UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

SISTEMA DE ANÁLISIS DE PARTITURAS MUSICALES PARA SU DIGITALIZACIÓN Y REPRODUCCIÓN

Ingeniería Electrónica y Automática

Trabajo Final de Grado

19 de septiembre del 2018

Autor: ESCRIG VILLALONGA, MARC

Cotutor: SÁNCHEZ SALMERÓN, ANTONIO JOSÉ

Tutor: IVORRA MARTÍNEZ, EUGENIO

Resumen

Se desarrollará una metodología capaz de diferenciar las diferentes figuras musicales de una partitura para su digitalización, almacenamiento y reproducción. Los resultados pasarán por rigurosos filtrados para eliminar falsos positivos con el fin de crear un sistema robusto y sin errores. Antes de su almacenamiento y reproducción serán ordenadas en base a la cronología que dicta el lenguaje musical.

Resum

Es desenvoluparà una metodologia capaç de diferenciar les diferents figures musicals d'una partitura per a la seva digitalització, emmagatzematge i reproducció. Els resultats passaran rigorosos filtrats per eliminar falsos positius amb el fi de crear un sistema robust i sense errors. Abans d'emmagatzemar-los i reproduir-los seran ordenats d'acord amb la cronologia que dicta el llenguatge musical.

Abstact

A methodology capable of differentiating the different musical figures of a score for its digitization, storage and reproduction will be developed. The results will go through rigorous filtering to eliminate false positives in order to create a robust and error-free system. Before storage and reproduction will be ordered based on the chronology dictated by the musical language.

Índice general

Índice de figuras

1. Introducción

La gran expansión tecnológica ha traído al pequeño usuario herramientas de un potencial de creación y procesado que años atrás solo podían disfrutar los más privilegiados. La llegada de la visión artificial superando los límites del ojo humano ha revolucionado la industria durante estas últimas décadas, y a día de hoy se abre camino en nuestras vidas para instaurarse en todos los ámbitos de esta.

1.1. Motivación

Desde los inicios de la humanidad la música ha acompañado a la raza humana a lo largo de la historia. Al igual que nosotros, la música ha pasado por diferentes etapas que la han hecho evolucionar hasta llegar incluso a su digitalización. Hoy en día no existe la vida sin música, incluso algunos dirían que su vida es la música. Pero a pesar de esto, mucha gente no sabe que la música es un lenguaje como otro y que por ello tiene una forma de escritura. Educativamente no se ha dado la importancia necesaria y se tiene que optar por clases en escuelas de música paralelas al sistema educativo nacional. Por esto gran parte de las personas no adquieren en su educación los conocimientos básicos de este lenguaje. Con los años la curiosidad despierta en la mente humana y en muchas ocasiones adultos deciden llenar este vacío porque aman la música, y nunca es tarde para aprender.

Esto sirvió como motivación para crear una vía a disposición de todo interesado que quiera ampliar sus conocimientos, una vía al alcance de cualquiera. Ver como unos símbolos en un papel se transforman en una maravillosa melodía ayudará al alumno a entender mejor este mundo. Así se aprende que cualquier melodía que podamos imaginar siempre podrá ser plasmada en un trozo de papel.

1.2. Objetivos

- Elaborar un método generalizado capaz de encontrar las figuras musicales en una partitura y diferenciarlas unas de otras.

- Guardar los resultados de forma que todas las características del sonido a reproducir queden registradas y ordenar estos siguiendo la línea temporal del lenguaje musical.

- Reproducir los resultados caracterizando el sonido según la altura y duración que indique la figura que representa.

1.3. Teoría musical

El lenguaje musical es un conjunto de figuras, números y letras que permiten la escritura de los sonidos que componen la música. Este es para la música lo que la gramática y el vocabulario es para la lengua [1].

El sonido se compone de cuatro cualidades [2]:

- Altura: indica si un sonido es más agudo o más grave, depende de la longitud de onda
- Duración: determina su permanencia en el tiempo
- Intensidad: permite diferenciar la fuerza con la que se emite el sonido
- Timbre: distinguir su procedencia, depende de la forma de onda

Una partitura es un texto escrito que contiene los sonidos que componen una obra musical. Estas están formadas por pentagramas [Figura 2], que son conjuntos de cinco líneas dónde colocamos las figuras musicales. Cada sonido que queremos plasmar en una partitura ha de tener indicadas las cuatro cualidades explicadas anteriormente.

- Para representar la duración se utilizan los diferentes tipos de figuras. La negra es el tiempo base y el resto son múltiplos y divisores de este tiempo, como se aprecia en la Tabla 1. Si se añade un punto al lado de cualquiera de estas figuras su duración adquirirá un nuevo valor equivalente al original x 1.5.

Nombre	Figura para sonido	Figura para silencio	Duración
Redonda			4*T
Blanca			2*T
Negra			T
Corchera			T/2
Semicorchea			T/4
Fusa			T/8

Tabla 1. Tabla de figuras musicales

- Para representar la **altura** se usa la posición de las figuras en función de las líneas de pentagramas como se ve en la Figura 2. Cada posición se define como una nota diferente. En el lenguaje musical sólo existen 7 notas que se pueden encontrar en múltiples octavas: Do, Re, Mi, Fa, Sol, La, y Si. Estas pueden aumentar o disminuir su altura con sostenidos o bemoles, respectivamente [Figura 2]. Se añaden al lado de la nota y le suben (bemol) o le bajan(sostenido) medio tono.

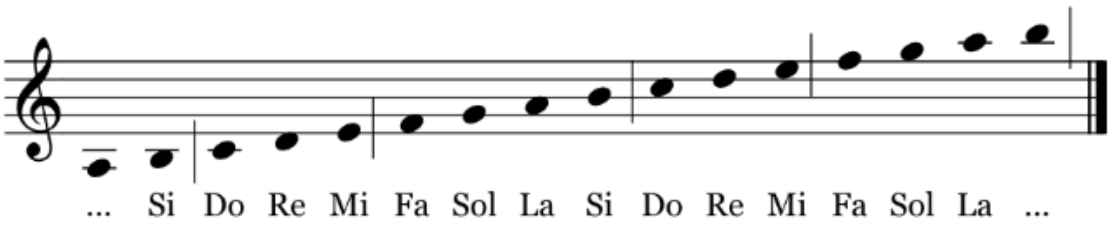


Figura 1. Notas musicales [11]

Figura 2. Sostenido (izquierda) y bemol (derecha)

- Para representar la **intensidad** este lenguaje cuenta con una serie de abreviaciones que se colocan debajo de las figuras para matizarlas en fuerza.

Nombre	Abreviatura	Significado
Pianíssimo	*pp*	Muy débil
Piano	*p*	Débil
Mezzopiano	*mp*	Medianamente débil
Mezzoforte	*mf*	Medianamente fuerte
Forte	*f*	Fuerte
Fortíssimo	*ff*	Muy fuerte

Tabla 2. Intensidades musicales

- El **timbre** se representa indicando el instrumento para el cual se ha escrito la partitura.

Además de las formas de representar el sonido, en una partitura coexisten más elementos de importancia, entre los que destacan:

- Las **claves**: las claves [3] son signos que se colocan al principio del pentagrama y sirven para dar nombre a las notas. Indican el nombre de la nota que está en la misma línea que la clave se asienta, y de ahí se deduce el resto. Actualmente hay tres tipos de claves, algunas de la cuales pueden aparecer en varias posiciones. Estas son la clave se Sol, Do y Fa [Figura 3].

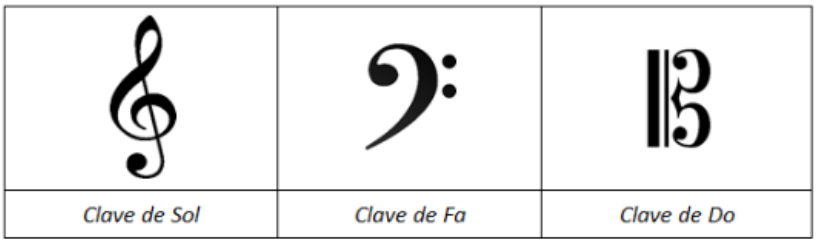


Figura 3. Claves musicales [3]

- El **tempo**: indica la velocidad con la que ejecutar una pieza musical [4]. Se expresa en pulsaciones por minuto y suele ir acompañada de la figura a la que hace referencia. Es decir, si aparece una negra igualada a un 60, esto quiere decir que cada negra valdrá un segundo.

- El **compás** es la división del pentagrama en partes de igual duración o igual suma de valores. Los compases se indican por dos números superpuestos: numerador y denominador [5].

    - Numerador: es el número que se ubica arriba. Indica la cantidad de tiempos o pulsos del compás.
    - Denominador: Es el número que se ubica abajo. Indica la figura que representa el pulso.

Ejemplo:

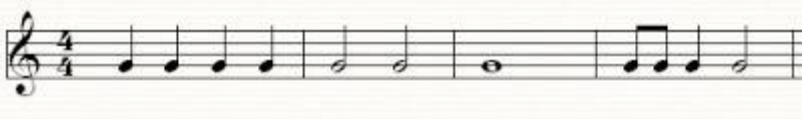


Figura 4. Ejemplo compás [5]

El numerador es 4, lo que indica que hay 4 tiempos. El denominador es 4, lo que indica que son negras.

Con esta introducción se habrán adquirido los conocimientos necesarios para consolidar una buena base con el fin de entender el proyecto a medida que se exponga. Añadir para terminar que lo que se expone en este apartado es el inicio de un complejo y laborioso lenguaje que necesitaría un trabajo entero para ser explicado.

1.4. Estado del arte

Los sistemas de visión por computador han demostrado ser una herramienta fundamental para automatizar diversos procesos.

Estos sistemas de visión permiten la inspección continua de procesos y productos, evitando fatigas y distracciones, y facilitando la cuantificación de las variables de calidad en prácticamente el 100% de la producción. Esto se traduce, no sólo en una mejora de la calidad final de los productos, sino también en un ahorro en términos económicos y medioambientales. Durante las últimas décadas se han podido resolver multitud de aplicaciones mediante la implantación de sistemas de inspección 2D en la industria. El principal problema a resolver en estos sistemas ha sido la gran variabilidad de las imágenes que puede dificultar la segmentación de los objetos de interés. Para resolver este problema se han propuesto algunas técnicas robustas de segmentación que intentan absorber dicha variabilidad [6][7]. Por otro lado, los últimos avances producidos en la aplicación de técnicas de conocimiento de objetos [10][8][9]están permitiendo automatizar algunas aplicaciones complejas que hace unos años eran impensables.

Con la creación de librerías de licencia libre cualquier usuario puede utilizar estas herramientas de visión artificial en cualquier proyecto imaginable, como por ejemplo en este caso el mundo de la música. Ya existen aplicaciones de precio asequible en el mercado que analizan partituras y las reproducen rápidamente, y no es de extrañar ya que este campo no tiene barreras. Un buen ejemplo seria la aplicación móvil SnapNPlay [12] que fácilmente te las reproduce y las transforma en pdf, o incluso otras no tan complejas como PhotoScore [13] que trabaja con pdfs de partituras y permite modificarlos para mejorarlos o simplemente adaptarlos a tu gusto.

En campos muy similares como el de análisis de escritura la competencia es mucho mayor. Decenas de aplicaciones y extensiones en algunos programas, como la de Google Translator [14]

que te detecta el texto y te lo traduce al instante. Incluso se pueden encontrar aquellas que traducen hasta tu propia escritura. Toda esta lucha ha llevado a la creación de nuevos métodos y mejora de los ya existentes.

2. Análisis e implementación

2.1. Análisis

En esta sección se pretende dar respuesta a los aspectos más importantes en el análisis de partituras con el fin de ahorrar el máximo tiempo posible durante el posterior desarrollo.

El proyecto busca elaborar y generalizar un método para la digitalización de partituras, de modo que se intentará crear un programa adaptable que pueda analizar el mayor rango de imágenes posible, ya que como se sabe la variabilidad del lenguaje musical es alta. A pesar de esto, como es lógico, se empezará por partituras simples y por figuras simples subiendo la dificultad a medida que el proyecto crezca. Por lo tanto, se eliminarán de las partituras ejemplo aquellas que contengan varias voces escritas en un mismo pentagrama (armónicos), o aquellas cuya complejidad comprometa el desarrollo inicial del programa.

Se centrará el resultado en que este sea real, es decir, antes de tener todas las figuras detectada es preferible que estas detecciones sean ciertas. Debido a la similitud de muchas de estas figuras se requerirán rigurosas estrategias de filtrado que permitan una óptima distinción del tipo de figura. El proyecto no solo debe centrarse en las figuras, también debe dar solución a la asignación de notas para cada correspondiente figura, y debe etiquetar todo para su posterior reproducción visual y sonora.

La vía de comunicación con el programa será la consola del programa que se vaya a usar, la cual mostrará los resultados del análisis de forma numérica, i permitirá interactuar con el programa.

Se procede pues a la selección de las herramientas que se van a utilizar y los problemas principales que analizar antes del inicio de la implementación del proyecto.

2.1.1. Herramientas seleccionadas

Para el desarrollo del proyecto se necesita una forma de analizar las imágenes que facilite la manipulación de estas, y también una plataforma en la que programar nuestro proyecto. Se ha optado como elección el uso de las librerías de OpenCV por varias razones. En primer lugar, se trata de código libre que puede ser usado para cualquier fin. En segundo, originalmente se creó en C++, lenguaje de programación que más se practica a lo largo del grado de Ingeniería Electrónica y Automática. Y en último lugar, es una de las librerías con mayor impacto en visión

artificial, por lo que se puede encontrar ejemplos de todo tipo y soluciones a cualquier duda en la web.

Como plataforma para llevar a cabo la escritura del código y su compilación se ha optado por el uso de Visual Studio 2015, una plataforma fácil de usar y de gran difusión que permite insertar con facilidad las librerías de OpenCV.

2.1.1.1. OpenCV

OpenCV es una librería de visión artificial diseñada originalmente por Intel para conseguir una eficiencia computacional elevada y enfocada en aplicaciones de tiempo real. Bajo una licencia BSD (Berkeley Software Distribution) que le permite un uso libre en fines académicos y comerciales, esta biblioteca es usada hoy en día en infinidad de campos y aplicaciones. Desarrollada originalmente en C/C++, hoy en día dispone de interfaces de desarrollo para Python Java y MATLAB siendo compatible con Linux, Mac OS X y Windows, así como también con dispositivos móviles Android y iOS [15].

La librería ofrece más de 2500 algoritmos que han sido optimizados llegando a abarcar un conjunto completo de algoritmos en el campo de la visión artificial y del aprendizaje automático que permiten al usuario implementar infinidad de proyectos.

OpenCV disponen de una estructura modular bloques [15] :

- Módulo Core. Incluye las estructuras de datos básicas y las funciones básicas de procesamiento de imágenes., entre ellos los arreglos multidimensionales Mat.

- Módulo Highgui Este módulo provee interfaz de usuario, códecs de imagen y vídeo y capacidad para capturar imágenes y vídeo, además de otras capacidades como la de capturar eventos del ratón…etc.

- Módulo Calib3d. Este módulo contiene algoritmos básicos para múltiples vistas, calibración de cámara, estimación de la postura de un objeto, algoritmos de correspondencia y elementos de reconstrucción 3D.

- Módulo Nonfree. El módulo contiene algoritmos que pueden estar patentados en algunos países o que tienen otras limitaciones de uso.

- Módulo Imgproc. Este módulo incluye algoritmos básicos de procesado de imágenes, incluyendo filtrado de imágenes, transformado de imágenes…

- Módulo Objdetect. Incluye algoritmos de detección y reconocimiento de objetos.

2.1.1.2. Visual Studio 2015

Visual Studio es un conjunto de herramientas y otras tecnologías de desarrollo de software basado en componentes para crear aplicaciones eficaces y de alto rendimiento, permitiendo a los desarrolladores crear sitios y aplicaciones web, así como otros servicios web en cualquier entorno que soporte la plataforma . Soporta multitud de lenguajes tales como Java, C++, C#, Visual Basic, Python, PHP… y el desarrollo de sitios y aplicaciones web [16][17].

