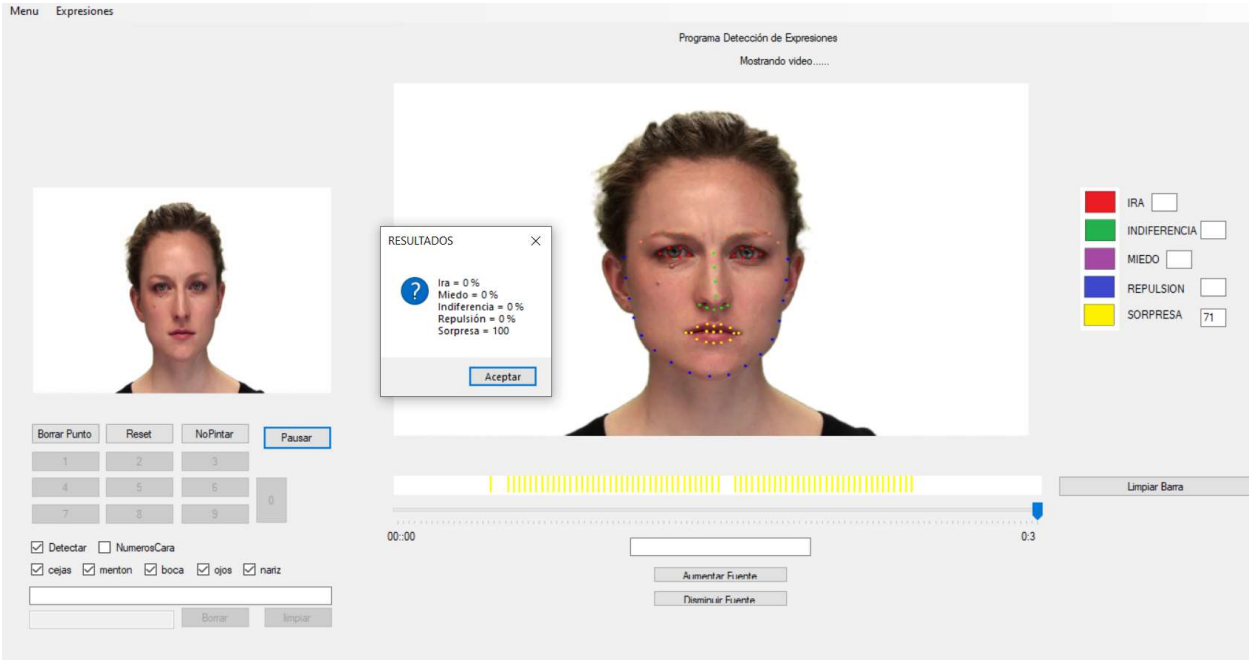

Figura 105 Resultado de *ira* para la actriz 02

5.3.4.2 Vídeo 02-01-08-01-01-02-02.mp4

Vídeo de la emoción sorpresa para la actriz 02. La actriz se sorprende levantando las cejas, por lo que el programa detecta esos *frames* como *miedo*.

En la Figura 106 podemos ver los resultados.

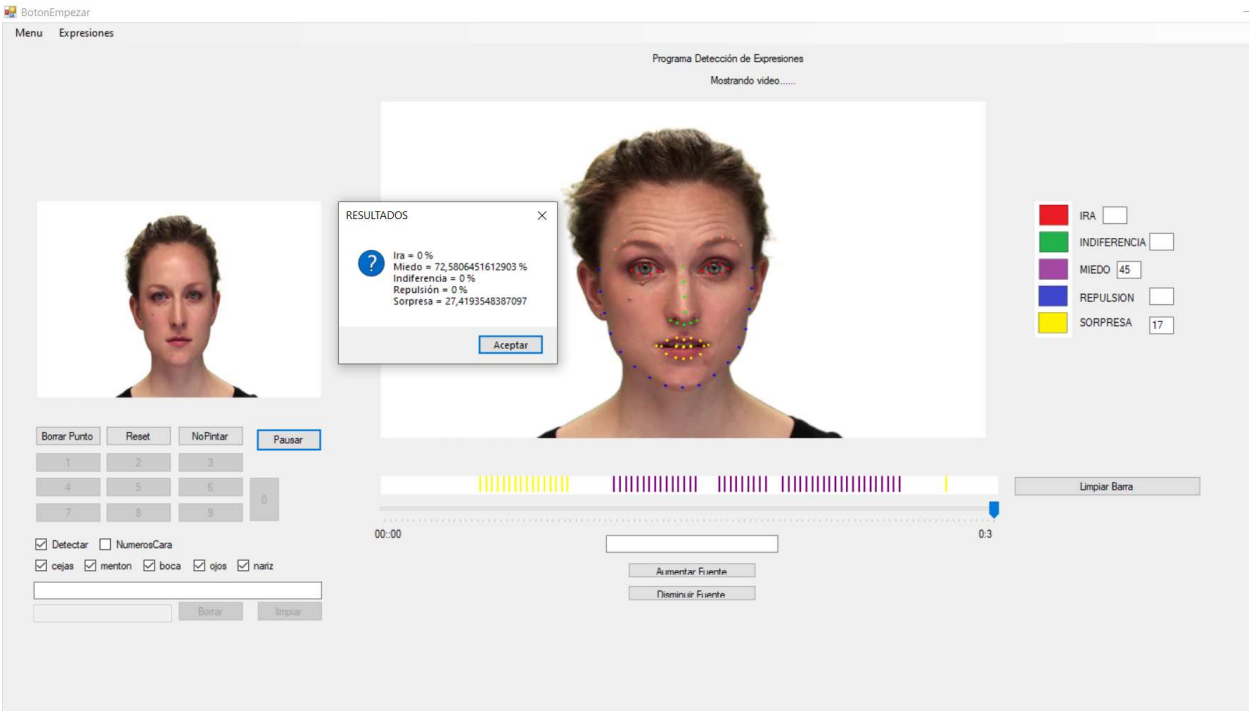

Figura 106 Resultado vídeo *sorpresa* para la actriz 02

5.3.4.3 Vídeo 01-01-06-02-02-01-02.mp4

Vídeo de miedo para la actriz 02. La actriz refleja miedo mostrando los dientes y frunciendo las cejas. Saca sorpresa en todo momento, ya que mantiene la boca abierta.

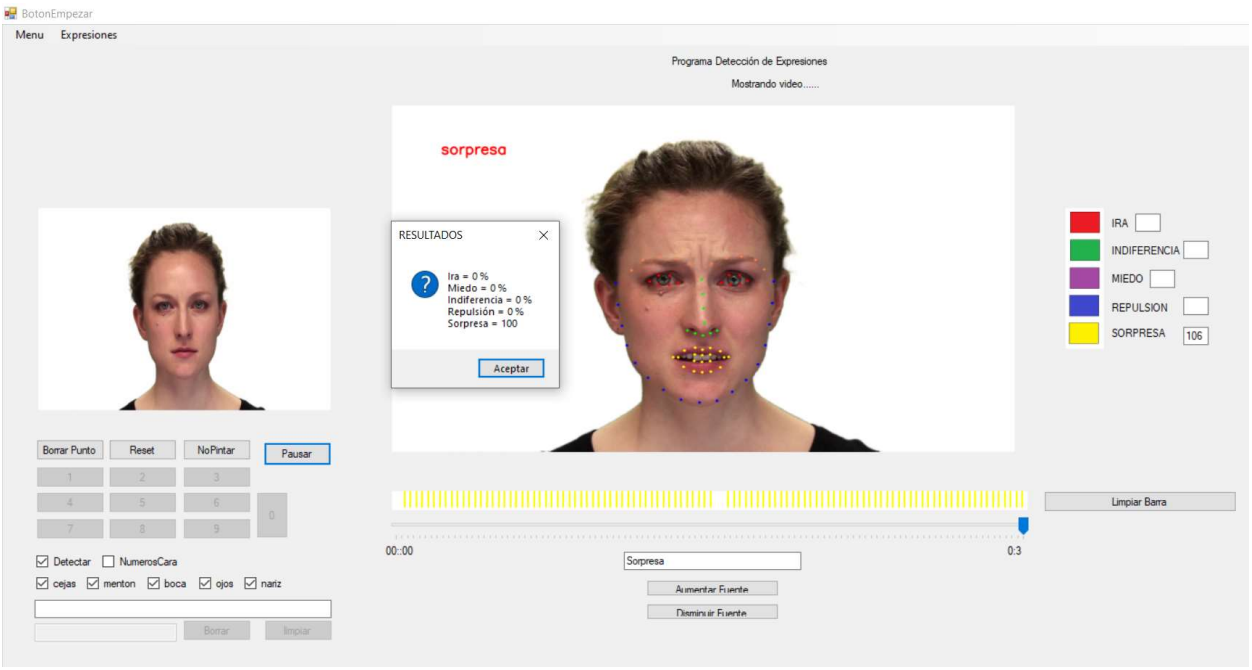

Figura 107 Resultado vídeo *miedo* para la actriz 02

5.4 Experimento fijando imagen de referencia

Para los vídeos de referencia vamos a utilizar una misma imagen como modelo para posteriormente analizar si varían mucho los resultados. La imagen modelo a utilizar va a ser la de Fernando Alonso usada en el vídeo de indiferencia.