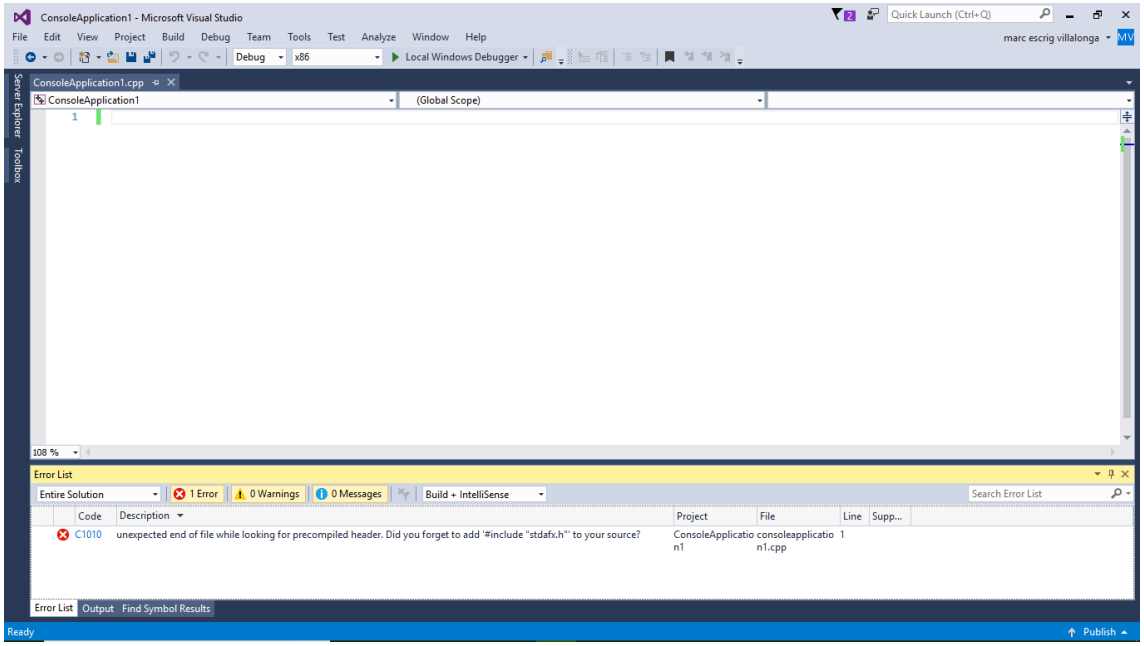


Figura 5. Entorno VS

Posibilita la creación de todo tipo de proyectos que se comuniquen entre móviles, consolas, dispositivos embebidos, páginas web y mucho más a través de un entorno de programación simple y ordenado [17].

2.1.2. Descripción de los principales problemas

Lo primero a tener en cuenta es como se va a detectar las diferentes figuras que componen una partitura para su posterior digitalización. Tras un breve estudio de las opciones que ofrece OpenCV se ve que las dos mejores son las siguientes:

- Detector en cascada
- Detector de patrones

La primera opción requiere un clasificador para cada tipo de figura que se quiera detectar. OpenCV ofrece herramientas para entrenar con imágenes estos clasificadores, pero el problema surge en el volumen de imágenes necesarias para un correcto funcionamiento de este. Se habla de un mínimo de 100 imágenes que contengan la figura que se quiere detectar y 500 que no la

contengan para empezar a tener un buen detector, por lo que para cada tipo diferente de figura musical se requieren como mínimo 600 imágenes de entrenamiento [18]. Si se tiene en cuenta la cantidad de figuras diferentes que hay se descarta como solución viable.

Por esta razón se elige la detección de patrones como método, para la que sólo se necesita una plantilla (una imagen) de la figura que se desea detectar y un tamaño con la que buscarla dentro de la imagen.

El proceso consiste en contrastar cada punto de la imagen con la plantilla, guardando para cada punto un coeficiente acorde al nivel de semblanza entre los dos. Posteriormente se filtrarán todas las posiciones y se elegirán aquellas con más posibilidad de contener la plantilla. En posteriores apartados se desarrollará con más detalle este proceso (aparatado 2.2.2.4 Búsqueda de figuras) [19].

Figura 6. Búsqueda de patrones [19]

El problema de este método surge cuando el objeto a buscar no es exactamente igual a la plantilla (por ejemplo, cuando está inclinado). Si no es exactamente igual quiere decir que el coeficiente de semblanza será menor, por lo que habrá que filtrar los resultados por un coeficiente menor. Esto conlleva a la aparición de falsos positivos que tendrán que ser procesados si se encuentra el modo de hacerlo. Añadir también que la plantilla necesita ser referencia con un tamaño de búsqueda determinado, así que se necesita una referencia de tamaño dentro de la imagen de búsqueda. Sin esta referencia se deberá de buscar con todos los tamaños posibles dentro de la imagen, lo cual se convierte en un proceso tedioso que consumirá mucho tiempo de procesado. De la elección del tamaño surge el segundo problema principal: encontrar una referencia de tamaño dentro de la partitura. La elección es simple, se escoge aquello que siempre está presente en todas las partituras y de cuyo tamaño depende el resto de objetos de la imagen: los pentagramas. Los pentagramas (apartado 1.3 Teoría musical) son conjunto de cinco líneas sobre los cuales de dibujan las figuras. Si se conoce el tamaño del pentagrama en la imagen de búsqueda también se conocerá el tamaño con el que buscar el patrón dentro de la imagen.

Antes de terminar, cabe tener en cuenta el material que se utilizará. Hay que seleccionar partituras con una calidad media como mínimo, merece la pena invertir más tiempo de procesado

que utilizar imágenes de baja resolución. A fin de cuentas, el objetivo es analizar, no mejorar imágenes que a lo mejor pueden analizarse.

Durante la próxima fase se abordarán más problemas surgidos a lo largo de la creación del proyecto y se detallarán los métodos utilizados para resolverlos.

2.2. Implementación

En esta sección se explicará el cuerpo del proyecto. Se empezará con una introducción teórica sobre los elementos utilizados de la librería OpenCV para un mejor entendimiento del desarrollo del proyecto. Después de esto se explicará detalladamente el proyecto parte por parte.

2.2.1. Introducción teórica a OpenCV

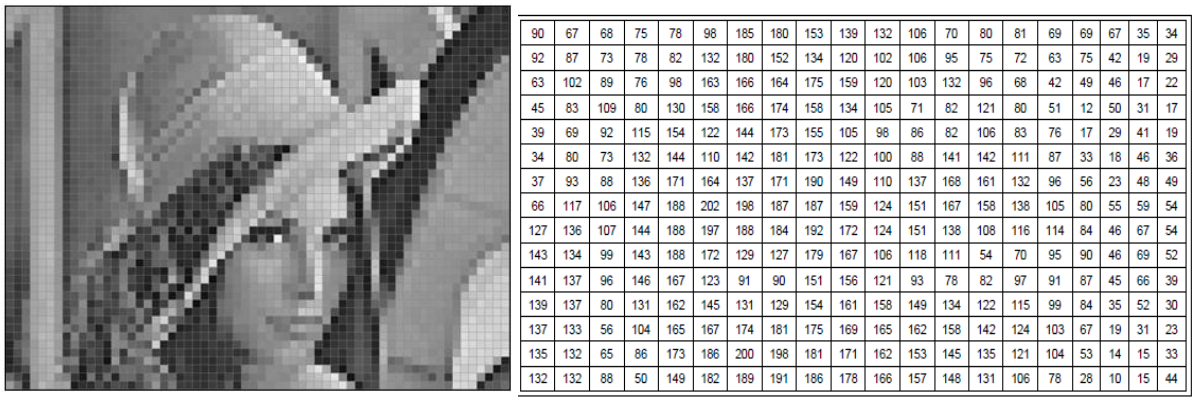
En ente subapartado se describirán las clases y funciones de la librería OpenCV más usadas durante el programa de un modo entendible para cualquier usuario principiante. Se da por hecho que el lector ya posee unos conocimientos básicos de lenguaje C para entender el proyecto, como por ejemplo los tipos de datos, punteros, bucles, vectores …

La base de OpenCV es el tratamiento de imágenes como matrices de números, aspecto que facilita la vida del programador. Para ello, desde los inicios de la librería se creó la clase Mat, la cual se explicará a continuación, así como algunas funciones esenciales que trabajan con ella y que han utilizado en el proyecto.

2.2.1.1. Imágenes y matrices

Para representar una imagen en formato digital se utilizan las matrices, donde cada celda representa un pixel. El color de este pixel estará representado por un valor numérico que pude variar según el formato que estemos utilizando para representar la imagen.

Para las imágenes a escala de grises cada pixel sólo necesita un canal (una matriz) y su valor varía de 0 a 255, siendo 0 el negro y 255 el blanco [Figura 6][20].



90	67	68	75	78	98	185	180	153	139	132	106	70	80	81	69	69	67	35	34
92	87	73	78	82	132	180	152	134	120	102	106	95	75	72	63	75	42	19	29
63	102	89	76	98	163	166	164	175	159	120	103	132	96	68	42	49	46	17	22
45	83	109	80	130	158	166	174	158	134	105	71	82	121	80	51	12	50	31	17
39	69	92	115	154	122	144	173	155	105	98	86	82	106	83	76	17	29	41	19
34	80	73	132	144	110	142	181	173	122	100	88	141	142	111	87	33	18	46	36
37	93	88	136	171	164	137	171	190	149	110	137	168	161	132	96	56	23	48	49
66	117	106	147	188	202	198	187	187	159	124	151	167	158	138	105	80	55	59	54
127	136	107	144	188	197	188	184	192	172	124	151	138	108	116	114	84	46	67	54
143	134	99	143	188	172	129	127	179	167	106	118	111	54	70	95	90	46	69	52
141	137	96	146	167	123	91	90	151	156	121	93	78	82	97	91	87	45	66	39
139	137	80	131	162	145	131	129	154	161	158	149	134	122	115	99	84	35	52	30
137	133	56	104	165	167	174	181	175	169	165	162	158	142	124	103	67	19	31	23
135	132	65	86	173	186	200	198	181	171	162	153	145	135	121	104	53	14	15	33
132	132	88	50	149	182	189	191	186	178	166	157	148	131	106	78	28	10	15	44

Figura 7. Imagen en escala de grises [20]

Otras representaciones como en formato RGB necesitan más canales por pixel, en este caso 3. En estos casos se utilizan matrices multidimensionales para satisfacer la necesidad de más canales para cada pixel. Sus valores también varían entre 0 y 255, y con la combinación de los tres canales se obtiene el color deseado [20].

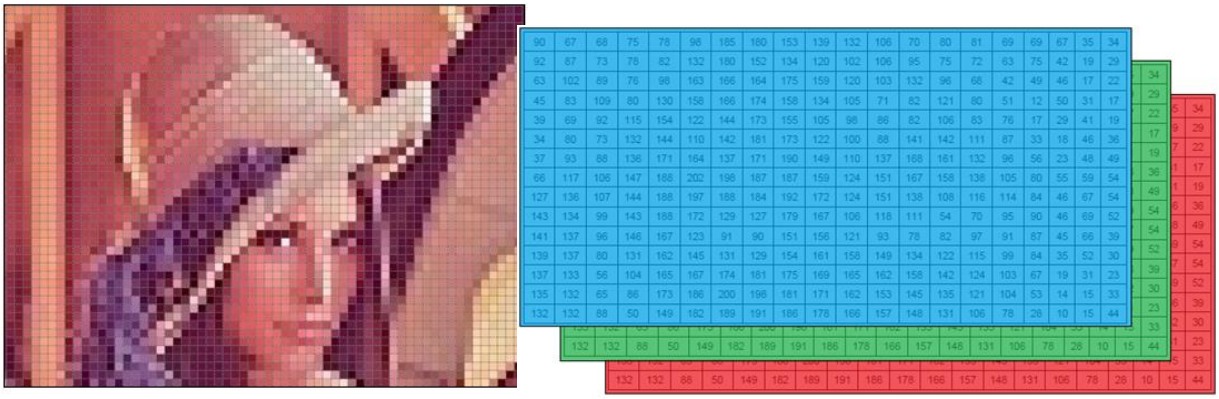


Figura 8. Imagen en escala RGB [20]

Un aspecto muy importante en el tratamiento de matrices es saber dónde se coloca el punto de referencia en relación a la matriz. En OpenCV la columna 0 corresponde a la columna de pixeles que está más a la izquierda, y la fila 0 es la fila de pixeles que está más arriba [Figura 9].

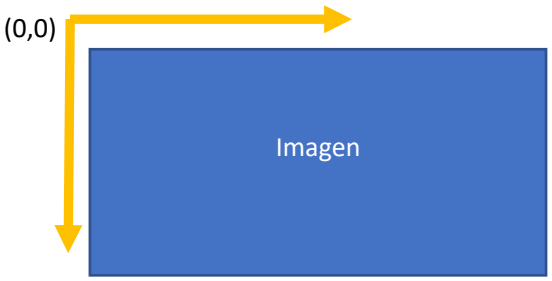


Figura 9. Origen de coordenadas

2.2.1.2. Clase Mat

La clase Mat es parte fundamental de la biblioteca OpenCV. Esta clase permite almacenar y manipular los pixeles de una imagen. Las funciones incluidas en la librería por lo general requieren un objeto Mat de entrada que pasa por un procesado y que genera otro objeto Mat de salida. Para su generación podemos usar su constructor [20]:

```
Mat(int filas, int columnas, int tipo)
```

- Filas: el número de filas de la matriz
- Columnas: número de columnas
- Tipo: aquí se indica el número de canales y el tipo de dato que guardará, siguiendo el siguiente formato.

```
CV_<profundidad de bits>{U|S|F}C(<número de canales>)
```

En esta forma *<profundidad de bits>* indica el número de bits que se utilizarán para el almacenamiento, puede ser: 8, 16, 32, etc., seguido se indica una de las letras: **U**, **S**, **F** que establece el tipo de dato, U sin signo, ejemplo: (0 - 255), S con signo, ejemplo: (-127 - +127), o un dato flotante F, ejemplo: (0.001 - 1.000), respectivamente, al final se indica la cantidad de canales que usaremos en *<número de canales>*, ejemplo: C3, C1, también se puede usar la forma: C(3), C(1)[20].

La clase Mat presenta varios métodos muy útiles y frecuentemente utilizados que facilitan el acceso a información de la propia matriz. Los utilizados en este proyecto son los siguientes [20][21]:

- .at

  Para tener acceso a un elemento de la matriz se utiliza este método. Para ello se le debe indicar el índice de la columna y la fila dónde buscar, teniendo en cuenta que la primera es 0. Además, se le debe indicar el tipo de dato de retorno, este debe ser igual o compatible con el tipo usado por la matriz.

  ```
  image.at<tipo de dato> (int fila, int columna)
  ```

- . cols

  Devuelve el número de columnas que tiene la matriz.

- .rows

  Devuelve el número de columnas que tiene la matriz.

- . empty()

  Devuelve 1 si la matriz está vacía.

- .copyTo(destino)

  Copia el contenido de la matriz a otra (destino).

2.2.1.3. Clase Rect

La clase Rect se utiliza para crear rectángulos en la imagen. Su constructor es el siguiente:

```
Rect(int x, int y, int anchura, int altura)
```

Donde x e y marcan el punto superior izquierdo del rectángulo dentro de la matriz y los dos siguientes, la anchura y altura en pixeles contando el origen[21].

En este proyecto los objetos Rect han sido usados con recurrencia en la búsqueda de patrones para guardar las posiciones.

2.2.1.4. Clase Point

La clase Point se usa para guardar las coordenadas de un punto (un pixel) [21]. Su constructor es el siguiente:

```
Ponit(int x, int y)
```

2.2.1.5. Clase Size

La clase Size permite guardar el tamaño de una región rectangular en pixeles [21]. Su constructor es el siguiente:

```
Size (int anchura, int altura)
```

2.2.1.6. Clase vector

Esta clase no pertenece a las librerías de OpenCV, pertenece a las de C++, pero su uso es tan recurrente en el proyecto que merece un espacio informativo. Los objetos "vector" son contenedores de datos o de otros objetos. Son de gran utilidad para guardar resultados y el acceso a algún dato contenido en ellos es fácil, sólo hay que saber la posición que ocupa en la pila.

```
vector<tipo de dato, numero de espacio del contenerdor>
```

Indicar el número de espacios que debe tener es optativo, si no se hace el propio objeto se adapta a la demanda de espacio. Los vectores permiten incluso crear vectores de vectores, lo que en pocas palabras es una matriz. Estos objetos contienen una serie de métodos, algunos parecidos a los ya explicados anteriormente, que nos facilitan su tratamiento [22].

- . at

  Para tener acceso a un elemento del vector se utiliza este método. Para ello se le debe indicar la posición dentro del vector donde buscar, sabiendo que el primer dato está en la posición 0 y el último en la posición (tamaño del vector – 1).

  ```
  vec.at(posición)
  ```

- . size()

  Devuelve el tamaño del vector.

- .push_back()

  Incrementa el tamaño del vector añadiéndole "dato" al final de vector, es decir, en la posición (tamaño del vector - 1).

  ```
  vec.push_back (dato)
  ```

- . pop.back()

  Decrementa el tamaño del vector en 1 borrando el dato que se encuentra en la última posición.

- .clear();

  Elimina todos los datos del vector, lo deja vacío.

2.2.1.7. Función imread

Esta función permite cargar las imágenes y transformarlas en matrices [21].

```
Mat imagen = imread (string nombre del archivo, int tipo de
                                color)
```

- En el primer parámetro se indica el nombre del archivo a buscar junto a su dirección dentro de la memoria. Por ejemplo:

  ```
  "C:/Users/HP/Documents/Visual Studio
  2015/Projects/AnalisisPartituras/AnalisisPartituras/partitura.jpg"
  ```

- El tipo de color:

  - = 0 la guarda en escala de grises.
  - <0 la guarda tal y como es, con todos los canales que tiene la imagen original.