Figura 108 Imagen usada como referencia fija en el experimento

5.4.1 Análisis de *Miedo*

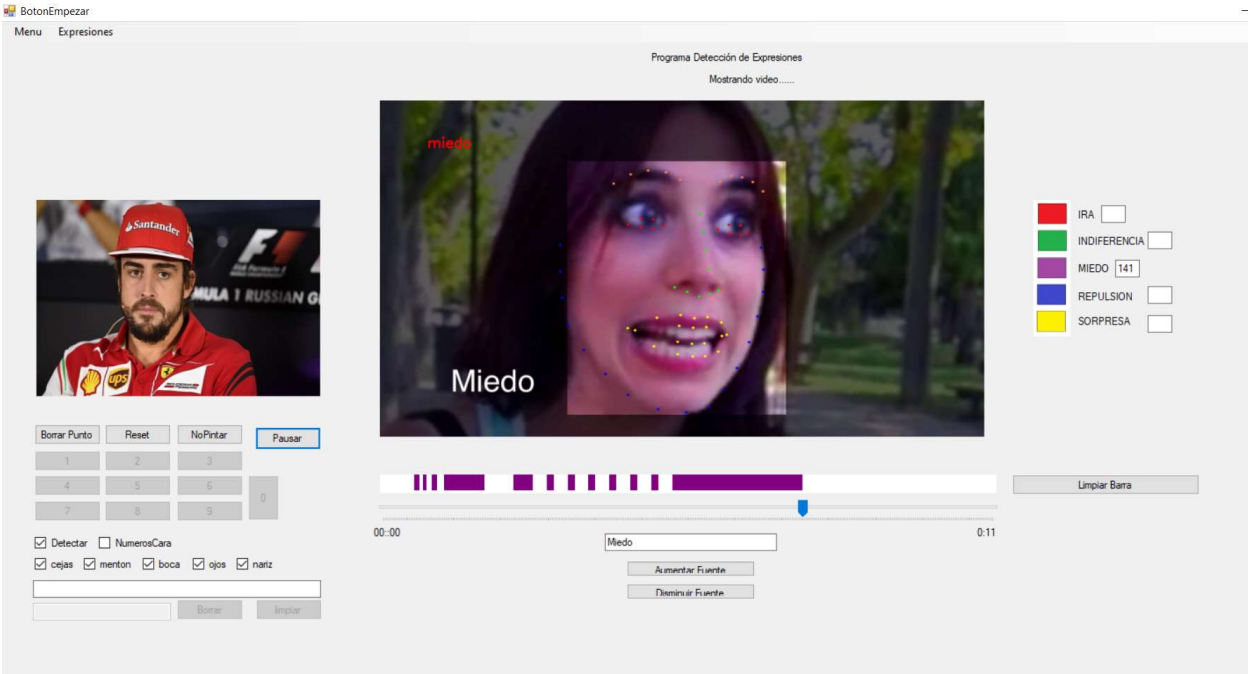

Figura 109 Análisis de *miedo* con imagen fija

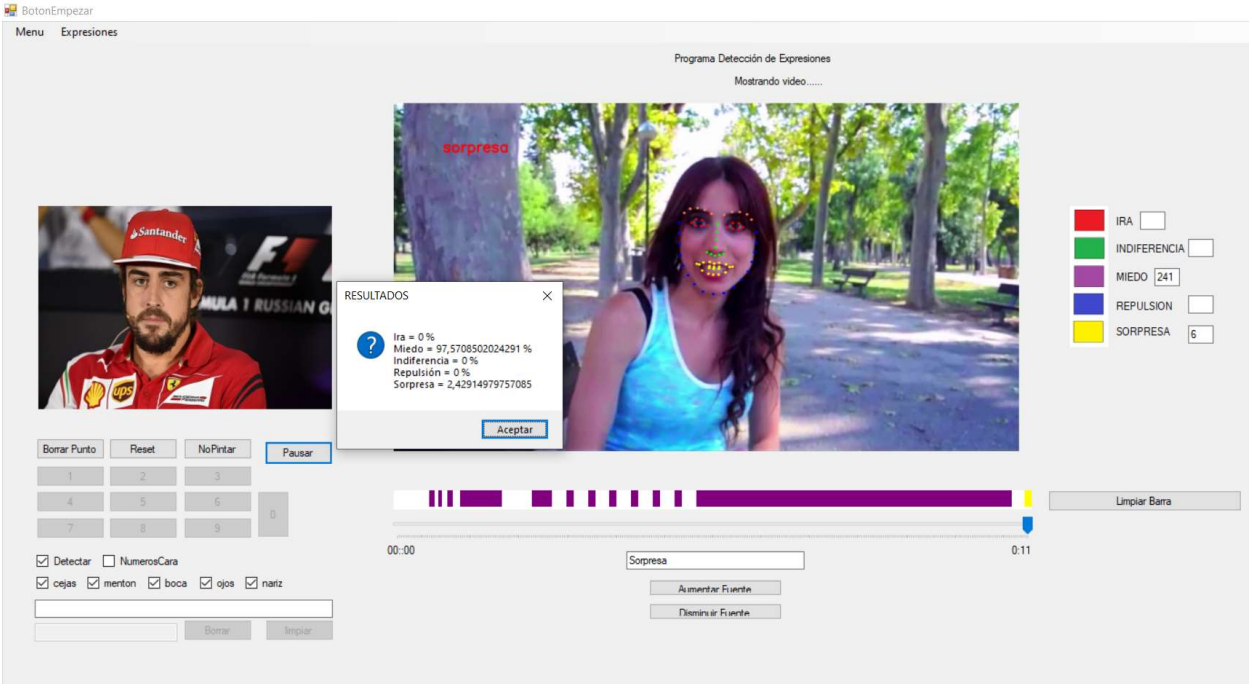

Figura 110 Resultado de *miedo* con imagen fija

5.4.2 Análisis de *Ira*

5.4.2.1 Vídeo 1

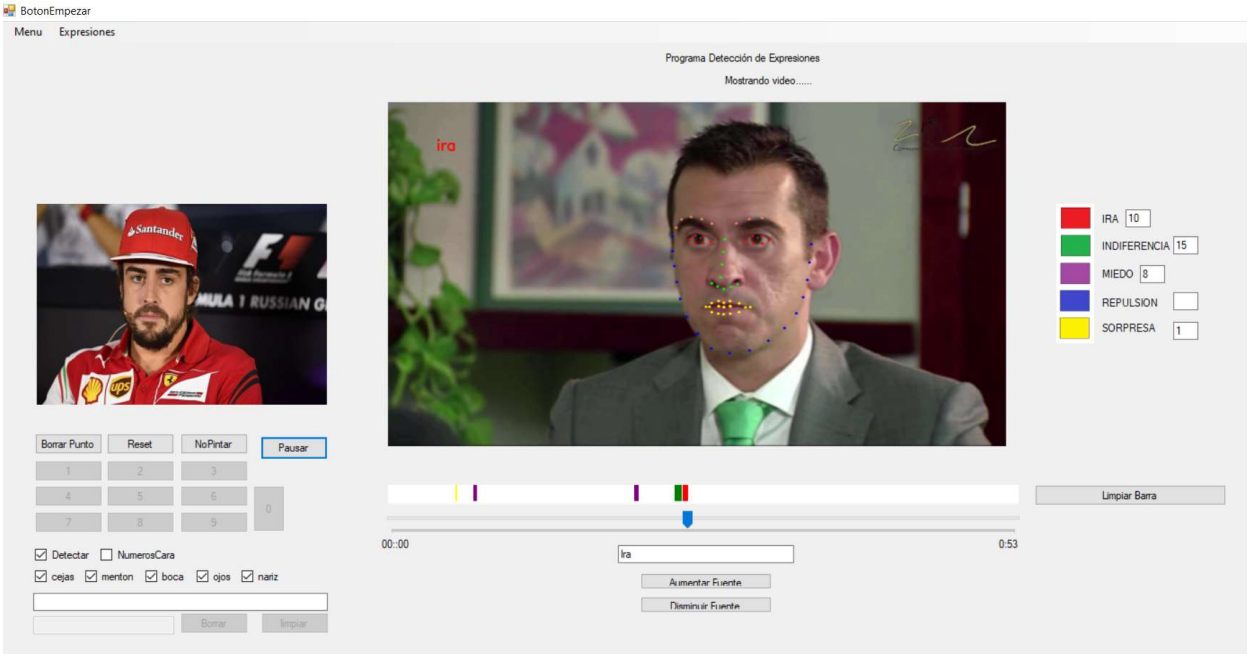

Figura 111 Momento en el que se detecta *ira* vídeo 1 con imagen fija

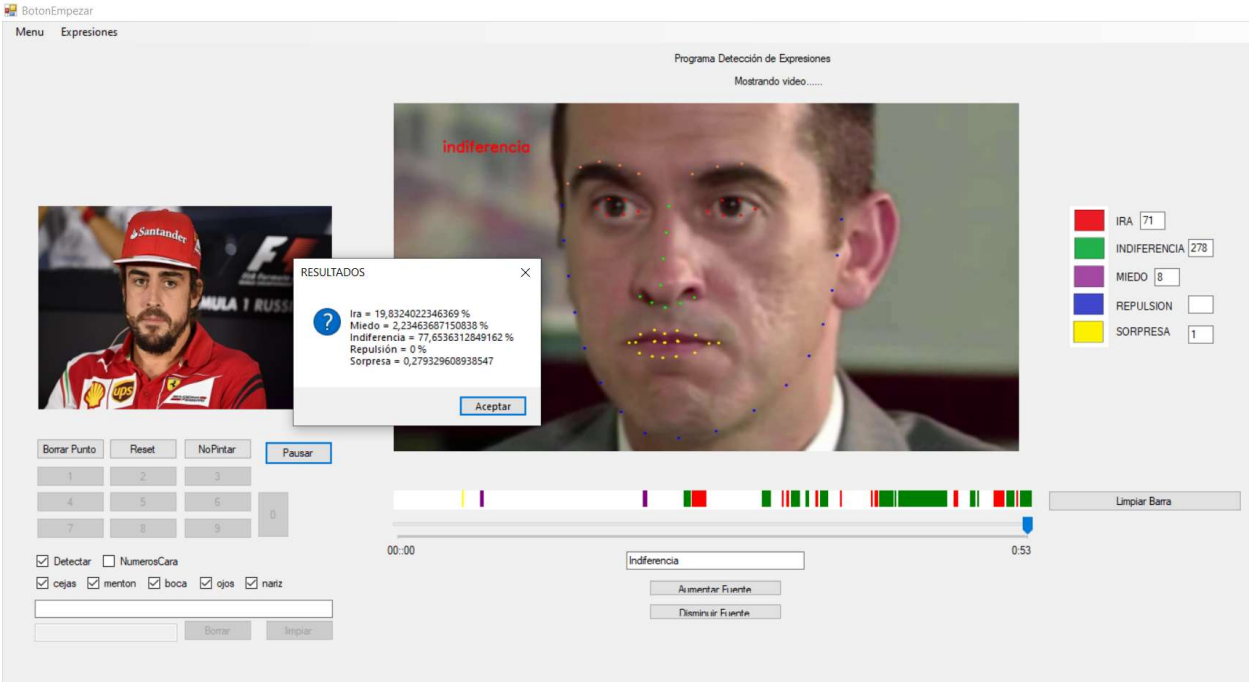

Figura 112 Resultado de *ira* vídeo 1 con imagen fija

5.4.2.2 Vídeo 2

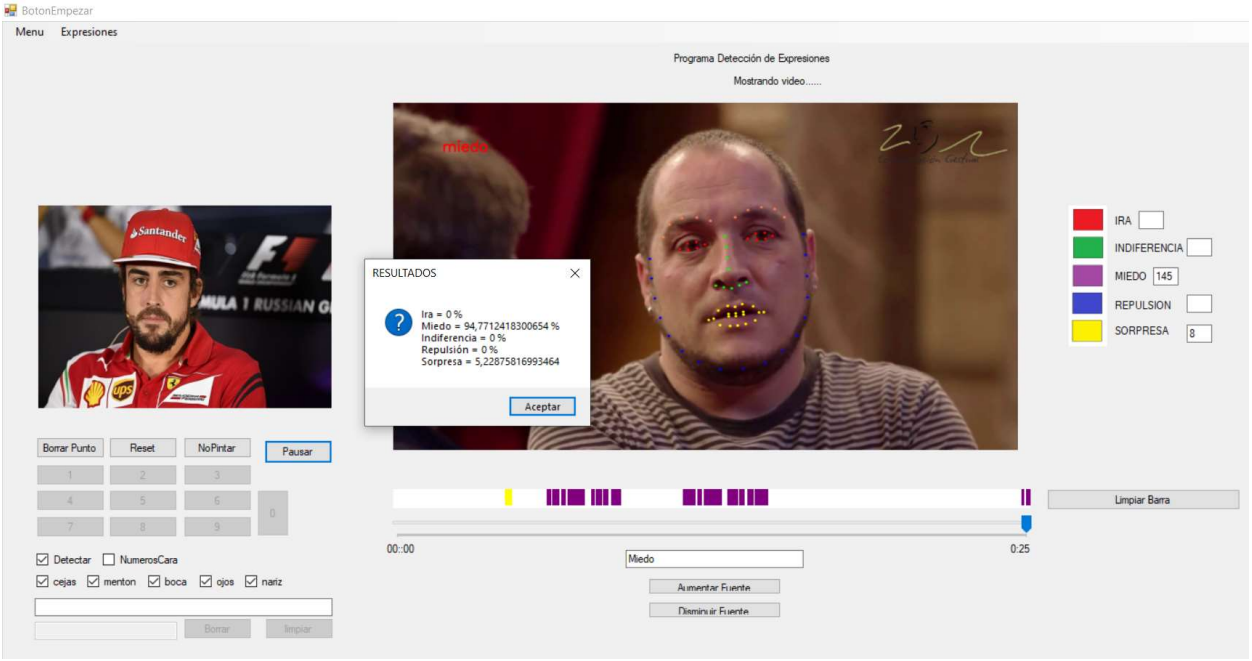

Figura 113 Resultado de *ira* vídeo 2 con imagen fija