2.2.1.8. Función cvtColor

Transforma una imagen de un formato a otro [21].

```
cvtColor (Mat img. de entrada, Mat img. de salida, int código,
                        int nº canales)
```

- En el parámetro "código" se indica la trasformación, su formato es el siguiente:

  ```
  CV_A2B
  ```

  donde A es el formato original y B al que lo vamos a transformar. Algunos de estos formatos son RGB, HSV, GRAY ..

- En "número de canales" se indica cuantos canales se quieren en la imagen de destino, si no se pone nada se adapta por el parámetro código.

2.2.1.9. Función threshold

Crea un umbral límite a partir del cual el valor del pixel pasa a ser otro [21].

```
threshold (Mat img. entrada, Mat img. salida, double umbral,
                double nuevo valor, int estilo)
```

- "umbral" es el valor que marca el límite sobre los dos grupos de números, los que están por encima y los que están por debajo.

- Representa el valor que se dará si el valor del píxel es más que (a veces menor) el valor del umbral.
- OpenCV ofrece varios estilos de limitar, y este se indica en el último parámetro. Ejemplos:

```
threshold (img,dst,127,255,cv.THRESH_BINARY)
threshold (img,dst,127,255,cv.THRESH_BINARY_INV)
threshold (img,dst,127,255,cv.THRESH_TRUNC)
threshold (img,dst,127,255,cv.THRESH_TOZERO)
threshold (img,dst,127,255,cv.THRESH_TOZERO_INV)
```

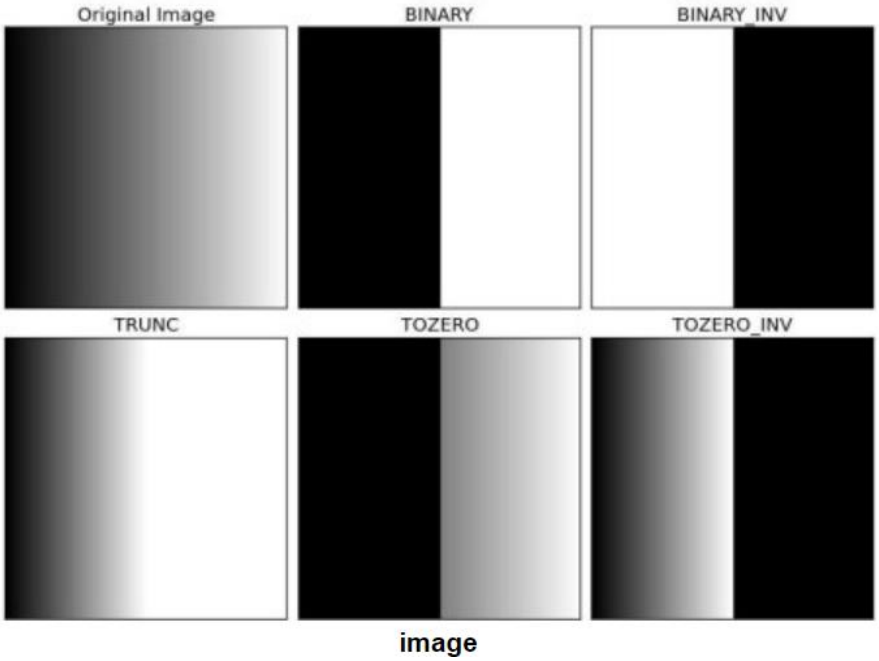


Figura 10. Estilos de umbral

En este proyector esta función permitirá transformar las partituras en una matriz binaria, cosa que facilitará la detección de los pentagramas y filtrar los resultados.

2.2.1.10. Función resize

Redimensiona una matriz con el tamaño que se le indique [21].

```
resize (Mat img. de salida, Mat img. de entrada, Size nuevo tamaño)
```

2.2.1.11. Función imshow

Crea una ventana y muestra la imagen [21].

```
imshow (Título de la ventana, Mat matriz que se va a reproducir como
                                imagen)
```

2.2.1.12. Función waitKey

Función de espera [21].

```
waitKey (int numero)
```

Si el número que se introduce es igual a 0 el programa espera a que se pulse alguna tecla para continuar. Si es mayor a 0 espera el número de milisegundo que le indiquemos.

2.2.1.13. Función destroyAllWindows

Destruye todas las ventanas que el programa ha creado [21].

```
destroyAllWindows ()
```

2.2.1.14. Función rectangle

Dibuja un rectángulo acorde a los parámetros indicados [21].

```
rectangle (Mat imagen, Rect rectángulo, CV_RGB/CV_GRAY color, int
              grosor, int tipo de línea, int shift)
```

- Imagen es la matriz sobre la que quedará dibujada el rectángulo.
- Rectángulo es el objeto que indicará la posición del rectángulo.
- Color indicará el color del rectángulo, o el brillo en escala de grises.
- Grosor indica el grosor de la línea del rectángulo, si es -1 el rectángulo se rellena con el color indicado en el parámetro anterior.
- Tipo de línea definirá el tipo de línea a dibujar:
    - 8 (u omitido) - línea 8-conectada.
    - 4 - 4 líneas conectadas.
    - CV_AA - línea antialiased.

- Shift Número de bits fraccionarios en las coordenadas del punto.

Esta función será de gran utilidad para marcar los resultados de la búsqueda de patrones sobre la imagen.

2.2.2. Desarrollo del proyecto

A continuación, se muestra un esquema general del proyecto para después analizar el funcionamiento de cada una de sus partes junto con los resultados de cada una de ellas.

2.2.2.1. Esquema general

El proyecto puede dividirse en 6 fases generales, las cuales se repetirán para cada partitura que se desee analizar:

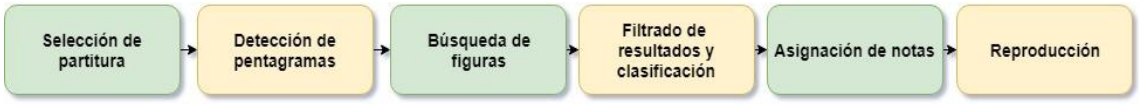


Figura 11. Esquema general del programa

- Selección de partitura. A través de la consola se seleccionará la partitura entre las disponibles y se cargará para su posterior tratamiento.
- Detección de pentagramas. Con la partitura preparada se aplicarán varios procesos a la matriz que detectarán los pentagramas.
- Búsqueda de figuras. Se aplicará la búsqueda de patrones con diferentes plantillas con el fin de clasificar todas las figuras que hay en la partitura.
- Filtrado de resultados. A los resultados obtenidos en la fase anterior se les aplicará una serie de filtros para eliminar los falsos positivos.
- Asignación de notas. A las figuras que se les tenga que asignar una nota de la escala se les asignará la que les corresponda dependiendo de su posición en el pentagrama.
- Reproducción. Reproducción visual y auditiva del resultado.

2.2.2.2. Selección de partitura

Con esta fase se inicia el programa. La consola servirá de vía de comunicación, permitiendo escribir el nombre de la imagen que se desee. A continuación, se detallan los subprocesos de la fase en orden cronológico:

1. Mostrar a través de la consola todos los archivos que hay en el directorio donde se guardan las imágenes de partituras [Figura 12].

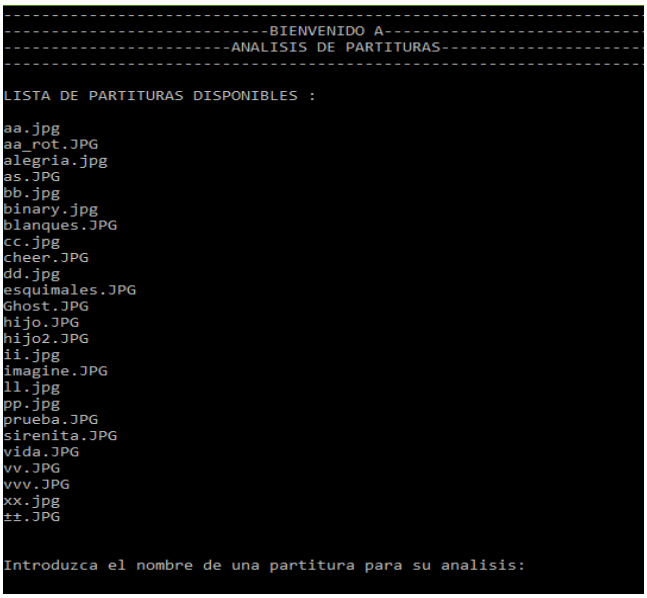


Figura 12. Lista de archivos

2. Se introduce el nombre de la partitura que se desea analizar.
3. Si el archivo existe se carga la imagen en escala de grises y se guarda en un objeto Mat de nombre **cdst** [Figura 13].

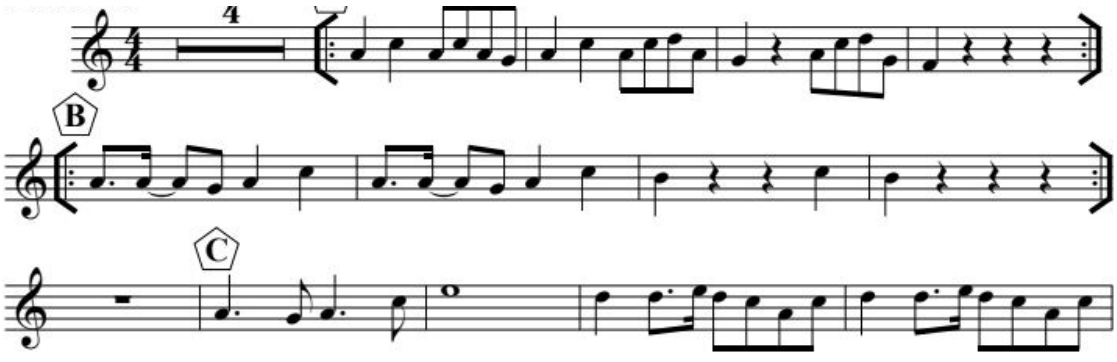


Figura 13. Imagen en escala de grises

4. Se crea una copia de la matriz y se transforma a binario usando la función *threshold* (apartado 2.1.2.9.) de modo que todo valor superior a 200 sea 255 (blanco), y todo valor inferior sea 0 (negro). Se guarda la matriz resultante en un objeto Mat llamado **bin** [Figura 14].

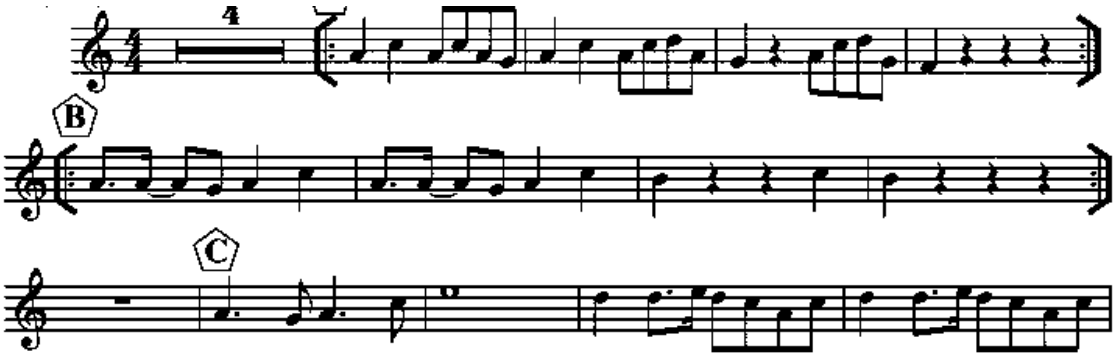


Figura 14. Imagen bin

5. El original **cdst** se transforma otra vez de escala de grises a RGB para poder dibujar en color encima de ella en prómimas fases. Para ello se usa la función *cvtColor* (apartado 2.1.2.9).

Terminada la fase, la imagen ya queda preparada para su uso a lo largo del análisis. En el Anexo I se puede observar el diagrama de flujo.

2.2.2.3. Detección de pentagramas

Se trata de la parte más importante del programa, ya que sin una buena detección no se podrá llevar a cabo las la búsqueda de patrones. El pentagrama determinará el tamaño de las plantillas, por lo que hay que asegurarse de que se obtiene un resultado lógico. Con una línea de un pentagrama sin detectar, el programa no debe de continuar ya que los resultados estarán basados en malas referencias. Esta fase empieza con la imagen ya preparada para el análisis. La binarizada (Mat **bin**) será la utilizada para analizar, y la RGB (Mat **cdst**) para mostrar los resultados. Este será el proceso llevado a cabo:

1. Se crea una función (*Buscar_posibles_pentagramas ()*) que nos devuelva las filas de la matriz que tienen más de un tanto por ciento de negro. Para ello, le pasaremos como dato de entrada la matriz binaria(bin) y la función se encargará de contar uno por uno los pixeles que componen cada fila. Si el valor de un pixel es negro se creará un incremento en una variable int (**color**) creada con el único fin de guardar los pixeles de color de cada fila. Si la variable **color** es superior al tanto por ciento indicado, se guardará dentro de un vector (**v**) el número de la fila correspondiente, ya que entrará como posible línea de pentagrama.

```
void  Buscar_posibles_lineas_pentagrama(Mat bin, vector<int> *v,
                                        float porcentaje);
```

Tras varias pruebas se ha tomado como porcentaje limite el 40 %. Se puede pensar que un valor mayor sería más restrictivo, pero a veces inducía a error a causa de cierta inclinación que tienen algunas imágenes.

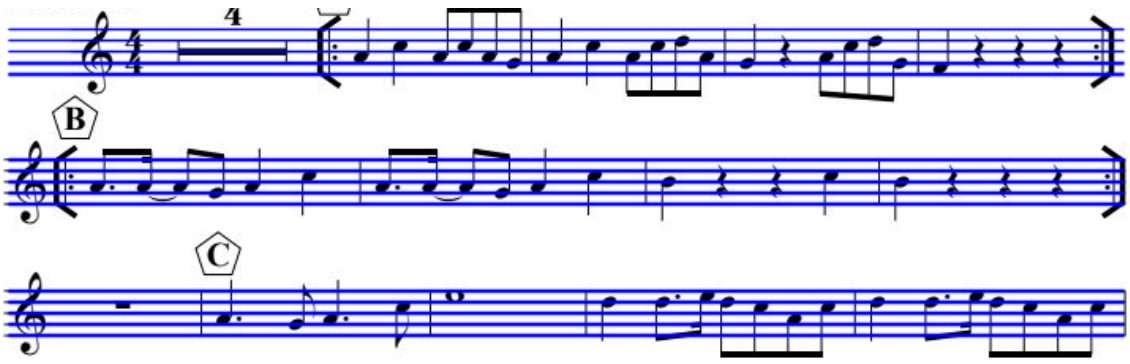


Figura 15. Posibles líneas de pentagramas

2. En ocasiones los bordes de la imagen pueden entrar en el grupo que cumple la condición anterior, por eso hay que crear un pequeño filtrado para eliminarlos. La función *Filtrar_pentagramas ()* se encarga de eliminar las primeras y últimas filas de la imagen si están dentro del vector de pentagramas.

```
void Filtrar_pentagramas(vector<int>*v, Mat cdst);
```

3. Otra función (*Inicio_Fin_Pentagramas ()*) se encarga de encontrar la columna de inicio y la final de los pentagramas. El proceso es similar al anterior. En primer lugar, se busca el inicio. Para ello, empezando desde la izquierda (columna 0) hacia la derecha de la imagen se irán observando los pixeles de cada columna. Si una columna tiene más de un 10 % de sus pixeles de color negro se parará la búsqueda y se guardará en la variable int **inicio** la columna correspondiente. Para calcular el final se repite el mismo procedimiento, pero de derecha a izquierda y se guarda la columna en la variable int **fin**.

```
void Inicio_Fin_Pentagramas (Mat bin, int *inicio, int
                             *final);
```

4. Para posteriores fases se necesita crear una imagen en la que se le hayan borrado las líneas de pentagrama. Sabiendo las filas que forman parte de estas líneas, se dibuja en ellas líneas blancas y guardamos el resultado en una nueva matriz llamada **cdst_white**. También se necesitará una binaria sin pentagramas por lo que se crea también **bin_white** [Figura 16].