5.4.3 Análisis de *Indiferencia*

Este vídeo no va a ser analizado ya que se usa la imagen de referencia que le corresponde.

Los resultados son exactamente los mismos.

5.4.4 Análisis de *Sorpresa*

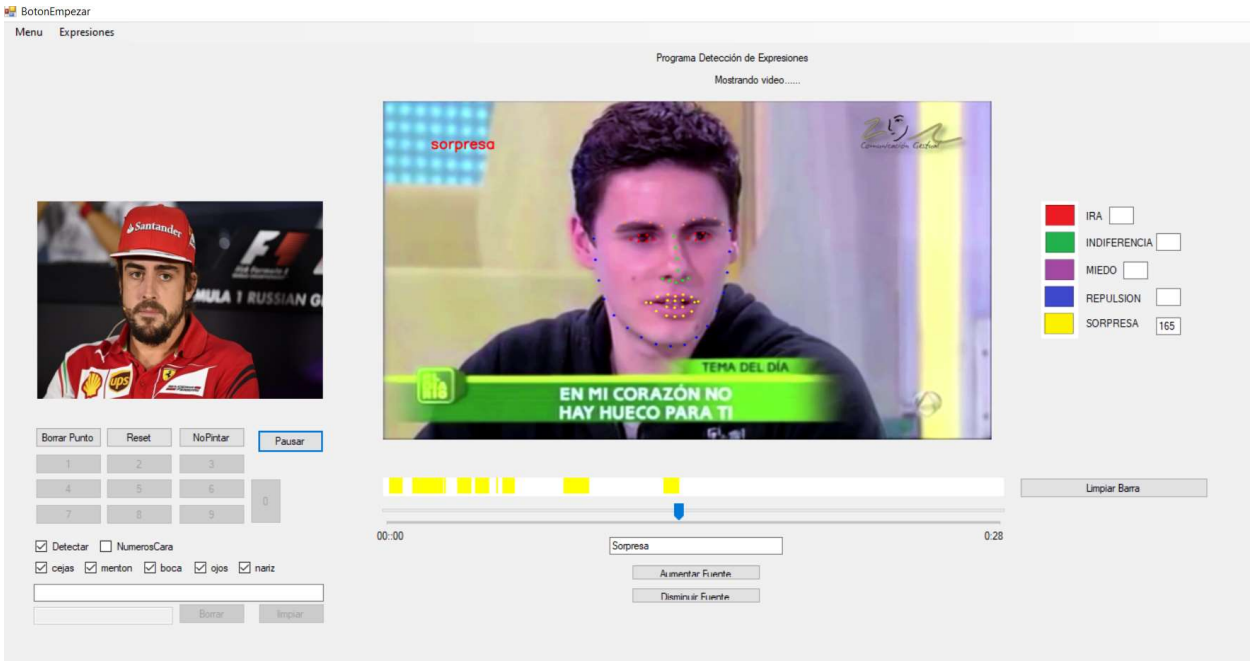

Figura 114 Momento en el que se detecta *sorpresa* con imagen fija

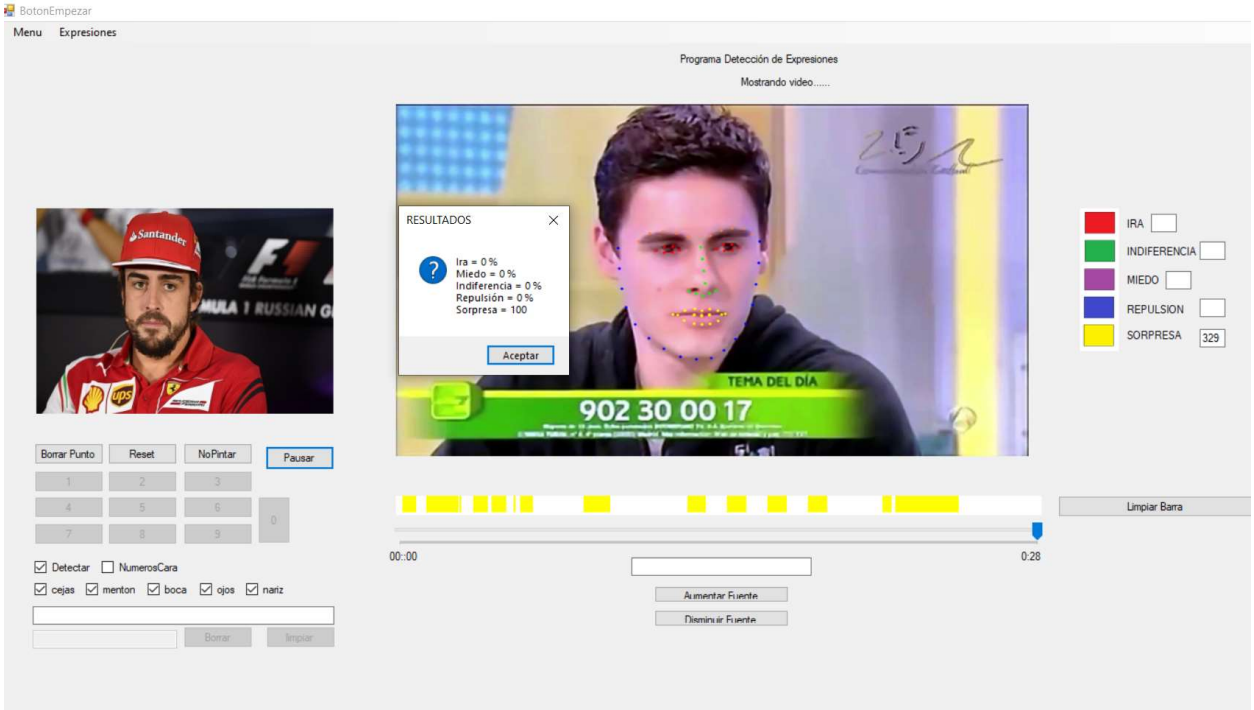

Figura 115 Resultado *sorpresa* con imagen fija

5.4.5 Análisis de Repulsión

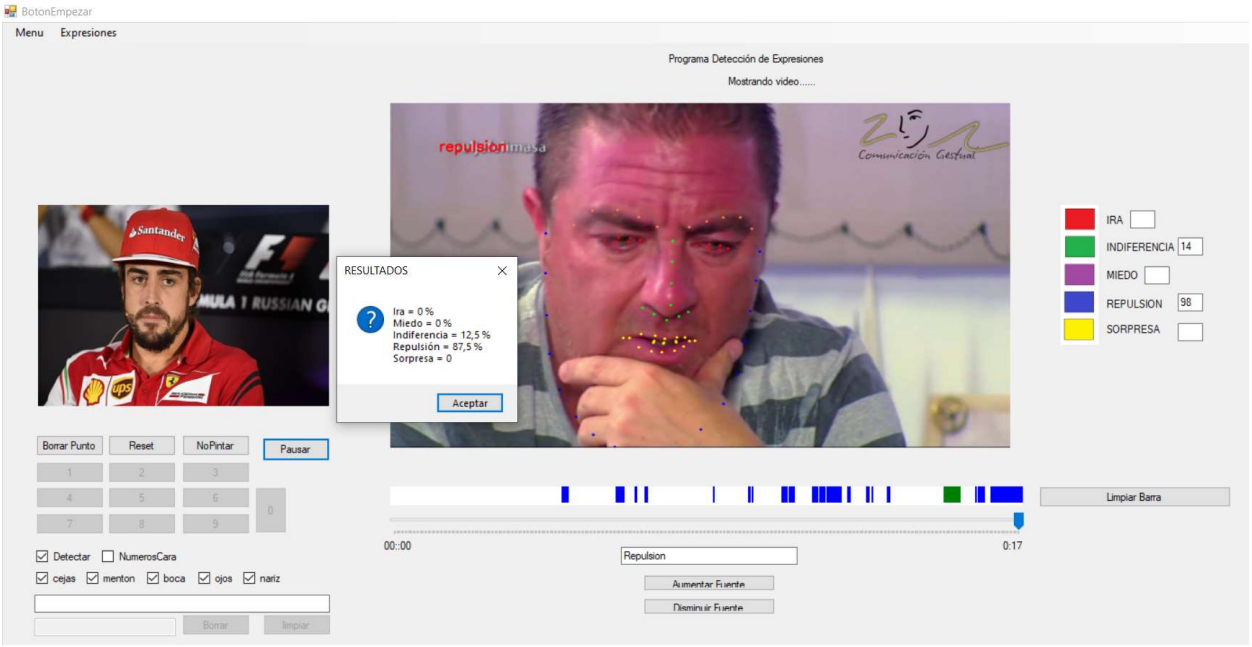

Figura 116 Resultado *repulsión* con imagen fija

5.5 Análisis de resultados

En la Figura 117 podemos ver una tabla resumiendo los resultados obtenidos de los vídeos de referencia.