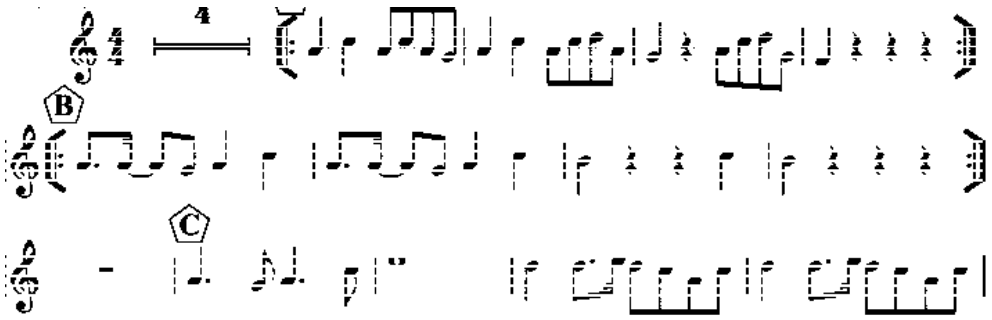


Figura 16. Partitura con pentagramas eliminados

5. Para cada línea de pentagrama no solo existe una fila de la imagen que la componga, de modo que las que formen parte de una misma línea tienen que juntarse y guardarse como un único resultado. Para ello se hará la media de todas ellas. La función *Fusionar_lineas ()* se encarga de detectar todas las filas que componen una línea y de calcular su media. Al final de la función se guardan en el mismo vector de entrada los resultados, con lo que para cada línea de pentagrama le corresponderá solo una fila de la imagen. El método que usa es simple, compara unas con otras y las que están a pocos pixeles de distancia (4 como máximo) las guarda en un mismo vector. Se calcula la media de todas y se elige este valor como fila de la línea.

```
void Fusionar_lineas(vector<int> *v)
```

6. Si después de fusionar todas las líneas queda un resultado que no es múltiplo de 5 quiere decir que no se han detectado todas las líneas de todos los pentagramas y, por lo tanto, el posterior análisis será erróneo. Como salida se vuelve a la fase anterior, *Selección de partitura*, para poder probar otro ejemplo.

7. Si no hay errores, mediante la función *Ordenar ()* se reordenan de forma ascendente los datos que componen el vector de pentagramas. Y finalmente la función Agrupar () guarda los datos en un vector de vectores (`vector<vector<int, 5>>` **`vector_pentagramas`**) de modo que un pentagrama formado por 5 números (sus 5 líneas) se guarda como un solo dato en el vector de vectores. Se muestra el resultado por pantalla [Figura 17].

```
void Agrupar (vector<int> v, vector<Vec<int, 5>> *vector_pentagramas);
                    void Ordenar(vector<int> *v);
```

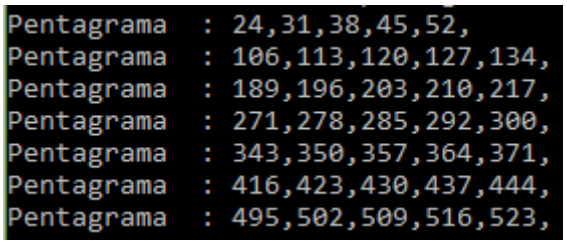


Figura 17. Filas de la imagen donde se encuentran los pentagramas

Finalizada esta fase se tendrán los pentagramas guardados en un vector, a disposición de las próximas fases. También la imagen cdst_white a la que se le han eliminado los pentagramas. En el Anexo I se puede observar el diagrama de flujo.

2.2.2.4. Búsqueda de figuras

Esta es la fase en la que se llevará a cabo la búsqueda de los diferentes tipos de figuras musicales gracias a la búsqueda de patrones. Con anterioridad en este proyecto se ha explicado de forma general el funcionamiento de la búsqueda de patrones. Aquí se explicará su funcionamiento de forma matemática y detallada.

La función que nos ofrece OpenCV es la de *matchTemplate(),*la cual necesita como parámetros la plantilla, la matriz donde buscarla, la matriz donde se guardarán los coeficientes de semblanza y el método utilizado de comparación. Esta función se encarga de aplicar en cada punto una ecuación de correlación entre la plantilla y el punto, y guardar su resultado.

```
void matchTemplate(Mat imagen, Mat plantilla, Mat resultado, int modo)
```

Se ha elegido como modo el método de correlación de coeficiente normalizada, el cual aplica a cada punto de la matriz la siguiente ecuación:

$$R(x,y) = \frac{\sum_{x',y'}(T'(x',y') \cdot I'(x+x',y+y'))}{\sqrt{\sum_{x',y'} T'(x',y')^2 \cdot \sum_{x',y'} I'(x+x',y+y')^2}}$$

Obteniendo como resultado una matriz de coeficientes, donde los más grandes marcarán los lugares más probables de encontrar la plantilla.

El tamaño de la plantilla es crucial en la búsqueda, por lo que tiene que ser escalada en relación a la imagen. Como ya se ha explicado con anterioridad, se usarán para ello los pentagramas. Al crear las plantillas se tiene que tener en cuenta la altura, siempre se tiene que recortar la plantilla como si recortases también el pentagrama que la lleva, es decir, si se mira en la Figura 18 el cuadrado verde indica el recorte correcto, el que se adapta al pentagrama. El recorte rojo sería el incorrecto ya que ese se ajusta a la figura y no habría manera de escalar la plantilla.

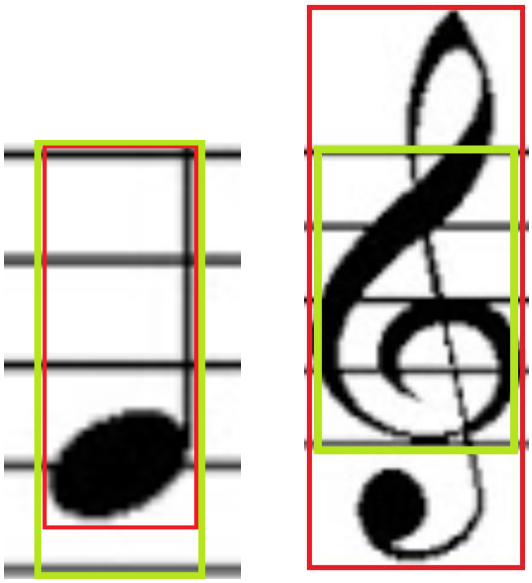


Figura 18. Creación de plantillas

Si se sabe la altura original de la plantilla y la que va a tener en el pentagrama se puede calcular el coeficiente que relaciona estas dos para poder aplicarlo posteriormente al ancho:

$$h_{pentagrama} = h_{plantilla} * \alpha \rightarrow \alpha = \frac{h_{pentagrama}}{h_{plantilla}}$$

$$b_{pentagrama} = b_{plantilla} * \alpha$$

Donde la h y b pentagrama son las dimensiones con las que buscar la plantilla dentro de la imagen, y h y b plantilla son sus dimensiones reales.

El proceso de búsqueda de patrones se lleva a cabo con una función llamada *Buscar_Figuras* () la cual devuelve un vector de objetos Rect que contienen las posiciones de las zonas que superan el coeficiente umbral(coef) que s ele ntroduce a la función.

```
vector<Rect> Buscar_Figuras (int altura_pentagrama, string
dirección_plantilla, double umbral, Mat* Imagen donde buscar)
```

Esta contiene a su vez la función *matchTemplate ()* explicada arriba. El proceso de búsqueda es el siguiente:

1. Se calcula la altura del pentagrama para introducirla como parámetro de entrada a la función *Buscar_Figuras ()*. Para ello se usa el vector de vectores del apartado anterior que contiene las líneas de los pentagramas. Se restan la primera y la última línea del primer pentagrama y se obtiene la altura de este. De este modo ya se puede calcular la relación como se ha explicado con anterioridad.

2. Se ejecuta la función *Buscar_Figuras ()*. En primer lugar, la función reescala la plantilla en relación al pentagrama como se menciona arriba. A continuación, se transforma a escala de grises tanto la plantilla como la imagen donde buscarla. Creamos un objeto Mat

resultado donde se guardarán los coeficientes de cada punto y ya se le aplica la función *matchTemplate ()* de OpenCV. Cada valor es comparado con el umbral que se introduce como parámetro a la función *Buscar_Figuras ()*. Uno de 0.7 es el que más se adecua después de muchas pruebas. Con uno inferior se obtienen demasiados falsos positivos y con uno superior quedan muchas figuras sin detectar. Aquellas posiciones que superen ese valor se guardan en un vector de objetos Rect añadiéndole la altura y el ancho de la plantilla. Finalizada la comparación de todos los valores se devuelve el vector.

3. Para cada tipo de figura se aplicará la función *Buscar_Figuras ()* varias veces con diferentes plantillas por razones que posteriormente explicaremos. Por lo tanto, los diferentes vectores obtenidos tendrán que sumarse en uno con la función *unir*(vector<Rect> total, vector<Rect>unir), ya que todos son de la misma figura. Esta función le suma al primer vector el segundo.

```
void unir(vector<Rect> *total, vector<Rect>*unir)
```

4. Como es lógico al aplicar varias plantillas habrá resultados doblados, por ello se ha creado la función *eliminar_dobles*(vector<Rect>vector) que se encarga de corregir este error. Esta función considera una figura doble de otra cuando el rectángulo que la contiene no dista más de 3 pixeles en el eje X de otro rectángulo, ni más de x pixeles en el eje Y, dónde x es igual a la altura del pentagrama [Figura 19].

```
void Eliminar_dobles(vector<Rect> *vector)
```

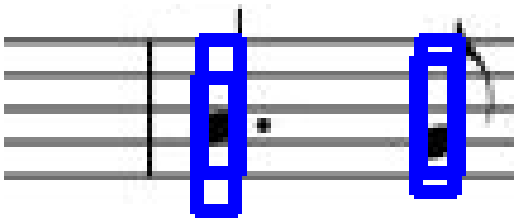


Figura 19. Figuras dobladas

Ahora que ya se conoce el proceso de búsqueda para una plantilla determinada hay que determinar qué plantillas se van a usar para cada figura. En general, hay dos formas de buscar la figura. Una es hacerlo en la partitura normal (**cdst**), y la otra es buscar la figura en la partitura que se han borrado los pentagramas (**cdst_white**) [Figura 16].

Dependiendo el tipo de figura hay que usar un método o los dos:

- Silencios de blanca / Silencios de redonda

  A causa de su simplicidad, estas figuras solamente se pueden buscar con pentagramas. Si se eliminan las líneas su forma se transforma en un pequeño rectángulo pudiendo dar

más falos positivos [Figura 20]. Además, siempre ocupan la misma posición dentro de un pentagrama, cosa que facilita su filtrado.

Figura 20. Silencio de blanca con(derecha) y sin(izquierda) pentagramas

- Silencio de corchera/silencio de negra/Claves/Compases

  Estas figuras al igual que las anteriores siempre ocupan la misma posición dentro del pentagrama, pero además su forma compleja [Figura 21] permite obtener también buenos resultados habiéndose eliminado los pentagramas, incluso mejores en algunas ocasiones.

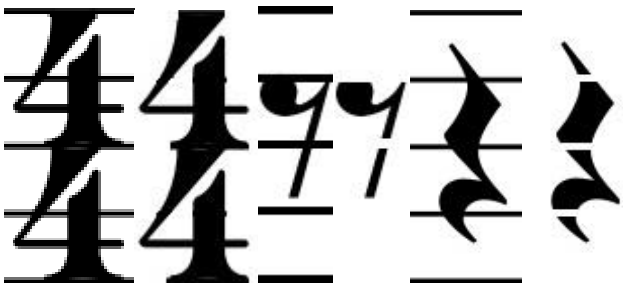


Figura 21. Figuras con y sin pentagrama

- Negras / Blancas /Redondas

  Las negras, las blancas y las redondas son figuras que tiene la peculiaridad de moverse arriba y abajo en el pentagrama dependiendo de la nota que representen. Además, añádele a las negras y blancas que su posición se invierte generalmente cuando pasan la línea del medio del pentagrama . Esto quiere decir que para este tipo de figuras se necesitará una plantilla para cada posible forma de aparición [Figura 22][Figura 23][Figura 24].

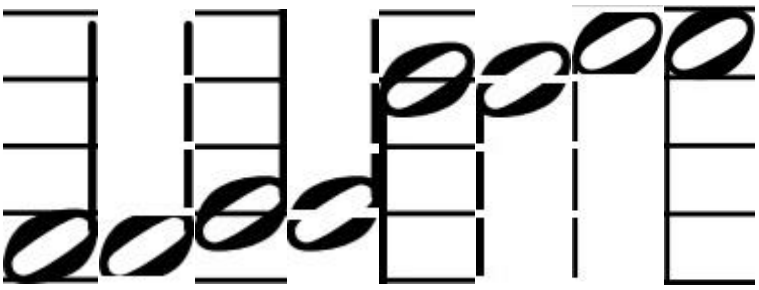


Figura 22. Posibles formas de aparición de una blanca

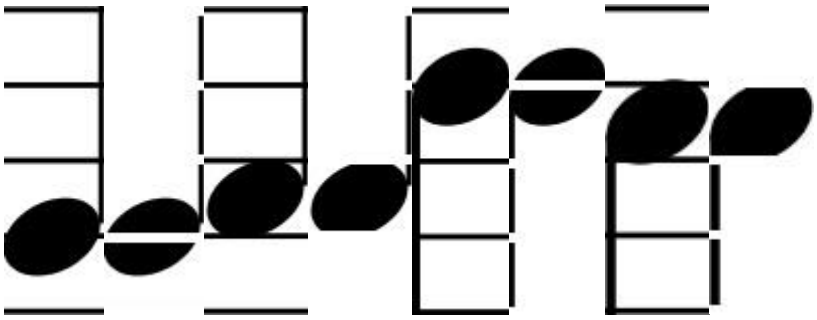


Figura 23. Posibles formas de aparición de una negra

Figura 24. Posibles formas de una redonda

En estos casos se obtiene mucho más éxito sin pentagramas, debido a que en las plantillas con pentagrama la figura está en una posición concreta, una de las muchas que se pueden encontrar en una obra musical. Si no está en la misma posición los pentagramas bajarán el coeficiente de correlación que indica la semblanza. En las redondas se opta directamente por buscarlas sin pentagramas ya que no se invierten ni nada, con sólo dos plantillas se consiguen los mismos resultados que si hubiese 4.

Si se quisiese hacer un buen análisis al completo se utilizarían las plantillas con pentagramas en todas las posibles posiciones, ya que usar las que no tienen pentagramas es incrementar el número de falsos positivos. Por otra parte, cuantas más plantillas más computo. En este proyecto se han usado las de las plantillas de las figuras 22,23 y 24, con las que se han obtenido bastante buenos resultados, aunque siempre mejorables.

Todas estas plantillas seguirán el proceso anteriormente descrito, pasando por las funciones *Buscar_Figuras(), unir ()* y *eliminar_dobles ()* y se guardarán sus resultados en vectores de objetos Rect.

Las corcheras y semicorcheas no han entrado en ninguna de las clasificaciones ni han sido mencionadas. Esto se debe a su semejanza con las negras. El parecerse tanto impide distinguir unas de otras mediante la búsqueda de patrones [Figura 25]. Por lo tanto, en esta fase todas estarán incluidas en el mismo vector y será en la siguiente fase cuando se filtren para separarlas.

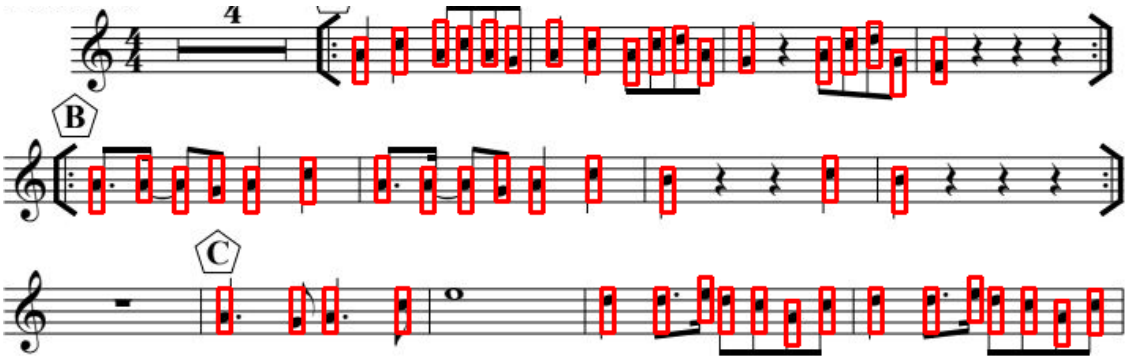


Figura 25. Resultado búsqueda con plantillas de negra. Corcheras y negras siguen el mismo patrón

En el Anexo I se puede observar el diagrama de flujo.

2.2.2.5. Filtrado y Clasificación

A lo largo de esta fase se filtrarán todos los resultados obtenidos con el fin de eliminar el máximo número de falsos positivos. Una vez terminada, los resultados serán clasificados para su reproducción en la última etapa.

Como se verá a continuación, no hay un filtrado para separar las semicorcheas y las fusas de las corcheras y negras. Por esta razón no se hará referencia a ellas en la fase de filtrados y se incluirán al grupo de las corcheras. Se deja esta parte para futuras ampliaciones del proyecto (apartado 3.). Se procede pues al filtrado del resto de figuras.

2.2.2.5.1. Filtrado de Negras/Corcheras

En primer lugar, se filtrará el vector que contiene tanto negras como corcheras con el fin de eliminar el mayor número de falsos positivos. Estas figuras comparten características que serán utilizadas para este proceso. Para ello se han desarrollado tres formas diferentes de hacerlo:

- En primer lugar, se buscará la característica barra negra que acompaña a la elipse negra dentro de cada rectángulo donde haya una negra o corchera [Figura 23]. Se mirará pixel a pixel cada una de las columnas dentro del rectángulo. Si hay una columna que tenga más de un 50% de pixeles negros se mantendrá en el grupo. Si no se da el caso, se eliminará.

- El segundo método va dirigido a aquellos falsos positivos causados por las blancas y las redondas. Hay veces que se incluyen dentro del grupo de las negras y, por ejemplo, las blancas tienen palo, por lo que no se eliminaran con el primer método. En este caso lo que se hace es buscar la elipse negra característica de las negras y las corcheras, semicorcheas y fusas. Las blancas y redondas también cuentan con una elipse característica, pero esta no tiene tanta concentración de negro. Por esta razón lo que se hará será recorrer con un rectángulo de anchura igual a la anchura de la plantilla, y de altura igual a la mitad del espacio entre dos líneas de pentagrama, el interior del rectángulo que contiene la negra. Si en algún punto el rectángulo que se desplaza tiene más de un 85% de negro en su interior se considerará que es una negra o corchera. Si todos tienen menos de un 85%, se eliminará del vector [Figura 26].

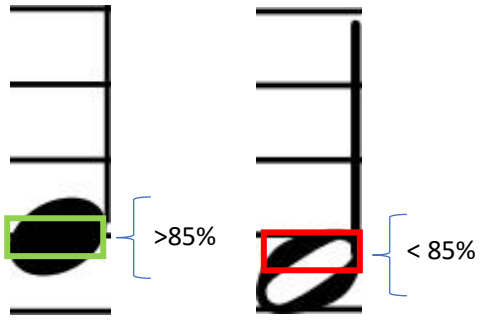


Figura 23. Filtrado de blancas de entre las negras

- El último método es el más complejo de todos. Se ha diseñado para el caso concreto que ofrece la Figura 27. Ocurre que el buscador de patrones encuentra negras en la barra que une las corcheras, creando falsos positivos.

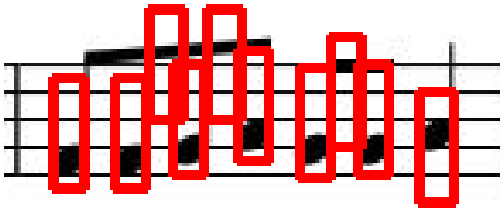


Figura 27. Falsos positivos

Para descartar este tipo de errores se siguen los siguientes pasos:

  - Primero se ordena todo el vector de negras/corcheras según el orden en el que se reproducirían dado el caso, es decir, que el primer objeto Rect sea la negra más a la izquierda del primer pentagrama y el último la que esté más a la derecha del ultimo pentagrama.

  - A continuación, se calcula la distancia de cada elemento del vector con el siguiente. Si sus rectángulos distan menos de 3 pixeles en el eje X, menos de la cantidad de pixeles equivalente a la altura del pentagrama en el Y, y más del espacio entre dos líneas de pentagrama en el Y, se considera que uno de ellos es un falso positivo causado por la barra de corcheras.

  - Que uno de los dos sea un falso positivo implica que cerca tiene que haber como mínimo una corchera que sea la pareja de la que no es un falso positivo. Por esto, lo siguiente que se hará será medir las distancias con la figura posterior y la anterior, es decir, si observamos la Figura 28 se medirá la distancia entre 1 y 2, y la distancia entre 3 y 4. La distancia más pequeña nos dirá cuál de los rectángulos (posterior o anterior, 4 o 1) es la corchera que forma pareja con la que no es un falso positivo.

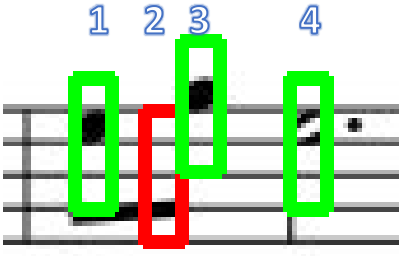


Figura 28. Posibles negras/corcheras

- Ahora que se sabe cuál es la compañera, se debe buscar la elipse negra de las dos candidatas a falso positivo y de la compañera corchera. Para cada rectángulo se mirarán todas sus filas y aquellas que tengan más de un 50% de sus píxeles en negro se guardarán en un vector auxiliar. De todas las líneas guardadas en el auxiliar se hará la media. El resultado será un valor parecido al punto central de la elipse (no se necesita una gran precisión) ya que aquí están las filas con mayor concentración de negro. Por supuesto esta detección de líneas se realiza sobre la matriz sin pentagramas, de modo que no influyen en encontrar el centro de la elipse negra.

- Con los tres puntos calculados se compara las dos candidatas con la que se conoce que es una corchera real. Aquella que diste más de la compañera corchera será el falso positivo. Este criterio se basa en que la barra que une estas dos corcheras siempre está por encima o por debajo de las dos elipses, por eso se puede usar para estos casos.

Estos son los pasos que hay que aplicar a todos los rectángulos del vector de negras/corcheras, con lo que se obtendrá el resultado de la Figura 29.

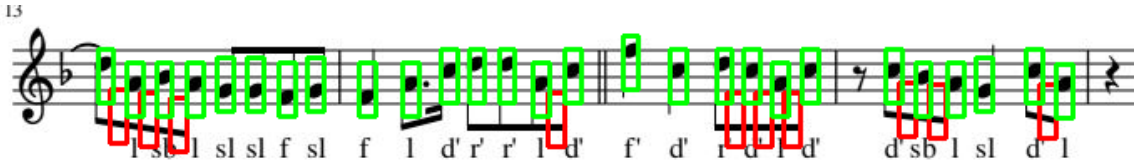


Figura 29. Método 3 del filtrado de negras/corcheras

Aunque se trata de un buen método para eliminar estos errores sucede a veces que dos notas están demasiado cerca y una de ellas acaba siendo considerada falso positivo. Por lo que en futuras ampliaciones habría que buscar una solución para esto.

Una vez aplicados los tres métodos el número de elementos del vector que contiene las negras y corcheras se ha reducido considerablemente, por lo que ya se le puede aplicar el método de separación creado.

2.2.2.5.2. Separación Negras y corcheras

Como se ha comentado con anterioridad, la búsqueda de patrones no permite clasificar por separado las negras de las corcheras, debido a que la forma de la corchera contiene la de la negra. Por tanto, usando las plantillas de la negra detectaremos también las corcheras [Figura 29].

La figura de la corchera es especial. Puede aparecer sola o por separado tal como se ve en la Figura 30.

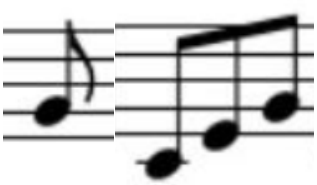


Figura 30. Formas de aparecer las corcheras

Se empieza pues por encontrar las corcheras que van en conjunto. Estas se caracterizan por ir unidas con una barra negra, y pueden aparecer con su posición normal o invertidas, dependiendo de la altura del conjunto dentro del pentagrama [Figura 31]

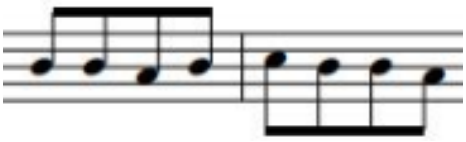


Figura31. Diferentes tipos de conjuntos de corcheras

El método desarrollado en este proyecto se basa en analizar las zonas cercanas donde se ha detectado la nota para ver si se encuentra algún cumulo de negro correspondiente a la barra que las une. Para ello se deben de determinar varios aspectos antes de la búsqueda:

- El tamaño del cuadrado tiene que basarse en alguna medida que venga dada por la imagen. Se escoge como altura la mitad de la distancia que hay entre dos líneas de pentagrama, y como ancho, la distancia entera entre dos líneas. Estas serán las medidas del rectángulo cuyo interior se medirá la cantidad de negro para saber si hay una barra de las características de corcheras o no.

- El segundo aspecto a determinar es el área de recorrido de este rectángulo. Si nos fijamos en la figura 32, dependiendo de la corchera hay que buscar la barra en una zona. La flecha azul señala una corchera que tiene la barra arriba a la derecha, la roja señala una que la tiene encima de ella, la amarilla la tiene debajo de la corchera y la verde abajo a la izquierda.

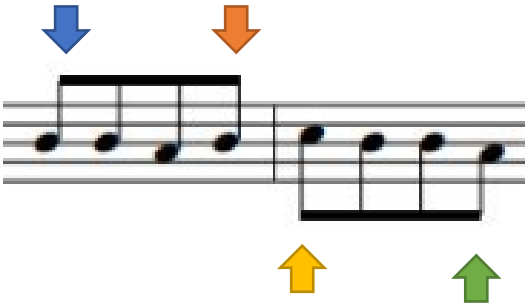


Figura 32. Zonas donde encontrar la barra de corcheras

Esto quiere decir que para cada patrón encontrado habrá que buscar en todas estas zonas. En este punto, uno se podría preguntar si no sería más fácil buscar en la zona de arriba para los resultados de la plantilla de negra normal, y en la de abajo para los resultados de la plantilla invertida. Sería una buena forma de quitar tiempo de computación, pero, las plantillas de negra normal detectan las negras inversas y viceversa. Por esto para todas las plantillas hay que buscar en todas las zonas.

Uno de los problemas buscando en estas zonas surge cuando hay que hacerlo justo encima o debajo de la figura. Ocurre en ocasiones que la misma barra negra característica de las corcheras está parcialmente dentro del rectángulo que marca la figura y al buscar la barra por encima o debajo de esta zona el programa no la encuentra. La solución por la que se optó consiste en cambiar las plantillas en la etapa de buscar las figuras. Las nuevas cuentan con un espacio en blanco arriba (en la de negra al revés) y abajo (en la de negra normal) tal como se muestra en la Figura 33. Esto permite empezar la búsqueda dentro de la zona donde se ha detectado el patrón para asegurar un optima detección como se ve en la Figura 34.

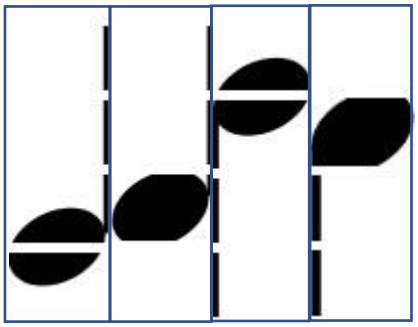


Figura 33. Plantillas de la negra usada. Se aprecia la zona blanca

Aclarados estos aspectos, ya se puede entender cómo va a trabajar la función Negras_Y_Corcheas (). El rectángulo recorrerá las zonas que se indican en la Figura 34 y si en algún punto de ellas el relleno del rectángulo contiene más de un 85% de píxeles negros, la negra pasará a ser corchera.

```
vector<Rect> Negras_Y_Corcheras(vector<Rect> negras/corcheras,
vector<Vec<int, 5>> *vector_pentagramas, int altura_pentagrama, Mat
*cdst, Mat *bin)
```

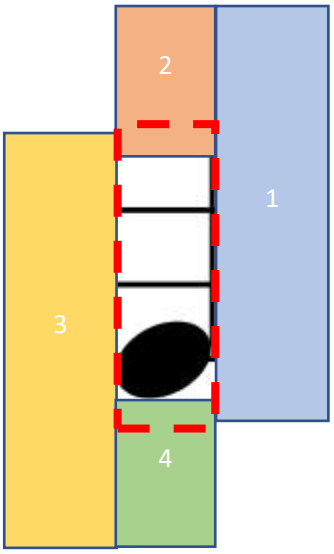


Figura 34. Zonas de búsqueda de la barra de corchera

Las zonas 1 y 3 están fuera del rectángulo donde se ha encontrado el patrón (rectángulo rojo de rallas discontinuas). En cambio, la 2 y 4 sí que entran dentro de esta zona por las razones explicadas arriba. Para ser exactos entran dentro de la zona una distancia equivalente a la mitad del espacio que hay entre dos líneas del pentagrama.

Cuando se encuentre una zona negra la función dibujará un rectángulo para poder comprobar los resultados, como se aprecia en la figura 35.

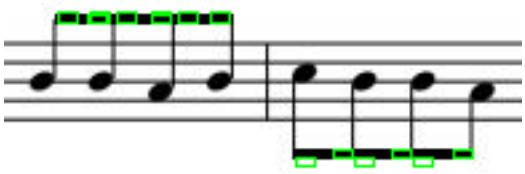


Figura 35. Detección de barras de corcheras

En general es un buen método de detección de corcheras, pero en ocasiones las notas están demasiado agrupadas y se crean errores de detección, como el la Figura 36. Lo bueno es que estos cúmulos suelen darse entre corcheras, por lo que, aunque sea una mala detección, no influye en el resultado porque ya se incluía dentro del grupo de corcheras. El resto de figuras suele respetar más el espacio unas con otras.

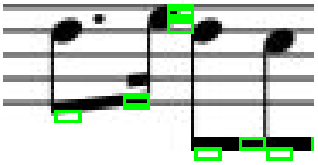


Figura 36. Error de detección por proximidad de notas

Finalizada la función se tiene un vector de objetos Rect con todas las negras que han pasado a ser corcheras. El siguiente paso es eliminar estas del vector de negras, así ya estarán separadas definitivamente. Para ello se ha creado la función *excluir()*. Esta permite eliminar del vector **total** los elementos del vector **excluir** si se encuentran en este.

```
void excluir(vector<Rect> *total, vector<Rect>*excluir)
```

En cuanto a las corcheras solitarias su detección se fundamentará en futuras ampliaciones del programa, como explicaremos en apartados posteriores(apartado 3.).

2.2.2.5.3. Filtrado silencios

La función *Filtrar_silencios* () se basa en una cualidad que comparten todas estas figuras: siempre tienen la misma posición dentro del pentagrama. Sabiendo esto, se ha desarrollado un método que consiste simplemente en eliminar aquellos patrones que no se encuentren dentro de uno de los pentagramas de la partitura.

```
void Filtrado_silencios (vector<Vec<int, 5>> *pentagramas,
                         vector<Rect> *silencios)
```

2.2.2.5.4. Filtrado blancas

En ocasiones cuando se detectan las blancas ocurre que alguna de las negras o corcheras de la partitura entra como falso positivo. Por esta razón, la función *Filtrar_blancas ()* usa el método 2 explicado anteriormente en el filtrado de negras (apartado 5.2.2.2.1. Filtrado negras/corcheras), pero a la inversa. Esta vez en vez de eliminar las detecciones que no tengan alguna zona con más de un 85% de negro, se eliminará la que las tienen, dejando dentro del vector solo las blancas.

```
void Filtrar_Blancas(vector<Rect> *blancas, Mat *bin, Mat *cdst)
```

2.2.2.5.5. Filtrado claves

Las claves (apartado 1.3 Teoría musical) son elementos que siempre ocupan la misma posición dentro del pentagrama y dentro de la partitura. Estos siempre están a la izquierda de todo, siendo la primera figura en todos los pentagramas o al menos en el primer pentagrama (depende de la partitura). Su filtrado es similar al de los silencios, sumándole además que sólo pueden estar en la parte más a la izquierda de la imagen. La función *Filtrar_claves* () se encarga de que todos los resultados de la búsqueda que no estén dentro de un pentagrama y en el primer 20% de columna de la imagen, sean eliminados.

```
void Filtrar_clave(Mat *cdst, vector<Rect> *clave)
```

2.2.2.5.6. Filtrado falsas figuras

La función *Falsas_figurar* () se encarga de un tipo de falsos positivos muy determinado. Elimina aquellas figuras que se han detectado encima de las claves (Clave de Sol, FA o Do), como consecuencia de la forma compleja que tienen estas. Para ello se comparará los rectángulos que contienen la/las claves con el resto de figuras, y si se encuentra alguna intersección entre los rectángulos se eliminará la figura correspondiente.

```
void Falsas_notas(vector<Rect> claves, vector<Rect> aux,
        vector<Vec<int, 5>> *vector_pentagramas)
```

2.2.2.5.7. Clasificación

Una vez filtrados todos los vectores de objetos Rect de las diferentes figuras se procede a clasificarlos. Con clasificar se quiere hacer referencia al hecho de asignar etiquetas para saber el tipo de figura que es, en que pentagrama está, que posición ocupa y a que nota hace referencia.