Estos vídeos son los que usamos como verdad de referencia, por lo tanto, la herramienta está optimizada para ellos. El funcionamiento es bueno. Cuando se detecta una emoción distinta a la que se debería detectar en el vídeo, está justificada.

Resultados	Ira	Indiferencia	Repulsión	Miedo	Sorpresa
Vídeo Miedo	0%	0%	0%	100%	0%
Vídeo Ira 1	92,4%	6,1%	0%	1,5%	0%
Vídeo Ira 2	48,5%	0%	0%	0%	51,5%
Vídeo Indiferencia	0%	100%	0%	0%	0%
Vídeo Sorpresa	0%	7,8%	0%	0%	92,2%
Vídeo Repulsión	2,3%	24,1%	73,6%	0%	0%

Figura 117 Tabla resumen de resultados de los vídeos de referencia

Para la base de datos los resultados numéricos son malos, ya que para un vídeo de enfado se detecta solo un 19% de ira y un 80% de sorpresa. Aunque como hemos visto, si analizamos individualmente cada fotograma tiene sentido que se detecte esa emoción en ese instante.

En la Figura 118 podemos ver los resultados de los vídeos pasados de la base de dato.

Base de datos	Ira	Indiferencia	Repulsión	Miedo	Sorpresa
Vídeo ira actor 05	19,56%	0%	0%	0%	80,44%
Vídeo sorpresa actor 05	0%	0%	0%	10,1%	89,9%
Vídeo ira actor 07	27,27%	0%	0%	0%	72,73%
Vídeo sorpresa actor 07	0%	0%	0%	100%	0%
Vídeo miedo actor 07	0%	0%	0%	100%	0%
Vídeo ira actor 09	0%	0%	0%	0%	100%
Vídeo sorpresa actor 09	0%	0%	0%	0%	100%
Vídeo miedo actor 09	0%	0%	0%	0%	100%
Vídeo ira actor 02	0%	0%	0%	0%	100%
Vídeo sorpresa actor 02	0%	0%	0%	72,6%	27,4%
Vídeo miedo actor 02	0%	0%	0%	0%	100%

Figura 118 Resultados numéricos de los vídeos de la base de datos

En la Figura119 se muestran los resultados de los vídeos de referencia usando una imagen fija.

Resultados	Ira	Indiferencia	Repulsión	Miedo	Sorpresa
Vídeo Miedo	0%	0%	0%	97,57%	2,43%
Vídeo Ira 1	19,83%	77,65%	0%	2,23%	0,29%
Vídeo Ira 2	0%	0%	0%	94,77%	5,23%
Vídeo Indiferencia	0%	100%	0%	0%	0%
Vídeo Sorpresa	0%	0%	0%	0%	100%
Vídeo Repulsión	0%	12,5%	87,5%	0%	0%

Figura 119 Tabla resumen de resultados de los vídeos de referencia con imagen fija

Los resultados varían mucho en los vídeos de la emoción ira. En el resto de vídeos no hay una diferencia muy grande. Lo podemos ver mejor en las figuras adjuntas a continuación.

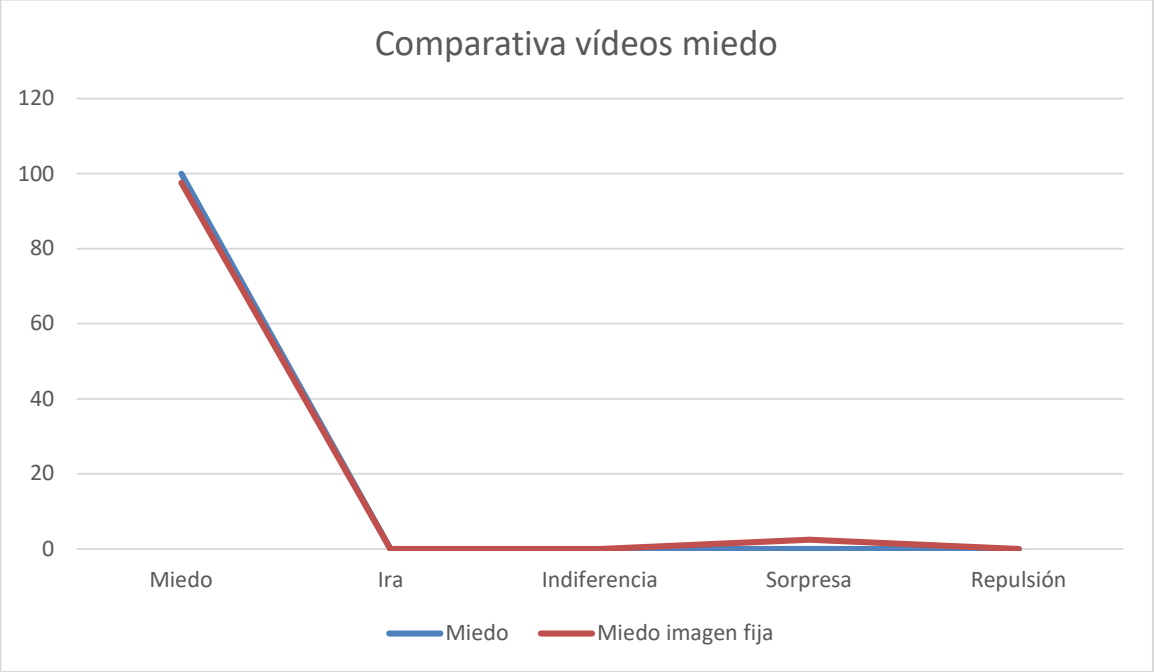

Figura 120 Comparativa vídeos *miedo*

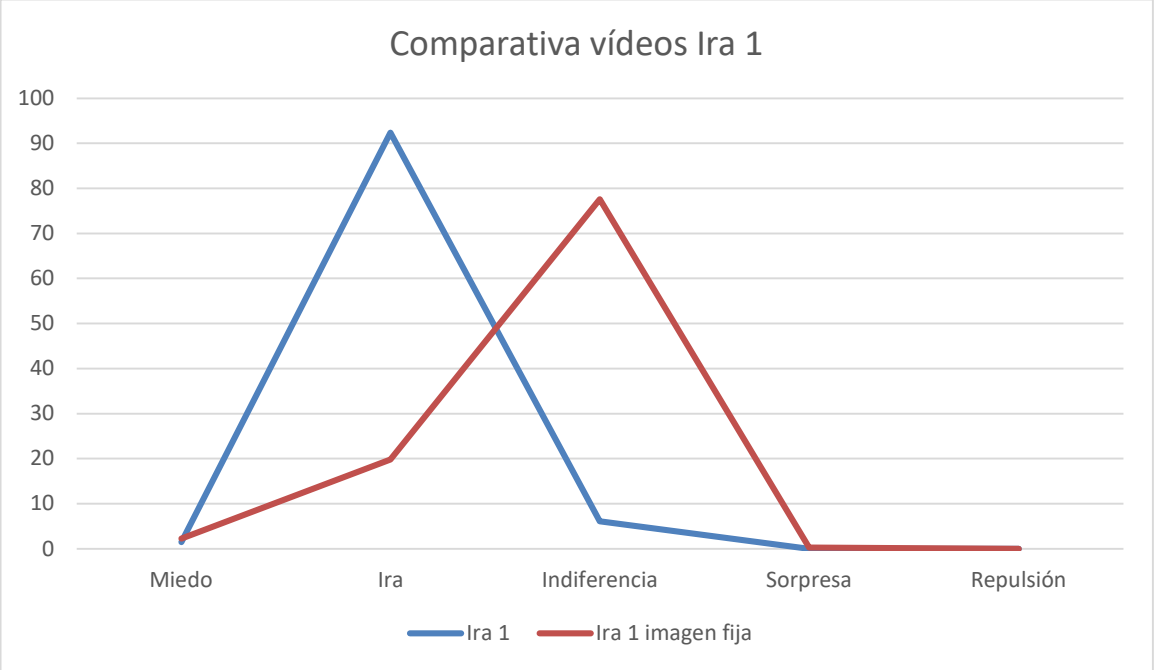

Figura 121 Comparativa vídeos *ira* 1

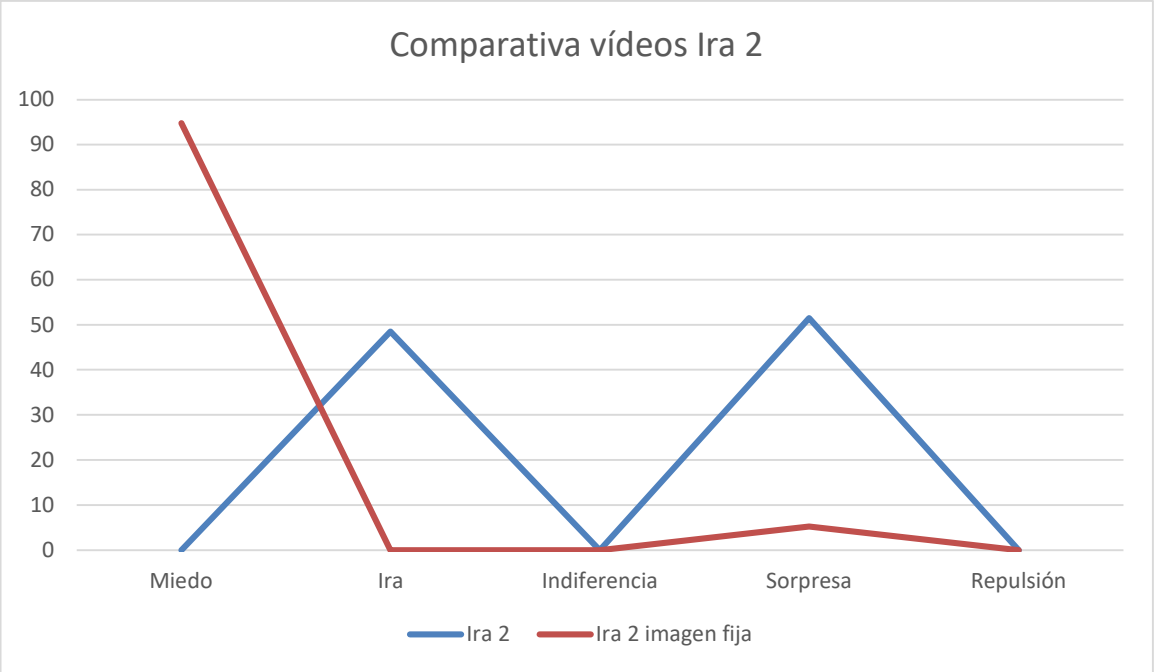

Figura 122 Comparativa vídeos *ira* 2

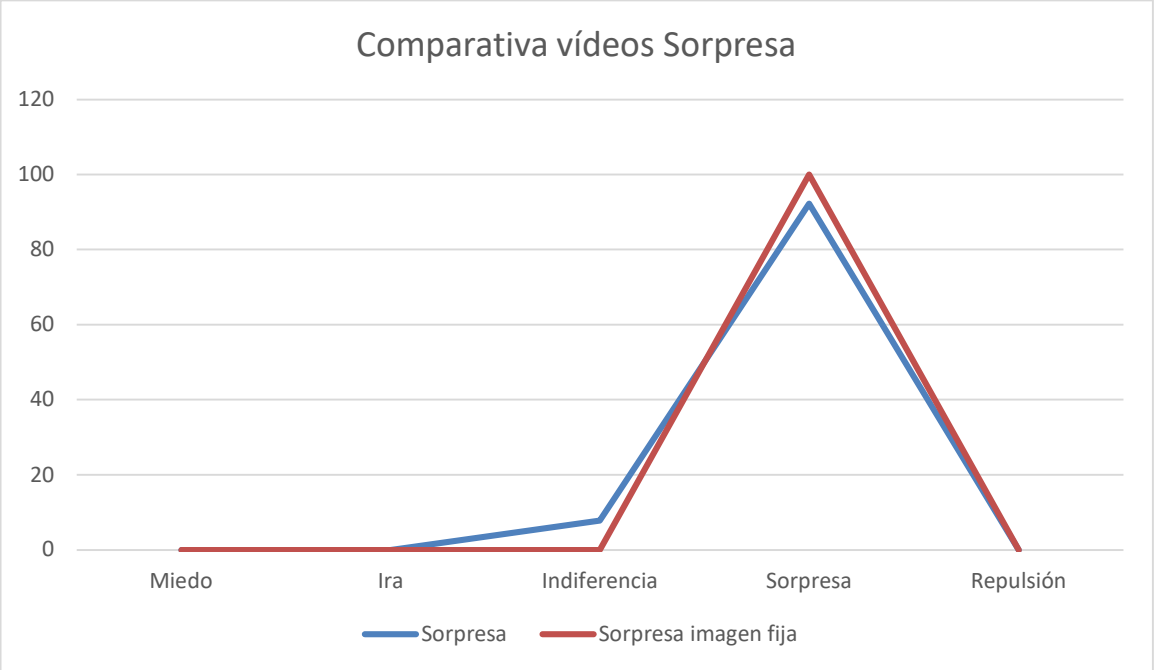

Figura 123 Comparativa vídeos *sorpresa*

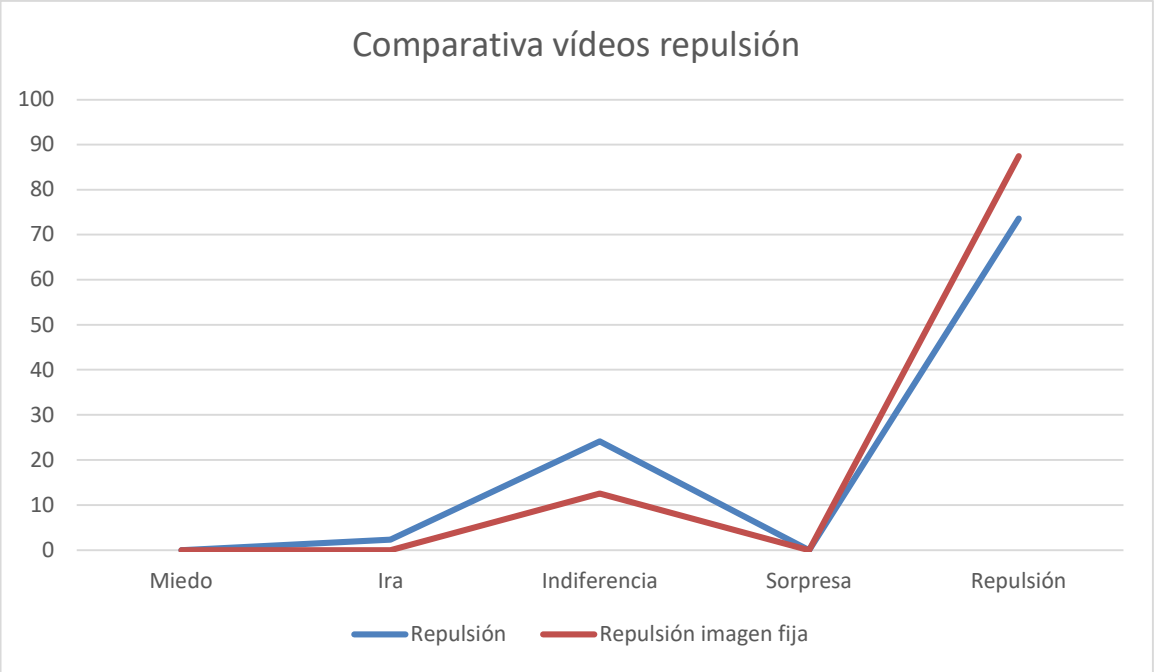

Figura 124 Comparativa vídeos *repulsión*

6 CONCLUSIONES Y FUTURAS LÍNEAS DE TRABAJO

6.1 Conclusiones

El resultado de la herramienta cambia según que vídeo le pasemos, funcionando mejor con los vídeos para los que está optimizada.

En los vídeos de la base de datos, que hemos descargado, el resultado depende de cómo expresa la emoción cada actor.

Como hemos contado en otros capítulos cada persona expresa sus emociones de una forma distinta, siendo algunos más expresivos que otros.

En mi opinión, es muy complicado desarrollar una herramienta que funcione igual de bien con todas las personas que le pasemos.

Esta herramienta da muchas facilidades para el estudio de emociones y poder modificar parámetros de manera dinámica.

6.2 Futuras líneas de trabajo

Para mejorar la herramienta se proponen los siguientes puntos:

- Añadir nuevas restricciones: Para las emociones con las que trabajamos, podemos añadir nuevas restricciones para evitar que, por ejemplo, se detecte *sorpresa* simplemente abriendo la boca.
- Añadir o eliminar emociones: Se puede intentar añadir nuevas emociones al software, incluso eliminando otras que no nos interesen. Por ejemplo, añadir *tristeza* y eliminar *repulsión*.
- Utilizar otros vídeos como verdad de referencia.
- Utilizar otras funciones de *OpenCV*: Podemos utilizar alguna función de *OpenCV* que nos ayude a detectar cuando se muestran los dientes en una expresión. Puede ser interesante para emociones como ira o sorpresa. Por lo general, cuando una persona se sorprende no suele mostrar los dientes y cuando se enfada sí.
- Darle un giro al programa y usarlo como herramienta social: Hoy en día estamos rodeados de redes sociales que usan filtros para añadirnos máscaras, maquillajes etc. Podríamos crear una herramienta que nos ponga una máscara en función de la emoción que reflejemos.
- Eliminar la elección de la imagen de referencia a mano, dándole al programa una imagen estándar de referencia. Esto hace que tengamos que realizar una captura del actor a mano cuando nosotros creamos que tiene una postura neutral. Como hemos visto en el experimento realizado varía más con la emoción

de ira, pero eligiendo una imagen adecuada y modificando los factores usados para detectar esta emoción, se podrían lograr resultados mejores.

- Optimizar el software basándonos en un sistema más complejo, como el sistema de codificación facial: Los movimientos individuales de los músculos faciales son codificadas por ' FACS ' desde leves cambios instantáneos en la apariencia facial.

  Ejemplo:

Emoción	Unidades de acción
Felicidad	6+12
Tristeza	1+4+15
Sorpresa	1+2+5B+26
Miedo	1+2+4+5+7+20+26
Coraje	4+5+7+23
Disgusto	9+15+16
Desprecio	R12A+R14A

  Donde Felicidad: 6 (Levantamiento de mejillas) + 12 (Tiramiento labial esquinal)

  Con este sistema podríamos añadir otras emociones y hacer un sistema de detección más serio.

6.3 Otros softwares similares disponibles en la actualidad

6.3.1 FACECODING

Es un software de reconocimiento facial online. Se basa en el sistema de codificación facial (FACS).

Se definen como la primera plataforma de Análisis de Micro Expresiones Faciales y Reconocimiento Emocional especializada en América Latina. Es capaz de discriminar entre Felicidad, Tristeza, Ira, Sorpresa, Miedo, Asco y Desprecio.

No es gratuito como podemos ver en la captura de su página web que copiamos a continuación.

Figura 125 Planes de pago de FACECODING

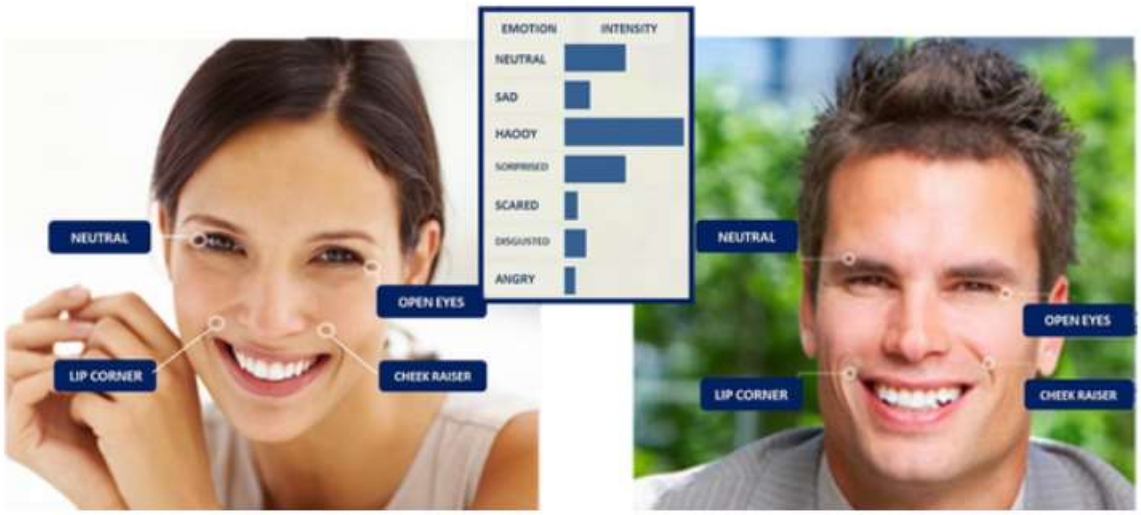

Figura 126 Ejemplo funcionamiento FACECODING

6.3.2 *EmoPLAY* (Orange)

EmoPLAY es una herramienta digital que permite entrenar el reconocimiento de varias emociones a través de las expresiones faciales.