Para satisfacer estas necesidades se crea una nueva estructura de programación llamada figura. Esta constará de los siguientes atributos:

- TIPO: string que hace referencia al tipo de figura. Las palabras clave utilizadas son las siguientes:

Tipo de figura	Abreviatura usada
Negra	n
Blanca	b
Corchera	c
Redonda	r
Silencio de negra	sn
Silencio de blanca	sb
Silencio de corchera	sc
Silencio de redonda	sr
Clave	cl
Compás	cp

Tabla 3. Abreviaciones para el atributo "tipo"

- PENTAGRAMA: int que indica el pentagrama en que se encuentra la figura, empezando por el 1.

- POSICIÓN: Rect que guarda la posición de la figura.

- NOTA: se trata de un int que indica la nota que le corresponde a cada figura. Aquellas que no tienen nota, como los silencios, se les asigna el valor 0. El resto siguen el modelo que muestra la Figura 37.

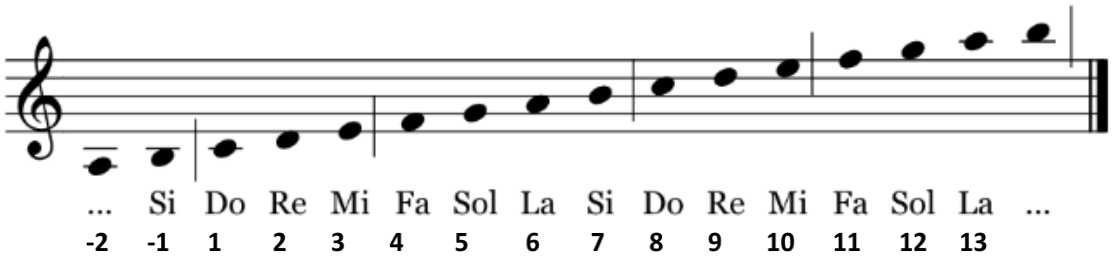


Figura 37. Números que se asignan a cada nota

Una vez creado la estructura con que guardar la información sólo falta la función que permite insertar todas estas características a cada figura:

```
Clasificardor_de_figuras (vector<struct figura> Final,
vector<Rect> figuras, vector<Vec<int,5>> vector pentagramas, int
altura, string tipo de figura)
```

- ***Final***: vector donde se guardarán las figuras clasificadas.
- ***Figuras***: vector de objetos Rect que le introduciremos para clasificar. Por ejemplo, el vector de blancas.
- ***Vector_pentagramas***: vector que contiene todas las líneas de todos los pentagramas de la partitura.
- ***Altura:*** altura del pentagrama.
- ***Tipo de figura***: aquí se le indicará el tipo de figura que contiene el vector *Figuras*. La función asignará a todos los elementos la misma etiqueta.

Lo más importante a tener en cuentas es que cada figura musical entra en la función como uno de los elementos del vector *Figuras* de objetos Rect. La función se encargará de crear a partir de cada Rect un objeto figura siguiendo estos pasos:

1. Empezará asignando el número de pentagrama que le corresponde a cada figura insertándolo en el atributo **pentagrama**. Lo hará dependiendo de la posición de la figura (que viene dada por el objeto Rect) con respecto a la línea central de cada pentagrama. La figura formará parte del pentagrama cuya línea central esté más cerca.

2. Para todas las figuras se asigna el valor 0 como predeterminado al atributo **nota**. La función que se encarga de atribuir las notas correspondientes se desarrollará más adelante.

3. A todos los elementos del vector de objetos Rect se les asignará el mismo atributo **tipo**, el cual se introduce como parámetro en la función (*tipo de figura*).

4. Finalmente, al atributo **posición** se le asigna el propio objeto Rect que representaba a cada figura antes de clasificarla.

Una vez la función ha finalizado su trabajo las figuras estarán clasificadas y guardadas en un vector que contendrá todas las figuras sean del tipo que sean. Con esto ya se puede dar paso a la siguiente fase, la asignación de notas.

En el Anexo I se puede observar el diagrama de flujo.

2.2.2.6. Asignación de notas

En esta fase del proyecto se tratará la asignación de la nota correspondiente a cada figura. La nota dependerá de la posición de la figura en relación a las líneas del pentagrama en el que se encuentre. Como se explica en la introducción teórica, el tipo de figura está relacionado con la durabilidad temporal del sonido, mientras que su posición en el pentagrama hace referencia a si es más aguda o más grave (indica la nota). Una vez tomada la decisión simplemente se accederá al atributo **"nota"** de la figura y se cambiará el valor predeterminado por el que le corresponde.

Esta fase la lleva a cabo la función *Decidir_Nota* () , que sigue esta estructura:

```
Void Decidir_Nota (vector<struct figura> *Final, vector<Vec<int, 5>>
        vector_pentagramas, int *altura, Mat bin, Mat *cdst);
```

- **Final**: vector donde se han guardado todas las figuras de todas las clases.
- ***Vector_pentagramas***: vector que contiene todas las líneas de todos los pentagramas de la partitura.
- ***Altura:*** altura del pentagrama.
- ***Bin:*** matriz binaria sin pentagramas (cdst_white).
- ***Cdst:*** matriz resultado, sonde se dibujan las notas.

Solamente hay 6 tipos de figuras que requieren nota: negra, blancas, redondas, corcheras, semicorcheas y fusas. Todas ellas seguirán el mismo proceso, el cual se puede dividir en dos partes principales:

1. En primer lugar, se debe de calcular en que parte del rectángulo que marca la posición de la figura se encuentra el punto que define la nota [Figura 38]. Este punto siempre corresponde al centro de la elipse que compone la figura.

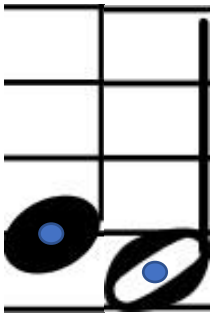


Figura 38. Punto que define la nota que se le asigna a la figura (en azul)

Para ello, se lleva a cabo un histograma dentro del rectángulo y se guardan en un vector las filas que tengan más de un 50 %.de sus pixeles de color negro (el recuento de pixeles siempre se hace en una matriz binaria). Como en la matriz de búsqueda se han borrado los pentagramas con anterioridad, la zona que más filas en negro tendrá será la zona donde se encuentra la elipse. De todas estas líneas que cumplen la condición se realiza la media. Con esta media se analiza de nuevo el conjunto de líneas que cumplen la condición del 50% y si hay alguna que se aleje más de lo que conviene (el espacio entre dos líneas de pentagrama como máximo) se elimina del grupo. Eliminados los datos irregulares el resultado se parecerá más a la realidad esperada[Figura 39]. Sólo queda por calcular por segunda vez la media y guardarla como la posición que definirá la nota que le corresponde a la figura, su *punto característico*.

Figura 39. Resultado de la primera(rojo) y la segunda(azul) media

2. Con la altura en el eje Y determinada (*punto característico*), ya se le puede asignar la nota. Para ello se compara este valor con los valores de las líneas del pentagrama en que se encuentra. El punto que definirá la nota para la figura puede estar encima de una de las líneas de los pentagramas o en el punto medio del espacio dos líneas consecutivas. Además, esta norma se extiende arriba y debajo del pentagrama, hay que imaginarse que el pentagrama continúa en estas zonas y se rige por las mismas reglas, como si ahí también hubiese líneas. Esta segunda etapa compara el punto característico de la figura con todas las posibles posiciones que se muestran en la Figura 40. Aquella a la que más se acerque será la que se le asigne.

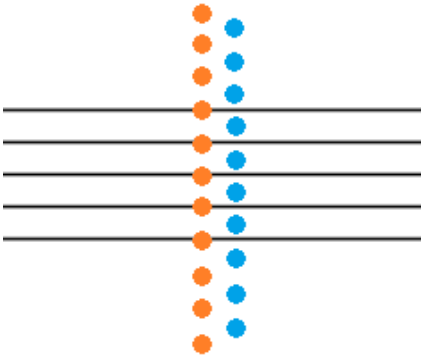


Figura 40. Posibles posiciones en las que encontrar una figura (tanto azules como naranjas)

Así termina la fase de asignación de notas, con un vector llenado con todos los elementos de la partitura, con sus particulares características y con una posición única para cada uno [Figura 41].

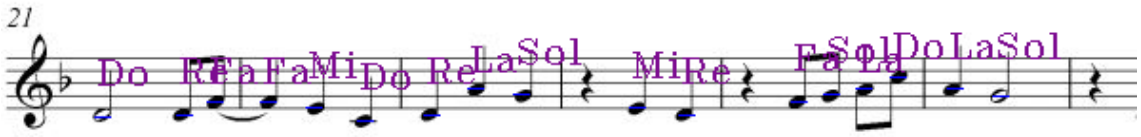


Figura 41. Ejemplo de figuras con sus correspondientes notas

Ya solo queda reproducir todos los datos recogidos para ver con cuerpo el resultado de todo el trabajo. En el Anexo I se puede observar el diagrama de flujo.

2.2.2.7. Reproducción

La última fase del programa se encargará sólo de acceder a la información de cada figura y efectuar unas acciones dependiendo de esta. Para ello se ha creado la función *Reproducir ()*:

```
void Reproducir (int tempo, vector<struct figura> Final, Mat cdst,
                 vector<Vec<int, 5>> vector_pentagramas)
```

- Tempo: es la velocidad con la que se reproducen la melodía (ver apartado Teoría Musical).
- Final: vector que contiene todas las figuras.
- Cdst: matriz que guarda la imagen de la partitura.
- Vector_pentagramas: contiene las líneas de todos los pentagramas.

Esta función en su interior consta de otras funciones dadas por la librería *MMSystem* que permiten reproducir archivos .wav [22]. Asignando a cada nota un archivo de sonido se consigue reproducir todas las notas de la escala. Estas tienen la siguiente estructura, donde *time* es el número de milisegundos que queremos que se reproduzca:

```
void PlaySound (string direccion, int modo);
```

- Dirección: dirección en memoria donde se encuentra el archivo de sonido
- Modo: modo de reproducción (Sincrono, asíncrono..)

Por tanto, la función *Reproducir ()* seguirá estos pasos:

1. En primer lugar, hay que reconvertir el tempo que le introducimos a milisegundos. Para ello usamos el siguiente factor de conversión:

$$milisegundos = \frac{60\ x\ 1000}{tempo}$$

2. A continuación, se ordenan todas las figuras del vector para que la primera sea la figura del primer pentagrama que esté más a la izquierda, y la última la que esté en el último pentagrama más a la derecha. Básicamente siguiendo su orden de reproducción según se rige en lenguaje musical.

3. Una vez ordenadas, una a una se mirará el tipo de figura que es, desde la primera hasta la última, y se llevaran a cabo las siguientes acciones:

    a. Se le asignará el tiempo de reproducción correspondiente a cada figura (revisar apartado Teoría Musical). Por ejemplo, la negra y el silencio de negra tienen la misma durabilidad, y la blanca y su silencio también comparten misma durabilidad, el doble que la de la negra.

    b. Se dibujará en torno a la figura un rectángulo cuyo color depende del tipo de figura, como se ve en la Figura 42.

    c. Si tiene una nota asignada se escribirá encima del rectángulo como se ve en la Figura 34.

    d. Se reproducirá el sonido correspondiente a la nota durante el tiempo asignado en el apartado a).

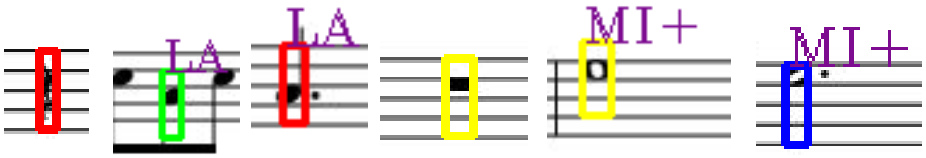


Figura 42. Figuras en fase de Reproducción.

    e. De figura a figura la matriz donde se dibujan los rectángulos y las notas será reseteada, ya que se desea que sólo esté dibujada la figura que se reproduce en el momento.

    f. Si la figura en una clave o un cualquier elemento sin sonido no se realizará ninguna de las acciones anteriores. Simplemente se pasará a la siguiente.

Y con esto concluye la fase y el proyecto. En el Anexo I se puede observar el diagrama de flujo.

En el siguiente apartado se explicarán algunos procesos que no se han llevado a cabo por falta de tiempo pero que se guardan para futuras ampliaciones.

3. Futuras ampliaciones del proyecto

Por distintas razones no se ha podido profundizar más en el proyecto, por eso en esta sección se explicarán procesos que no han podido llegar a programarse y que se guardan para futuras ampliaciones.

No se ha llegado a detectar las corcheras sueltas (apartado 5.2.2.2.2. Separación Negras y Corcheras) debido a su parecido con las negras y a que no constan de la barra negra característica de los grupos de corcheras. Estas figuras constan de un pequeño brazo como se puede apreciar en la Figura 23 el cual puede gastarse como característica para detectarlas si se implementara un filtrado.

Las semicorcheas y fusas también han quedado fuera del proyecto, pero detectarlas no supondrá mucho problema en un futuro. Usando la misma técnica que para los grupos de corcheras (apartado 5.2.2.2.2. Separación Negras y Corcheras) solo se deberá de buscar una doble o triple línea en vez de una línea simple.

Los puntos que a veces acompañan las figuras para incrementar su duración (apartado 1.3. Teoría musical) serán más difíciles. Usando plantillas se obtienen demasiados falsos positivos, ya que solo se trata de un punto. En este caso el mejor método será buscar alrededor cada una de las figuras este punto tan característico, algo parecido a la búsqueda de la barra de las corcheras (apartado 5.2.2.2.2. Separación Negras y Corcheras).

En futuras ampliaciones un aspecto importante sería detectar las repeticiones y añadirlas a la reproducción de la melodía. A nivel de programación detectarlas no supone un problema ya que siempre mantienen la misma posición dentro del pentagrama. Programar el efecto de la repetición será más laborioso ya que deberemos seleccionar las figuras que se deben repetir e introducirlas dobladas en el vector para conseguir el efecto de repetición.

Para conseguir una detección completa no se deberá obviar ningún elemento: ligaduras, armónicos, matices del sonido, alteraciones… La mayoría se podrán detectar con plantillas, pero también se deberán de crear funciones para introducir sus efectos sobre la melodía.

Una vez implementados todas las ampliaciones a nivel de detección el siguiente paso será transformar el proyecto en una aplicación Android que este al acceso de toda persona interesada. Simplemente con una foto de la cámara del teléfono se conseguiría reproducir cualquier partitura. Y ya no sólo eso, también una copia digital en pdf que pueda incluso ser modificada

desde el móvil a gusto del usuario. Con esto se creará una herramienta que satisfaga a músicos, compositores y curiosos del mundo.

4. Conclusiones

El aprendizaje de las librerías de OpenCV ha conllevado una inversión temporal importante en el desarrollo de este proyecto. Después de tanto tiempo centrado en su desarrollo se puede decir que:

- Se ha conseguido encontrar un método para identificar muchos de los tipos de figuras musicales que existen, pero todavía no se ha llegado a diferenciarlos todos. Se deja para futuras ampliaciones esta parte.

- Se perseguía un modelo de proyecto que crease métodos de filtrado sin error. Se han desarrollado muchos, pero no se ha podido eliminar por completo todos los falsos positivos.

- Crear un programa capaz de leer cualquier partitura ere una de las pautas que a medida que avanzaba se iba dificultando cada vez más y más. En múltiples ocasiones se han reescrito los métodos de filtrado para intentar generalizarlos, pero siempre hay alguna partitura que rompe la regla.

- El método de búsqueda de patrones ha dado unos resultados realmente positivos, y se ve que con una base de datos de plantillas mucho más elaboradas el resultado podría llegar a tener una precisión cercana al 100%.

- El uso de plantillas ha conllevado muchos errores y estas se han ido cambiando y modificando a lo largo de la implementación para incrementar el número de detecciones correctas. El un juego de plantillas final con el que funciona el programa supera las expectativas, ya que con pocas se ha conseguido detectar una gran cantidad de elementos diferentes.

En cuanto a temas más generales:

- El tiempo es un factor que ha influenciado en la decisión de finalizar el proyecto. Los resultados habrían sido más impresionantes si se hubiese hecho en grupo. Individualmente uno se carga mentalmente si debe crear y probar todos los métodos posibles para filtrar resultados.

- En el apartado del estado del arte ya se ha comentado que esta tecnología ya ha sido desarrollada por algunos programadores con muy buenos resultados. Por esto su valor en mercado baja, ya que no se trata de una.

- El poder aprender unas buenas bases de visión artificial llena a cualquier mente curiosa, y más si puedes juntarlo con la música. Se echa en falta haber dado el tema a lo largo del grado para haber aprovechado mejor el tiempo de desarrollo.