Mediante una webcam o cámara integrada, *EmoPLAY* trabaja con distintas emociones como alegría, tristeza o enfado, de manera sencilla. Tras la elección de la emoción a trabajar, y una explicación tanto de la expresión facial asociada como de una situación en la que aplicarla, el usuario debe realizar los gestos determinan esa emoción, por ejemplo, sonreír para expresar alegría, y *EmoPLAY* analizará automáticamente su imagen indicando si la expresión es adecuada o no.

El programa ofrece configuraciones personalizadas para, por ejemplo, ajustar la sensibilidad del algoritmo de reconocimiento (mayor o menor dificultad), asociar una imagen de contexto que ayude a trabajar la emoción, modificar el refuerzo dado al usuario o el tiempo disponible para conseguir expresar la emoción.

El proyecto, impulsado por Fundación Orange, está desarrollado por CTIC Centro Tecnológico en colaboración, como Comité Científico, del Grupo de Investigación sobre Discapacidad (GID) de la Universidad de Oviedo.

Se puede descargar la aplicación para Windows, Android o IOS.

Está pensado para trabajar con personas con autismo.

Figura 127 Funcionamiento *EMOplay*

6.3.3 EMOFOCUS

De la empresa que ya hemos hablado anteriormente *Emotion Research Lab*.

Esta empresa ha adaptado el reconocimiento facial para distintas aplicaciones como por ejemplo *Emofocus*.

Emofocus es la primera aplicación de reconocimiento facial de emociones que se adapta a la técnica del grupo de discusión a través de dos formas:

- A través de cámaras instaladas en la sala de grupos, se siguen en tiempo real las reacciones emocionales de los participantes a través de pantallas en la sala de observación.
- La sesión es grabada y se analizan los fragmentos más relevantes para el estudio.

Figura 128 EMOFOCUS

En su página web podemos encontrar más ejemplos de aplicaciones, de los softwares disponibles en la actualidad para mí es el que ofrece más posibilidades.

No es gratuito ni muestra cuánto cuesta el uso del software. Es necesario solicitar un presupuesto en su web.

7 ANEXO: CÓDIGO FUNCIONES CREADAS

7.1 Expresiones.h

Dentro del código del software libre añadimos:

```
if (current_file == videoName)//MGJ si el archivo es el vídeo lo ponemos en picturebox2
                                {
                                        System::Drawing::Graphics^ graphics = pictureBox2-
>CreateGraphics();
                                        System::Drawing::Bitmap^ b = gcnew
System::Drawing::Bitmap(captured_image.cols, captured_image.rows, captured_image.step,
System::Drawing::Imaging::PixelFormat::Format24bppRgb, IntPtr(captured_image.ptr()));
                                        System::Drawing::RectangleF rect(0, 0, pictureBox2-
>Width, pictureBox2->Height);
                                        graphics->DrawImage(b, rect);
                                }
                                if (LandmarkDetector::seProduceIndiferencia() == 1)//MGJ
comprobamos si se produce alguna micro expresión en Landmarkdetectorultils
                                {
                                        tb_expresion->Text = "Indiferencia";//pone en un
textbox la expresion en este caso indiferencia si ha detectado que se ha producido
                                        Brush^ brush = gcnew SolidBrush(Color::Green);
                                        System::Drawing::Graphics^ graphics = Pb_eventotb-
>CreateGraphics();
                                        graphics->FillRectangle(brush,
System::Drawing::Rectangle((trackBarVideo->Value) * 702 / numeroFramesVideo,
0,2,18));//MGJ representa un en el picturebox de arriba del trackbar
                                        LandmarkDetector::ponertodoacero();//MGJ pone todas las
expresiones a cero para evitar el problema de que se mantenga una expresión a 1 y nos
pinte en un sitio que no queramos


                                }
                                else if (LandmarkDetector::seProduceIra() == 1) //MGJ igual
pero para ira, repulsión, miedo...
                                {
                                        tb_expresion->Text = "Ira";
                                        Brush^ brush = gcnew SolidBrush(Color::Red);
                                        System::Drawing::Graphics^ graphics = Pb_eventotb-
>CreateGraphics();
                                        graphics->FillRectangle(brush,
System::Drawing::Rectangle((trackBarVideo->Value) * 702 / numeroFramesVideo, 0, 2, 18));
                                        LandmarkDetector::ponertodoacero();
                                }
                                else if (LandmarkDetector::seProduceRepulsion() == 1)
                                {
                                        tb_expresion->Text = "Repulsion";
                                        Brush^ brush = gcnew SolidBrush(Color::Blue);
                                        System::Drawing::Graphics^ graphics = Pb_eventotb-
>CreateGraphics();
```

```
                                graphics->FillRectangle(brush,
System::Drawing::Rectangle((trackBarVideo->Value) * 702 / numeroFramesVideo, 0, 2, 18));
                                LandmarkDetector::ponertodoacero();
                        }
                        else if (LandmarkDetector::seProduceMiedo() == 1)
                        {
                                tb_expresion->Text = "Miedo";
                                Brush^ brush = gcnew SolidBrush(Color::Purple);
                                System::Drawing::Graphics^ graphics = Pb_eventotb-
>CreateGraphics();
                                graphics->FillRectangle(brush,
System::Drawing::Rectangle((trackBarVideo->Value) * 702 / numeroFramesVideo, 0, 2, 18));
                                LandmarkDetector::ponertodoacero();
                        }
                        else if (LandmarkDetector::seProduceSorpresa() == 1)
                        {
                                tb_expresion->Text = "Sorpresa";
                                Brush^ brush = gcnew SolidBrush(Color::Yellow);
                                System::Drawing::Graphics^ graphics = Pb_eventotb-
>CreateGraphics();
                                graphics->FillRectangle(brush,
System::Drawing::Rectangle((trackBarVideo->Value) * 702 / numeroFramesVideo, 0, 2, 18));
                                LandmarkDetector::ponertodoacero();
                        }
                        else
                        {
                                tb_expresion->Text = "";
                        }
                }
```

7.1.1 Botón *Menu*

7.1.1.1 Botón “Seleccionar modelo”

```
private: System::Void seleccionarModeloToolStripMenuItem_Click(System::Object^  sender,
System::EventArgs^  e) {
        openFileDialog1->InitialDirectory = "D:\\Working MGJ\\Expresiones\\imagenes";//MGJ
buscamos la imagen en el directorio inicial
        openFileDialog1->Filter = "Image files                                          (*.jpg,
*.jpeg, *.jpe, *.jfif, *.png) | *.jpg; *.jpeg;                                          *.jpe;
*.jfif; *.png";//MGJ filtra y busca solo imagenes                                       de esos
formatos
        if (openFileDialog1->ShowDialog() == System::Windows::Forms::DialogResult::OK)
        {
                System::IO::StreamReader ^ sr = gcnew
                        System::IO::StreamReader(openFileDialog1->FileName);
                msclr::interop::marshal_context context;
                 direccion = context.marshal_as<std::string>(openFileDialog1-
>FileName);//direccion tiene almacenada la ruta de la imagen

                cv::Mat img = cv::imread(direccion, 1);
                LandmarkDetector::Ponern2a0();



                if (!img.empty())
                {
                        //Hacemos la conversión de opencv a System::Image
```

```
                    cv::Mat newImage;
                    cvtColor(img, newImage, CV_BGR2BGRA);
                    System::Drawing::Graphics^ graphics = pictureBox1->CreateGraphics();
                    System::Drawing::Bitmap^ b = gcnew
System::Drawing::Bitmap(newImage.cols, newImage.rows, newImage.step1(),
System::Drawing::Imaging::PixelFormat::Format32bppArgb, IntPtr(newImage.ptr()));
                    System::Drawing::RectangleF rect(0, 0, pictureBox1->Width,
pictureBox1->Height);
                    graphics->DrawImage(b, rect);

             }


             sr->Close();
       }
       seleccionarVideoToolStripMenuItem->Enabled = true;



}
```

7.1.1.2 Botón “Seleccionar Video”

```
private: System::Void seleccionarVideoToolStripMenuItem_Click(System::Object^  sender,
System::EventArgs^  e) {
       BotonEmpezar->Text = "Pausar";
       backgroundWorker1->CancelAsync();
       openFileDialog1->InitialDirectory = "D:\\Working MGJ\\Expresiones";//MGJ igual que
la anterior pero ahora para un video
       openFileDialog1->Filter = "Video files (*.mp4) | *.mp4";
       if (openFileDialog1->ShowDialog() == System::Windows::Forms::DialogResult::OK)
       {
             msclr::interop::marshal_context context;
             videoName = context.marshal_as<std::string>(openFileDialog1->FileName);
             int segundos;
             int minutos;

             cap = new cv::VideoCapture(videoName);

             cv::Mat imgOpenCV;
             if (cap->isOpened())
             {

                    numeroFramesVideo = cap->get(CV_CAP_PROP_FRAME_COUNT);//MGJ calcula
el numero de frames los frames por segundo...
                    fps = cap->get(CV_CAP_PROP_FPS);
       //tiene uso para saber la duración del video, cuando el video ha finalizado...
                    segundos = numeroFramesVideo / fps;
                    minutos = segundos / 60;
                    segundos = (segundos % 60) ;
                    duracion->Text = minutos.ToString()+":"+segundos.ToString();
                   //irse a una posición determinada
                   //cap->set(CV_CAP_PROP_POS_FRAMES, 100);
                   trackBarVideo->Minimum =  0;
                   trackBarVideo->Maximum = numeroFramesVideo - 1;


                   cap->read(imgOpenCV);
                   if (!imgOpenCV.empty())
                   {
                          //Hacemos la conversión de opencv a System::Image
```

```
                        System::Drawing::Graphics^ graphics = pictureBox2-
>CreateGraphics();
                        System::Drawing::Bitmap^ b = gcnew
System::Drawing::Bitmap(imgOpenCV.cols, imgOpenCV.rows, imgOpenCV.step,
System::Drawing::Imaging::PixelFormat::Format24bppRgb, IntPtr(imgOpenCV.ptr()));
                        System::Drawing::RectangleF rect(0, 0, pictureBox2->Width,
pictureBox2->Height);
                        graphics->DrawImage(b, rect);

                    }
                }


        }


        BotonEmpezar->Text = "Empezar";
        BotonEmpezar->Enabled = true;



}
```

7.1.1.3 Botón "Cancelar"

```
private: System::Void cancelarToolStripMenuItem_Click(System::Object^  sender,
System::EventArgs^  e) {
        if (backgroundWorker1->IsBusy)
        {

                // Notify the worker thread that a cancel has been requested.

                // The cancel will not actually happen until the thread in the

                // DoWork checks the m_oWorker.CancellationPending flag.

                backgroundWorker1->CancelAsync();
                primerclick = true;
                trackBarVideo->Value = 0;
                contador = 0;
        }
}
```

7.1.1.4 Botón "Salir"

```
private: System::Void salirToolStripMenuItem1_Click(System::Object^  sender,
System::EventArgs^  e) {
        Application::Exit();
}
```

7.1.2 Botón *Empezar/Pausa/Continuar*

```
private: System::Void BotonEmpezar_Click(System::Object^  sender, System::EventArgs^  e)
{
        tb2->Text = "";
        if (BotonEmpezar->Text == "Continuar")
```

```
	{
		BotonEmpezar->Text = "Pausar";
		Bborrar->Enabled = true;
		cancelarToolStripMenuItem->Enabled = true;

	}
	else
	{
		BotonEmpezar->Text = "Continuar";
		Bborrar->Enabled = false;
		cancelarToolStripMenuItem->Enabled = false;
	}
	if (primerclick)//MGJ si es el primerclick borra la barra de colores de las
expresiones y inicializa si no lo único que hace es cambiar la etiqueta del botón que
sirve para que el programa se quede pausado o no
	{
		BotonEmpezar->Text = "Pausar";
		Bborrar->Enabled = true;
		System::Drawing::Graphics^ graphics = Pb_eventotb->CreateGraphics();
		graphics->Clear(System::Drawing::Color::White);
		trackBarVideo->Value = contador;
		cancelarToolStripMenuItem->Enabled = true;
		backgroundWorker1->RunWorkerAsync();
		primerclick = false;


	}
}
```

7.1.3 Botón *Limpiar Barra*

```
private: System::Void BotonLimpiarBarra_Click(System::Object^  sender, System::EventArgs^
e) {
	System::Drawing::Graphics^ graphics = Pb_eventotb->CreateGraphics();
	graphics->Clear(System::Drawing::Color::White);
}
```

7.1.4 Botón *Borrar punto*

```
private: System::Void Bborrar_Click(System::Object^  sender, System::EventArgs^  e) {
	Bborrar->Enabled = false;
	pause = true;
	bloquear = true;
	result = MessageBox::Show("Quiere borrar un punto?", "Borrar punto",
MessageBoxButtons::YesNo, MessageBoxIcon::Question);
	bloquear = false;
	tb->Enabled = true;
	Blimpiar->Enabled = true;
	Bborrarpuntos->Enabled = true;
	B1->Enabled = true;
	B2->Enabled = true;
	B3->Enabled = true;
	B4->Enabled = true;
	B5->Enabled = true;
	B6->Enabled = true;
```

```
        B7->Enabled = true;
        B8->Enabled = true;
        B9->Enabled = true;
        B0->Enabled = true;
        tb2->Text = "Ha saleccionado borrar un punto de la cara";
        tb->Text = "";
}
private: System::Void Blimpiar_Click(System::Object^  sender, System::EventArgs^  e) {
        tb->Text = "";

}
```

7.1.5 Botones 0-9

```
private: System::Void B0_Click(System::Object^  sender, System::EventArgs^  e) {
        tb->Text = tb->Text + 0;

                                                }
private: System::Void B1_Click(System::Object^  sender, System::EventArgs^  e) {
        tb->Text = tb->Text + 1;

}
private: System::Void B2_Click(System::Object^  sender, System::EventArgs^  e) {
        tb->Text = tb->Text + 2;

}
private: System::Void B3_Click(System::Object^  sender, System::EventArgs^  e) {
        tb->Text = tb->Text + 3;

}

private: System::Void B4_Click(System::Object^  sender, System::EventArgs^  e) {
        tb->Text = tb->Text + 4;

}

private: System::Void B5_Click(System::Object^  sender, System::EventArgs^  e) {
        tb->Text = tb->Text + 5;

}
private: System::Void B6_Click(System::Object^  sender, System::EventArgs^  e) {
        tb->Text = tb->Text + 6;

}
private: System::Void B7_Click(System::Object^  sender, System::EventArgs^  e) {
        tb->Text = tb->Text + 7;

}

private: System::Void B8_Click(System::Object^  sender, System::EventArgs^  e) {
        tb->Text = tb->Text + 8;

}
private: System::Void B9_Click(System::Object^  sender, System::EventArgs^  e) {
        tb->Text = tb->Text + 9;

}
```

7.1.6 Botón *Borrar*

```
private: System::Void Bborrarpuntos_Click(System::Object^  sender, System::EventArgs^  e)
{
	borrarpuntos = false;
}
```

7.1.7 Botón *Limpiar*

```
private: System::Void Blimpiar_Click(System::Object^  sender, System::EventArgs^  e) {
	tb->Text = "";
}
```

7.1.8 Botón *Reset*

```
private: System::Void button1_Click_1(System::Object^  sender, System::EventArgs^  e) {
	LandmarkDetector::resetear();//MGJ resetea las opciones
	checkboca->Checked=true;
	checkcejas->Checked = true;
	checknariz->Checked = true;
	checkmenton->Checked = true;
	checkojos->Checked = true;
	checknumeroscara->Checked = false;
}
```

7.1.9 Botón *No pintar*

```
private: System::Void Bnopinta_Click(System::Object^  sender, System::EventArgs^  e) {
	checkboca->Checked = false;//MGJ no representa nada en la cara
	checkcejas->Checked = false;
	checknariz->Checked = false;
	checkmenton->Checked = false;
	checkojos->Checked = false;

}
```

7.1.10 Botón *Aumentar Fuente* y *Disminuir Fuente*

```
private: System::Void botonaumentar_Click(System::Object^  sender, System::EventArgs^  e)
{
	LandmarkDetector::aumentarfuente();
}
private: System::Void botondisminuir_Click(System::Object^  sender, System::EventArgs^
e) {
	LandmarkDetector::disminuirfuente();
```

```
}
```

7.1.11 Comprobación de expresiones

```
private: System::Void checkBox1_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
	LandmarkDetector::detectar();//MGJ llamamos a la función detectar que lo que hace
es que el programa va a mostrar la expresión que se produzca o no haga nada
	if (checkdetectar->Checked)
	{
		tb2->Text = "Detectando";
	}
	else
	{
		tb2->Text = "Dejamos de detectar";
	}
}
private: System::Void checkboca_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
	LandmarkDetector::pintasector("b");
}
private: System::Void checknumeroscara_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
	LandmarkDetector::pintarnumeroscara();
}
private: System::Void checkcejas_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
	LandmarkDetector::pintasector("c");
}
private: System::Void checkmenton_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
	LandmarkDetector::pintasector("m");
}
private: System::Void checkojos_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
	LandmarkDetector::pintasector("o");
}
private: System::Void checknariz_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
	LandmarkDetector::pintasector("n");
}
```

7.2 LandmarkDetectorUtils.cpp

7.2.1 Función *pintasector*

```
void pintasector(std::string borra)
{
```

```
if (borra == "c")
{
```

```
                cejas = cejas*(-1);

            }
            else if (borra == "o")
            {
                ojos = ojos*(-1);

            }
            else if (borra == "m")
            {
                menton = menton*(-1);

            }
            else if (borra == "b")
            {
                boca = boca*(-1);

            }
            else if (borra == "n")
            {
                nariz = nariz*(-1);

            }
}
```

7.2.2 Función *borraunpunto*

```
void borraunpunto(int unico)
{
    borraunico[unico] = borraunico[unico] * (-1);
}
```

7.2.3 Función *resetear*

```
void resetear()
{

    borraunico.assign(68, 1);

}
```

7.2.4 Función *nopintarnada*

```
void nopintarnada()
{
    menton = 0;
    ojos = 0;
    cejas = 0;
    nariz = 0;
    boca = 0;
}
```

7.2.5 Función *detectar*

```
void detectar()
{
	empezar = empezar*(-1);
}
```

7.2.6 Función *pintarnumeroscara*

```
void pintarnumeroscara()
{
	numeroscara = numeroscara*(-1);
}
```

7.2.7 Función *distancia*

```
//función que calcula la distancia entre dos puntos de la cara
//MGJ
double distancia(vector<cv::Point2d> vector1, int punto1, int punto2)
{
	int x1, y1, x2, y2;
	double d;
	x1 = vector1[punto1].x;
	y1 = vector1.at(punto1).y;
	x2 = vector1.at(punto2).x;
	y2 = vector1.at(punto2).y;
	d = sqrt(pow((x1 - x2), 2) + pow((y1 - y2), 2));
	return d;

}
```

7.2.8 Función *ponern2a0*

```
void Ponern2a0()
{
	n2 = 0;
}
```

7.2.9 Función *ponertodoacero*

```
void ponertodoacero()
{
	produceindiferencia = 0;
	produceira = 0;
```

```
        producemiedo = 0;
        producerepulsion = 0;
        producesorpresa = 0;
}
```

7.2.10 Funciones que modifican el factor de las expresiones

```
void bocaAbiertaFuncion(double factor)
{
        factorbocaAbierta = factor;
}


void cejasBajadasFuncion(double factor)
{
        factorCejasBajadas = factor;
}

void labiosApretadosFuncion(double factor)
{
        factorLabiosApretados = factor;
}

void ojosCerradosFuncion(double factor)
{
        factorOjosCerrados = factor;
}

void ojosAbiertosFuncion(double factor)
{
        factorOjosAbiertos = factor;
}
```

7.2.11 Funciones que detectan si se ha producido una expresión

```
int seProduceMiedo()
{
        return producemiedo;
}
int seProduceIra()
{
        return produceira;
}
int seProduceSorpresa()
{
        return producesorpresa;
}
int seProduceRepulsion()
{
        return producerepulsion;
}
int seProduceIndiferencia()
{
        return produceindiferencia;
}
```

7.2.12 Funciones para aumentar y disminuir fuente

```
void aumentarfuente()
{
	fuente = fuente + 0.1;
}
void disminuirfuente()
{
	fuente = fuente - 0.1;
}
```