5. Bibliografía

[1] Jose Alexander Gomez. https://queesela.net/lenguaje-musical/ [Consulta:7/9/18]

[2] Sergio Sierra. <https://es.slideshare.net/sssierra/lenguaje-musical?qid=2cf2ed72-398a-4ded-b8a1-4a7f57e8e912&v=&b=&from_search=2> [Consulta:7/9/18]

[3]Pau González (2016) "Lenguaje musical:¿Qué es y cuál es su importancia?" en *Guioteca* <https://www.guioteca.com/educacion-para-ninos/lenguaje-musical-que-es-y-cual-es-su-importancia/ >[Consulta:1/9/18]

[4]Tempo. <https://es.wikipedia.org/wiki/Tempo>[Consulta:9/18]

[5] German Bardina. <http://musicouniversal.blogspot.com/2009/04/que-es-el-compas.html>> [Consulta:9/18]

[6]Benlloch, J. V., Agusti, M., Sanchez, A., & Rodas, A. (1995, October). Colour segmentation techniques for detecting weed patches in cereal crops. In Proc. of Fourth Workshop on Robotics in Agriculture and the Food-Industry (pp. 30-31). [Consulta:9/18]

[7]Sánchez, A. J., Albarracin, W., Grau, R., Ricolfe, C., & Barat, J. M. (2008). Control of ham salting by using image segmentation. Food Control, 19(2), 135-142. [Consulta:9/18]

[8] Bosch-Jorge, M., Sánchez-Salmerón, A.-J., Valera, Á., & Ricolfe-Viala, C. (2014). Fall detection based on the gravity vector using a wide-angle camera. Expert Systems with Applications, 41(17).<https://doi.org/10.1016/j.eswa.2014.06.045>[Consulta:9/18]

[9] Bosch-Jorge, M., Sánchez-Salmerón, A.-J., & Ricolfe-Viala, C. (2012). Visual-based human action recognition on smart phones based on 2D and 3D descriptors. International Journal of Pattern Recognition and Artificial Intelligence, 26(8). https://doi.org/10.1142/S0218001412600099[Consulta:9/18]

[10]Martinez-Berti, E., Sánchez-Salmerón, A. J., & Ricolfe-Viala, C. (2017). Dual quaternions as constraints in 4d-dpm models for pose estimation. Sensors (Switzerland). [Consulta:9/18]

[11]Imagen. < https://www.escribircanciones.com.ar/teoria-musical/4440-escalas-musicales.html> [Consulta:9/18]

[12] Jay O. <https://www.androidpit.es/snapnplay-music-musica-maestro>[Consulta:9/18]

[13]PhotoScore. < https://www.neuratron.com/photoscore.htm>[Consulta:9/18]

[14]Google Ayuda. <ttps://support.google.com/translate/answer/6142483?co=GENIE.Platform%3DAndroid&hl=es> [Consulta:9/18]

[15] Ruoff R. (2015-2016). Aplicación para la obtención de un mapa 3D de una escena con un dispositivo Android. Trabajo Final de Carrera.

https://riunet.upv.es/bitstream/handle/10251/68606/RUOFF%20-%20Aplicaci%C3%B3n%20para%20la%20obtenci%C3%B3n%20de%20un%20mapa%203D%20de%20una%20escena%20con%20un%20dispositivo%20Android.pdf?sequence=1&isAllowed=y [Consulta:9/18]

[16] Visual Studio.< https://visualstudio.microsoft.com/es/> [Consulta:7/18]

[17]Visual Studio – Wikipedia. https://es.wikipedia.org/wiki/Microsoft_Visual_Studio.[Consulta:9/18]

[18] Sergio Figueroa Sánchez (4/7/2016). http://200.10.19.208/REPOS/COMPUTERVISION/Semana07.html. [Consulta:9/18]

[19] Carmelo M.A. (2016)"OpenCV Búsqueda de Patrones (Template Matching)" en *Tutor de programación* 4 de junio. < http://acodigo.blogspot.com/2016/06/template-matching.html> .

[20] Carmelo M.A. (2017)"Conociendo la clase cv::Mat de OpenCV" en *Tutor de programación* 29 de abril. < http://acodigo.blogspot.com/2017/04/conociendo-la-clase-cvmat-de-opencv.html>[Consulta:7/18]

[21] Documentación OpenCV versión 3.1.0. < https://docs.opencv.org/3.1.0/>[Consulta:7/18]
[22]Vector. < http://www.cplusplus.com/reference/vector/vector/>[Consulta:7/18]

[23]Como escribir un TFG y en concreto uno de informática. http://users.dsic.upv.es/~fjabad/pfc/comoEscribir.pdf[Consulta:9/18]

[24]Korud (septiembre, 2017)." Instalar OpenCV 3.1 y Visual Studio 2015" en *Korud apuntes de un autodidacta*. http://korud.com/blog/instalar-opencv-3-1-y-visual-studio-2015/ [Consulta:]

6. Anexos

6.1. Diagramas de flujo

6.1.1. Diagrama Fase 1 – Selección de partitura

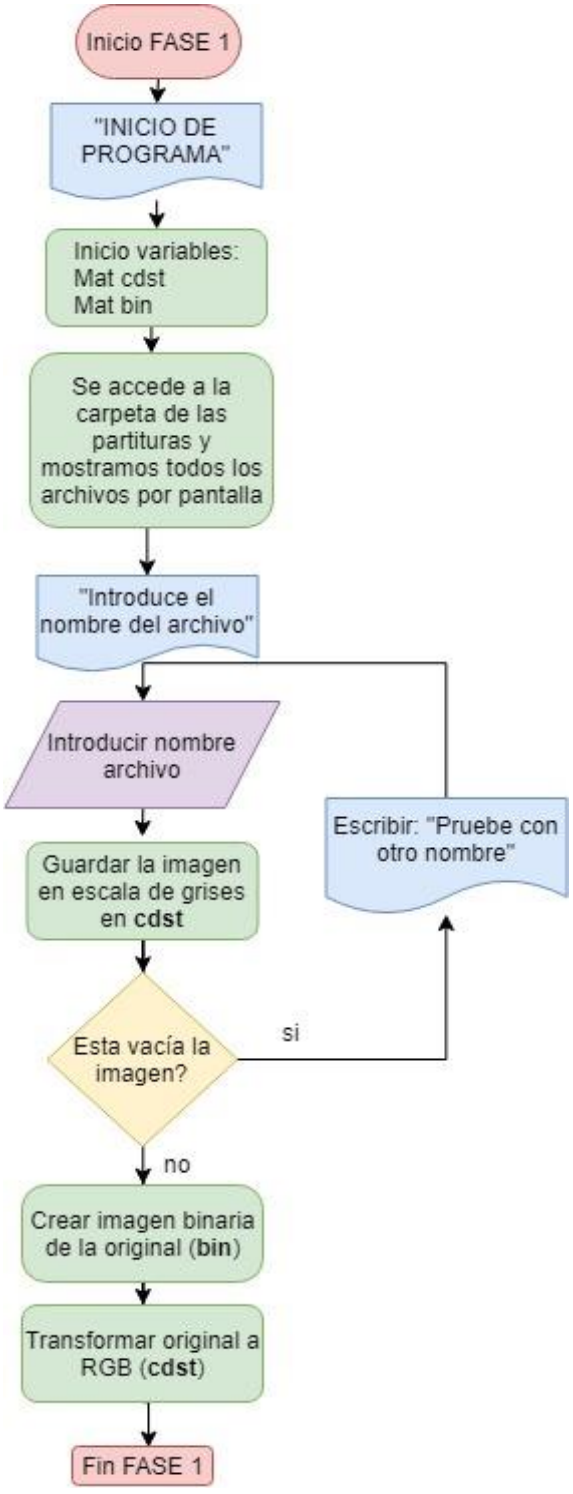


Figura 43. Flujograma 1.

6.1.2. Diagrama Fase 2 – Detección de pentagramas

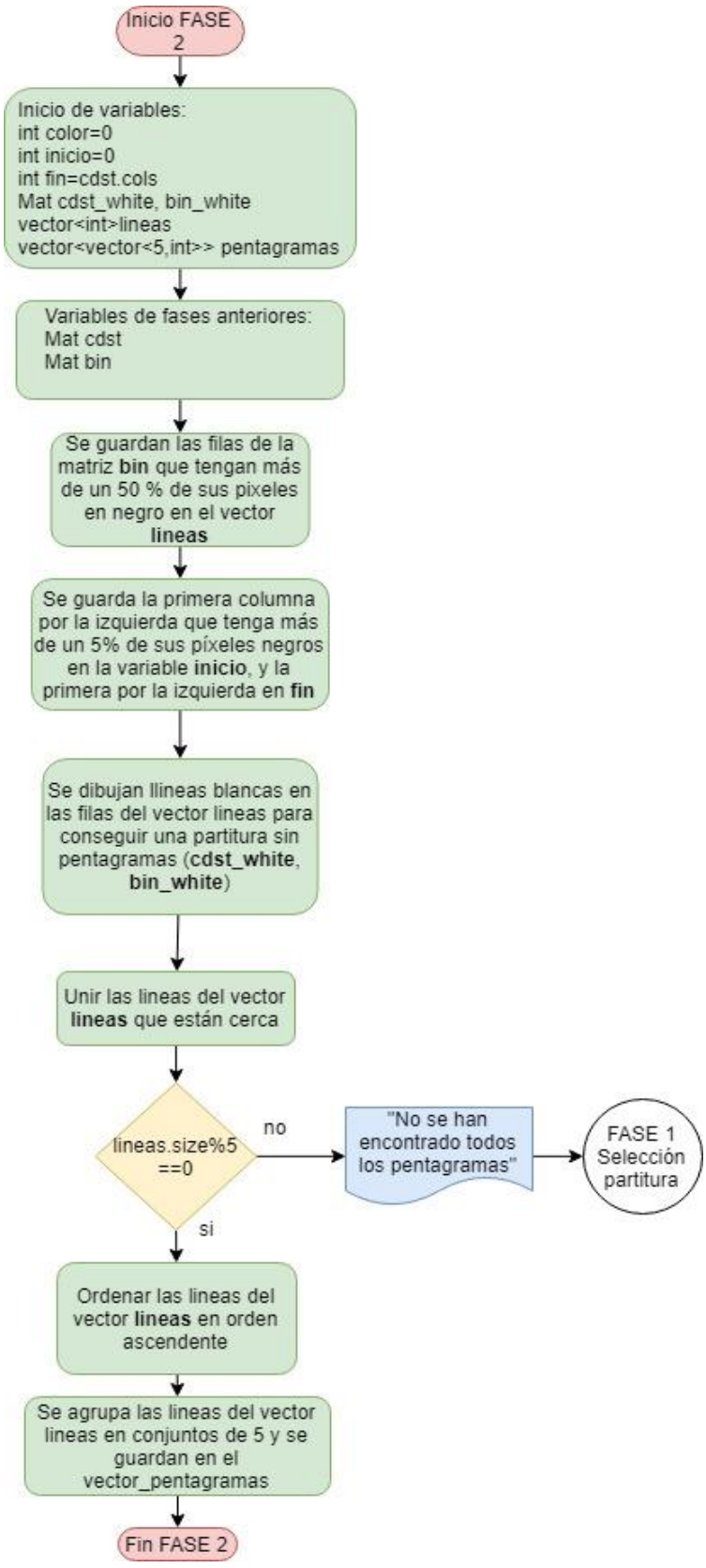


Figura 44. Flujograma 2.