7.2.13 Función *Draw*

```
if (n2 == 0)//en la primera llamada  hago la copia de mi imagen de referencia
		{

			vectorpuntos.clear();//limpio el vector donde guardo los puntos de
referencia por si acaso


			for (int i = 0; i < n; ++i)//bucle que almacena en vectorpuntos los
puntos detectados
			{
				if (visibilities.at<int>(i))
				{
					cv::Point2d punto(cvRound(shape2D.at<double>(i) *
(double)draw_multiplier), cvRound(shape2D.at<double>(i + n) * (double)draw_multiplier));

					vectorpuntos.push_back(punto);
				}

			}
			d1boca = distancia(vectorpuntos, 62, 66);
			d1cejaojo = distancia(vectorpuntos, 21, 39);
			dojoizq1 = distancia(vectorpuntos, 43, 47);
			dlabio1 = distancia(vectorpuntos, 51, 66);
			d1nariz = distancia(vectorpuntos, 27, 28);
			n2++;
		}



		if (n2 == 1)
		{
			n2++;

		}

		for( int i = 0; i < n; ++i)
		{
			if(visibilities.at<int>(i))
			{
				if (GetAsyncKeyState(VK_SHIFT))
				{
					n2 = 1;
```

```
                        }
                        //char text[2];
                        std::string text ="";
                        cv::Point featurePoint(cvRound(shape2D.at<double>(i) *
(double)draw_multiplier), cvRound(shape2D.at<double>(i + n) * (double)draw_multiplier));
                        cv::Point numero(cvRound(shape2D.at<double>(i)+(5,-5)) ,
cvRound(shape2D.at<double>(i + n)+(5,-5)));
                        //guardar en vector actual

                        actual.push_back(featurePoint);
                        text = std::to_string(i);
                        if (borraunico[i]==1)
                        {
                                if (i >= 0 && i <= 16 && menton==1) //contorno de cara
                                {
                                        cv::circle(img, featurePoint, 1 *
draw_multiplier, cv::Scalar(255, 0, 0), 2, CV_AA, draw_shiftbits);
                                        if (numeroscara==1)//si numeroscara es 1 dibuja
el numero correspondiente encima del punto MGJ
                                        {
                                                putText(img, text, numero,
CV_FONT_HERSHEY_SIMPLEX, fuente, cv::Scalar(255, 0, 0), 1, CV_AA);

                                        }
                                }
                                else if (i >= 17 && i <= 26 && cejas==1) //Cejas
                                {
                                        cv::circle(img, featurePoint, 1 *
draw_multiplier, cv::Scalar(80, 127, 255), 2, CV_AA, draw_shiftbits);
                                        if (numeroscara==1)
                                        {
                                                putText(img, text, numero,
CV_FONT_HERSHEY_SIMPLEX, fuente, cv::Scalar(80, 127, 255), 1, CV_AA);

                                        }
                                }
                                else if (i >= 27 && i <= 35 && nariz==1) //nariz
                                {
                                        cv::circle(img, featurePoint, 1 *
draw_multiplier, cv::Scalar(0, 255, 0), 2, CV_AA, draw_shiftbits);
                                        if (numeroscara==1)
                                        {
                                                putText(img, text, numero,
CV_FONT_HERSHEY_SIMPLEX, fuente, cv::Scalar(0, 255, 0), 1, CV_AA);

                                        }
                                }

                                else if (i >= 36 && i <= 47 && ojos==1) //Contorno de
ojos
                                {
                                        cv::circle(img, featurePoint, 1 *
draw_multiplier, cv::Scalar(0, 0, 255), 2, CV_AA, draw_shiftbits);
                                        if (numeroscara==1)
                                        {
                                                putText(img, text, numero,
CV_FONT_HERSHEY_SIMPLEX, fuente, cv::Scalar(0, 0, 255), 1, CV_AA);

                                        }
                                }
                                else if (i >= 48 && boca==1) //boca
                                {
```

```
                                        cv::circle(img, featurePoint, 1 *
draw_multiplier, cv::Scalar(0, 255, 255), 2, CV_AA, draw_shiftbits);
                                        if (numeroscara==1)
                                        {
                                                putText(img, text, numero,
CV_FONT_HERSHEY_SIMPLEX, fuente, cv::Scalar(0, 255, 255), 1, CV_AA);

                                        }

                                }
                        }



                }
        }

        //Si n2==2 comparo distancia entre puntos de vectorpunto y actual, al
terminar libero memoria de actual
        if (n2 == 2)
        {



                calculaexpresiones(img,
actual,d1boca,d1cejaojo,dojoizq1,dlabio1,d1nariz);//llamar a la función con las
distancias calculadas en n2=0 y ya no pasarle vectorpuntos



                if (GetAsyncKeyState(VK_BACK))
                {
                        system("cls");
                }
                if (GetAsyncKeyState(VK_SHIFT))
                {
                        n2 = 1;
                }
        }
        actual.erase(actual.begin(), actual.cend());
    }

}
```

7.2.14 Función *calculaexpresiones*

```
void calculaexpresiones(cv::Mat img,vector<cv::Point2d> actual,double d1boca,double d1cejaojo,double dojoizq1,double
dlabio1, double d1nariz)
{
        //distancia entre dos puntos de la boca para identificar cuando se abre
        produceindiferencia = 0;
        produceira = 0;
        producemiedo = 0;
        producesorpresa = 0;
        producerepulsion = 0;
        double d2boca;
        d2boca = distancia(actual, 62, 66);

        //Distancia entre punto central de la ceja y el ojo para identificar cuando levanto las cejas o las bajo
```

```
double d2cejaojo;
d2cejaojo = distancia(actual, 21, 39);

//distancia entre dos puntos del ojo para identificar cuando lo cierro
double dojoizq2;
dojoizq2 = distancia(actual, 43, 47);

//distancia entre dos puntos de la nariz para probar a escalar la imagen cuando m acerque o aleje
double d2nariz, escala;



d2nariz = distancia(actual, 27, 28);
double factor=1.1;
escala = d2nariz / d1nariz;
double  dlabio2;
dlabio2 = distancia(actual, 51, 66);


if ((dlabio2 * factorLabiosApretados  < dlabio1*escala) && (empezar == 1) && !(d2boca > (1.1)*escala*d1boca))//Detecta labios apretados
{
	if (dojoizq2 / escala*factorOjosCerrados < dojoizq1)//Detecta ojos más cerrados
	{
		if (d2cejaojo / escala*factorCejasBajadas < d1cejaojo)//Detecta cejas bajadas
		{


			contrepul++;
			contind = 0;
			contira = 0;
			contmiedo = 0;
			contsorp = 0;

			if (contrepul>ignora)
			{

				putText(img, "repulsion", Point(100, 100), CV_FONT_HERSHEY_SIMPLEX, 1, cv::Scalar(0, 0, 255), 2, CV_AA);
				producerepulsion = 1;

			}

		}
		else
		{
			contrepul = 0;
			contind++;
			contira = 0;
			contmiedo = 0;
			contsorp = 0;


			if (contind > ignora)
			{

				putText(img, "indiferencia", Point(100, 100), CV_FONT_HERSHEY_SIMPLEX, 1, cv::Scalar(0, 0, 255), 2, CV_AA);

				produceindiferencia = 1;

			}

		}
	}
```

```
		else
		{
			contrepul = 0;
			contind = 0;
			contira++;
			contmiedo = 0;
			contsorp = 0;


			if (contira>ignora)
			{

				putText(img, "ira", Point(100, 100), CV_FONT_HERSHEY_SIMPLEX, 1,
cv::Scalar(0, 0, 255), 2, CV_AA);

				produceira = 1;

			}

		}


	}




	//boca abierta con escala
	if ((d2boca > (factorbocaAbierta)*escala*d1boca) && (empezar == 1))
	{
		//cejas levantadas con respecto al ojo o	cejas levantadas menos porcentaje y además que el ojo no
este cerrado
		if ((d2cejaojo > factorOjosAbiertos*escala*d1cejaojo) || ( (d2cejaojo > 1.1*escala*d1cejaojo)&&
(!(dojoizq2*escala*1.1 < dojoizq1))))
		{
			contrepul = 0;
			contind = 0;
			contira = 0;
			contmiedo++;
			contsorp = 0;

			if (contmiedo>ignora)
			{

				putText(img, "miedo", Point(100, 100), CV_FONT_HERSHEY_SIMPLEX, 1,
cv::Scalar(0, 0, 255), 2, CV_AA);

				producemiedo = 1;

			}

		}
		else
		{
			contrepul = 0;
			contind = 0;
			contira = 0;
			contmiedo = 0;
			contsorp++;

			if (contsorp > ignora)
			{
```

```
                                putText(img, "sorpresa", Point(100, 100), CV_FONT_HERSHEY_SIMPLEX, 1,
cv::Scalar(0, 0, 255), 2, CV_AA);

                                producesorpresa = 1;
                            }

                        }

                }
}
```

REFERENCIAS

[1] C. Darwin,"The Expression of the Emotions in Man and Animals",. John Murray, London, pp. 88–144, 1872

[2] A. Mehrabian,"Communication without words",Psychology Today, vol.2, no.4, pp 53–56, 1968

[3] P. Ekman, W.V. Friesen, "Constants across cultures in the face and emotion."Journal of Personality and Social Psychology, vol.17, no. 2, pp. 124–129, 1971

[4] J.A. Russell, "Is there universal recognition of emotion from facial expressions? A review of the cross-cultural studies," Psychological Bulletin, vol.115, no. 1, pp. 102-141, Jan.1994.

[5] P. Ekman, "Strong evidence for universals in facial expressions: A reply to Russell's mistaken critique," Psychological Bulletin, vol.115, no. 2, pp. 268-287, Mar.1994.

[6] C.E. Izard, "Innate and universal facial expressions: Evidence from developmental and cross-cultural research," Psychological Bulletin, vol. 115, no. 2, pp. 288-299, Mar. 1994.

[7] W.G. Parrott, "Emotions in Social Psychology," Psychology Press, Philadelphia, Oct. 2000.

[8] V. Bettadapura, "Face Expression Recognition and Analysis: The State of the Art". ,College of Computing, Georgia Institute of Technology, pp. 8-9, 2012

[9] I. Cohen, N. Sebe, A. Garg, L.S. Chen, and T.S. Huang, "Facial Expression Recognition From Video Sequences: Temporal and Static Modeling", Computer Vision and Image Understanding, vol. 91, pp. 160-187, 2003.

[10] A. Samal and P.A. Iyengar, "Automatic Recognition and Analysis of Human Faces and Facial Expressions: A Survey," Pattern Recognition, vol. 25, no. 1, pp. 65-77, 1992.

[11] https://cmusatyalab.github.io/openface/

[12] https://opencv.org/

[13] https://zenodo.org/record/1188976#.XniI9HJ7lPY