6.1.3. Diagrama Fase 3– Búsqueda de figuras

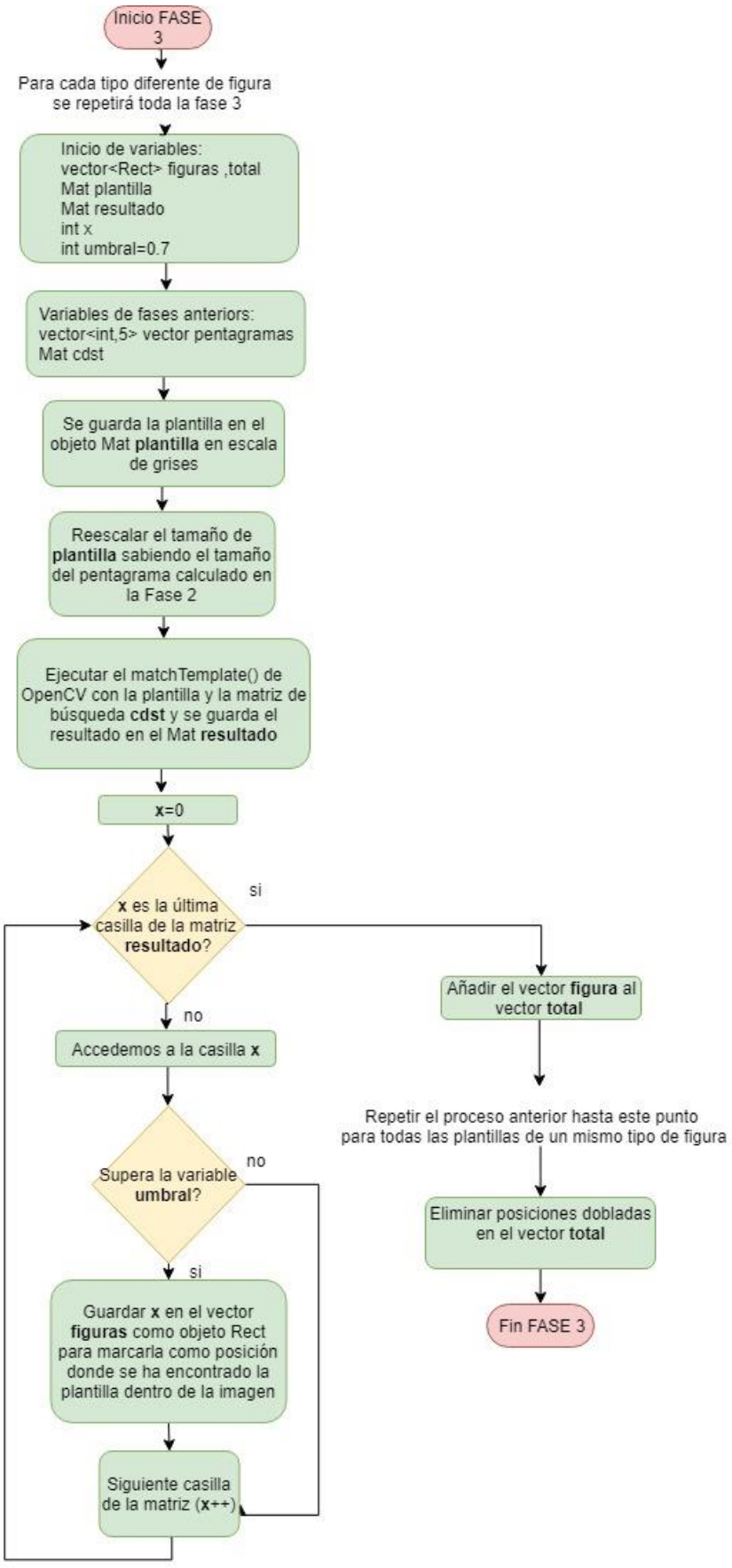


Figura 45. Flujograma 3

6.1.4. Diagrama Fase 4 – Filtrado y clasificación

Figura 46. Flujograma 4

6.1.4.1. Filtrado de negras 1

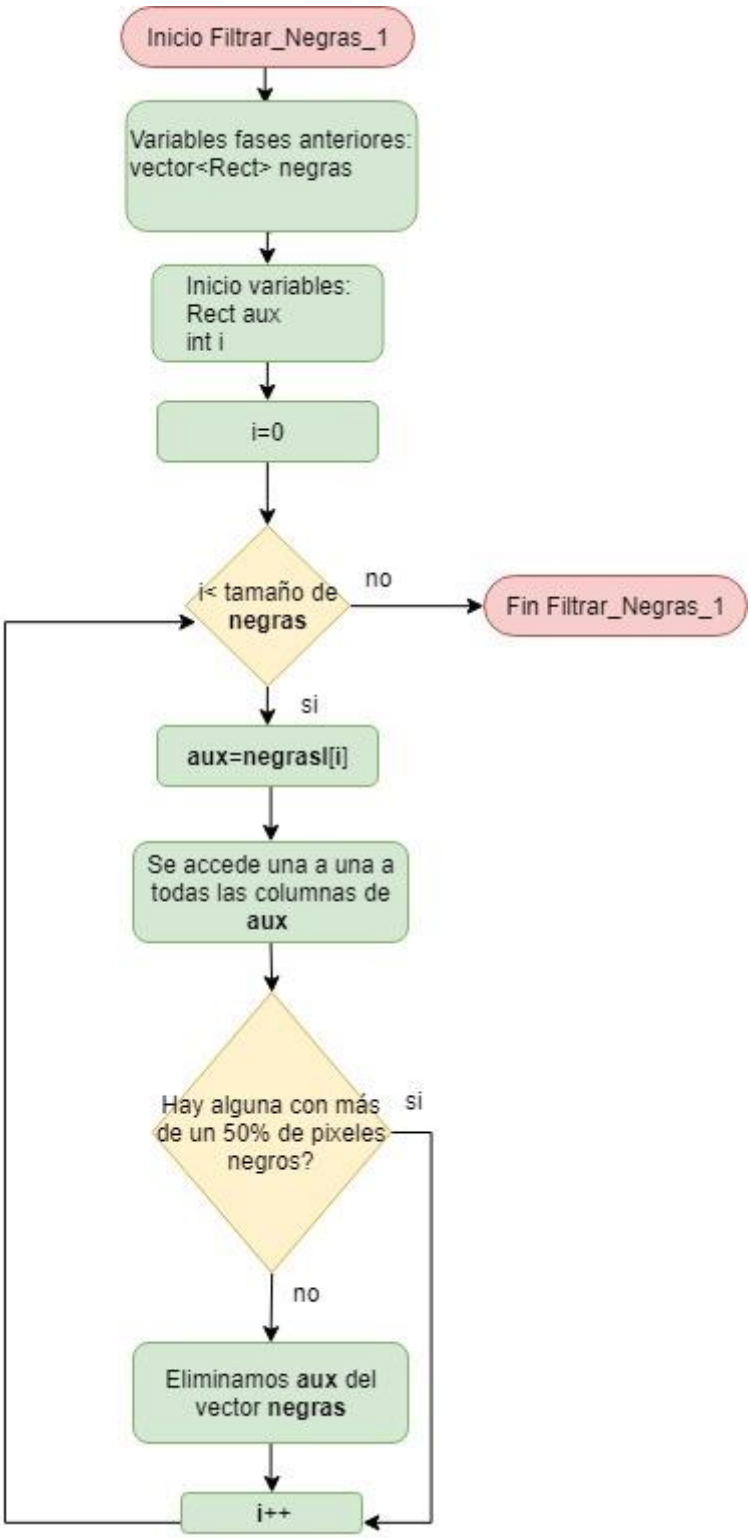


Figura 47. Flujograma 5

6.1.4.2. Filtrado de negras 2

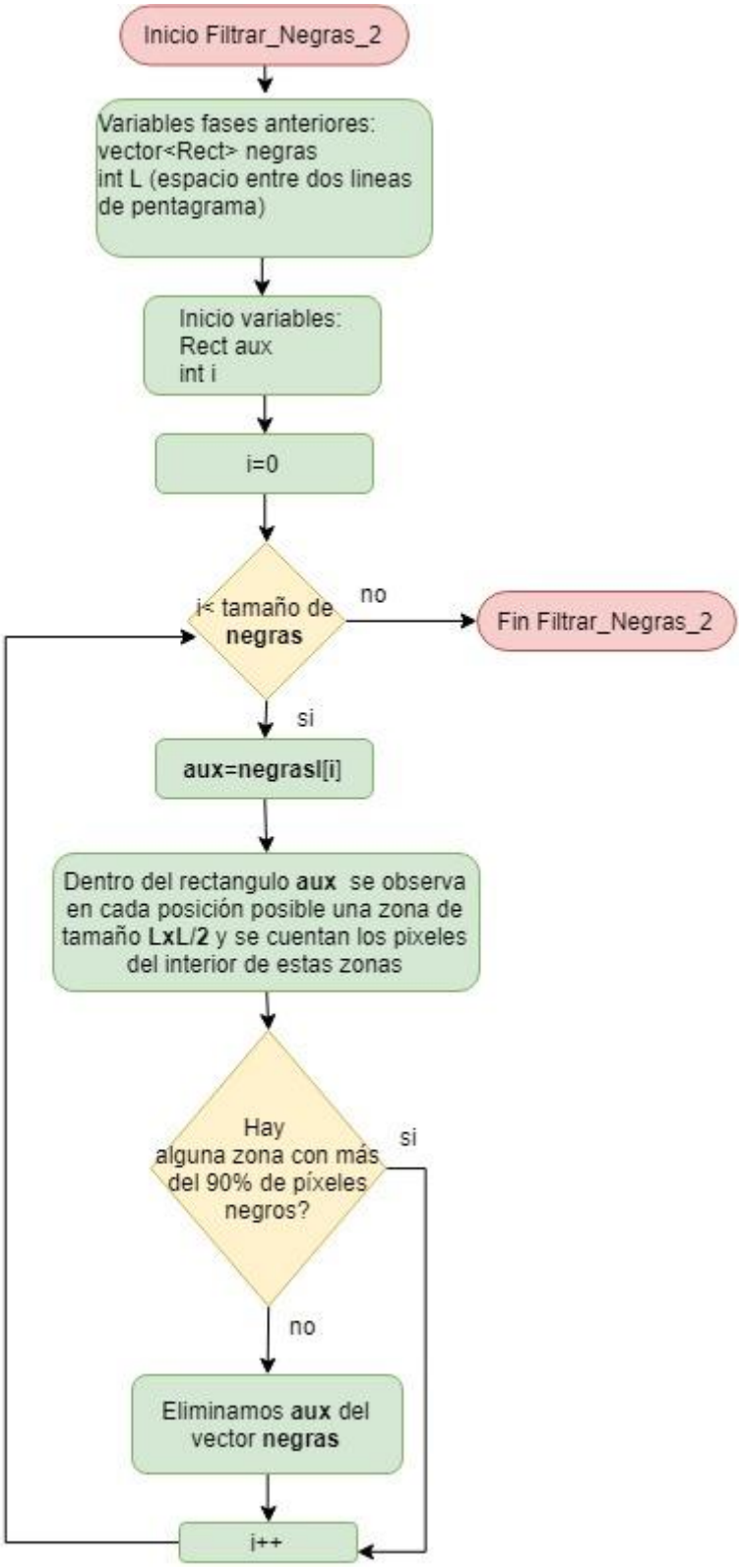


Figura 48. Flujograma 6

6.1.4.3. Filtrado de negras 3

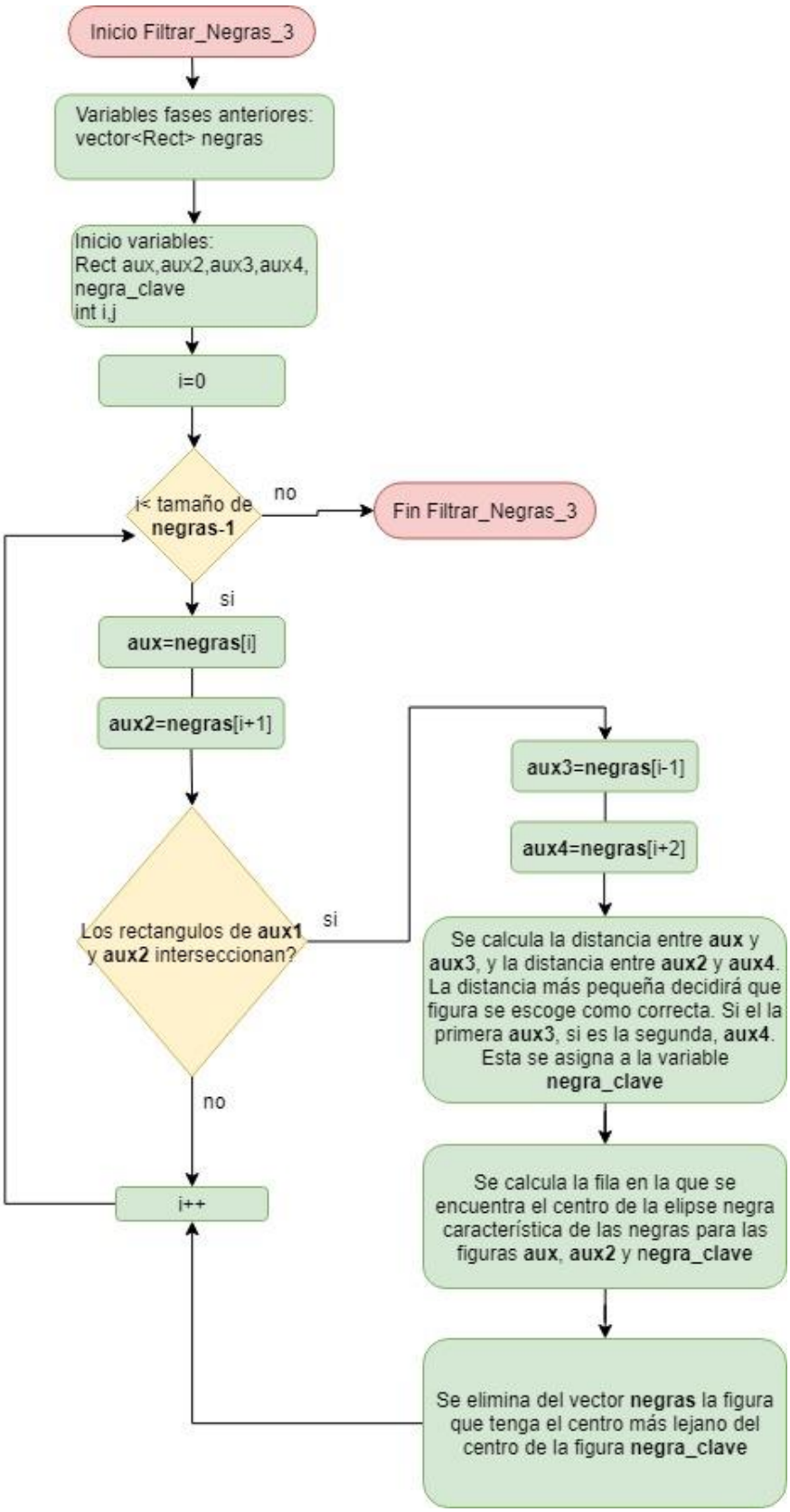


Figura 49. Flujograma 7

6.1.4.4. Separación negras corcheras

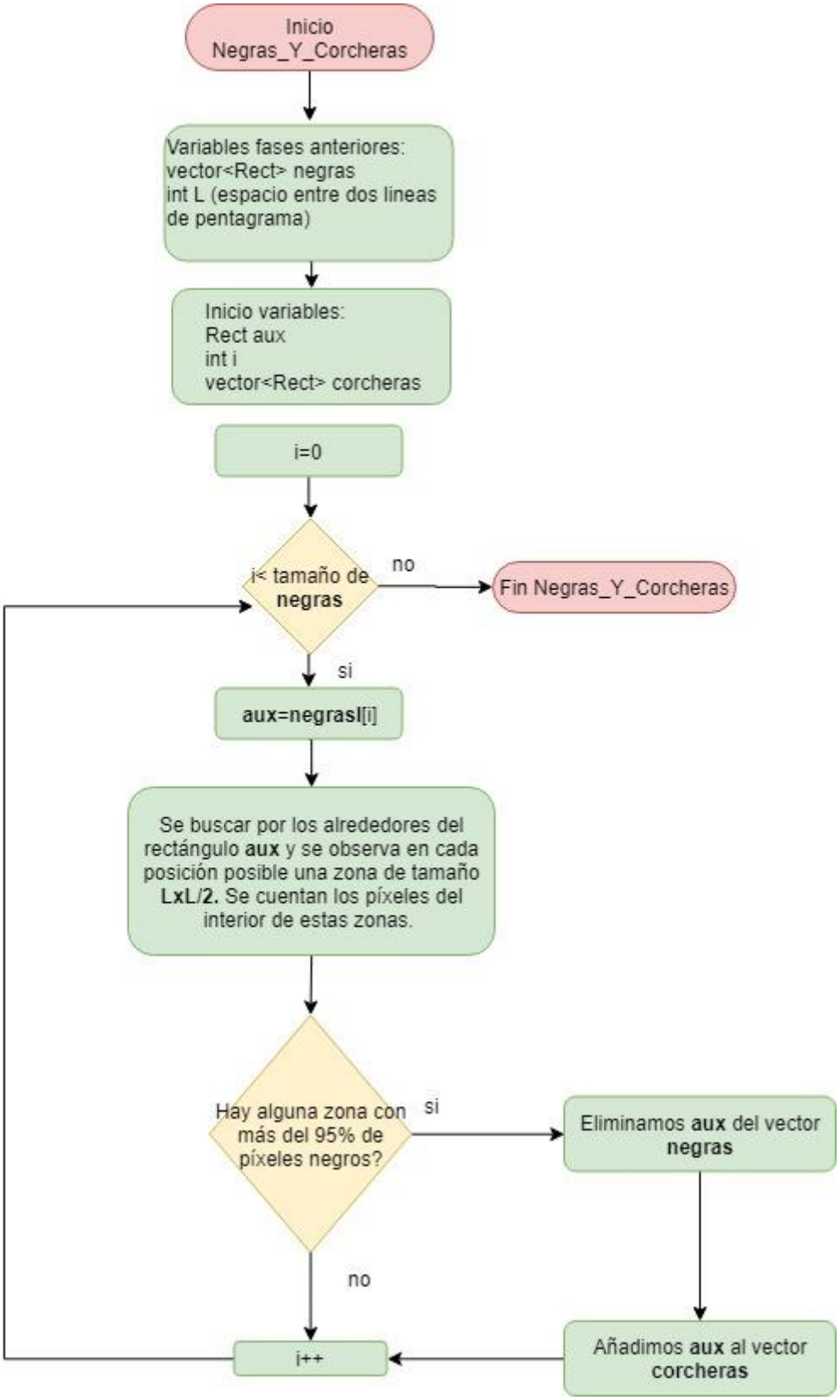


Figura 50. Flujograma 8

6.1.4.5. Filtrado de blancas

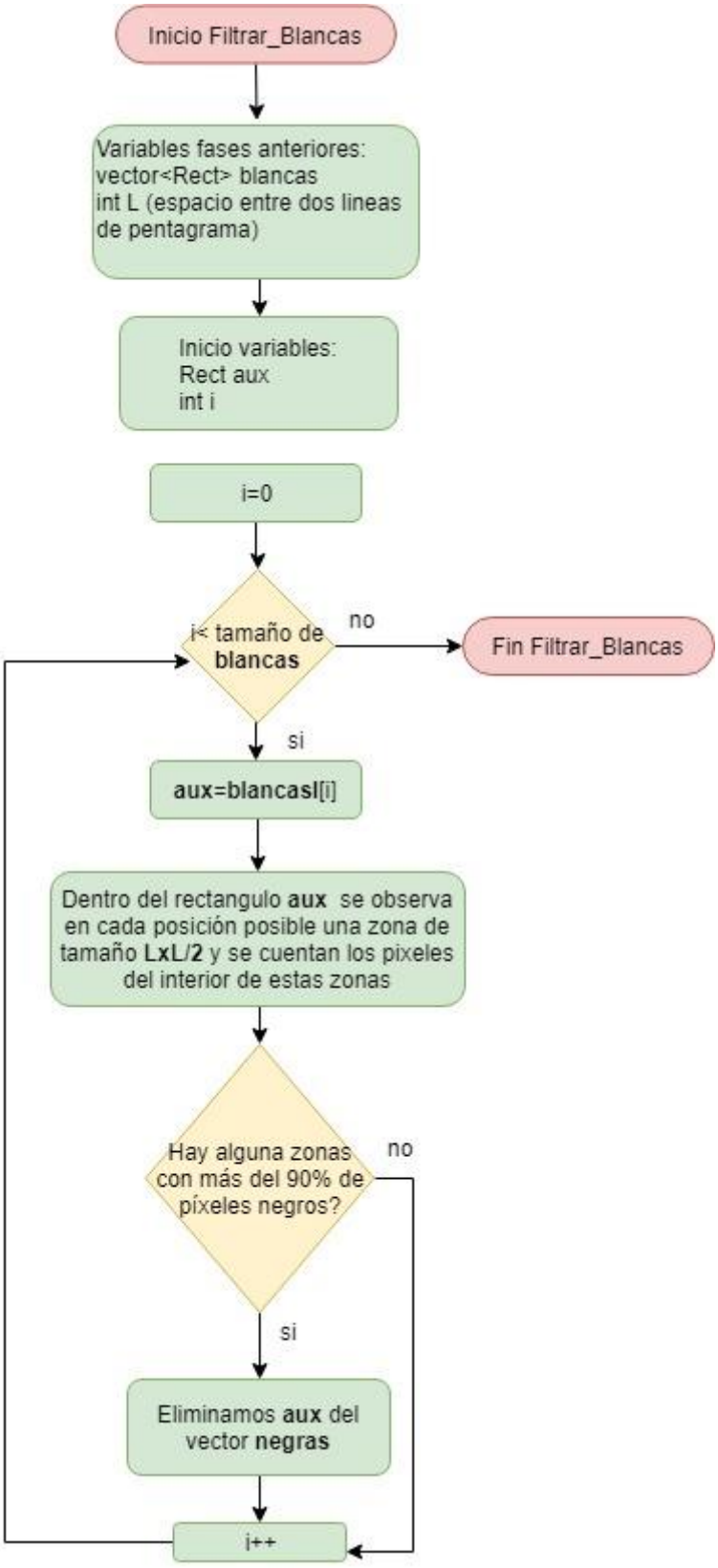


Figura 51. Flujograma 9

6.1.4.6. Filtrado de silencios

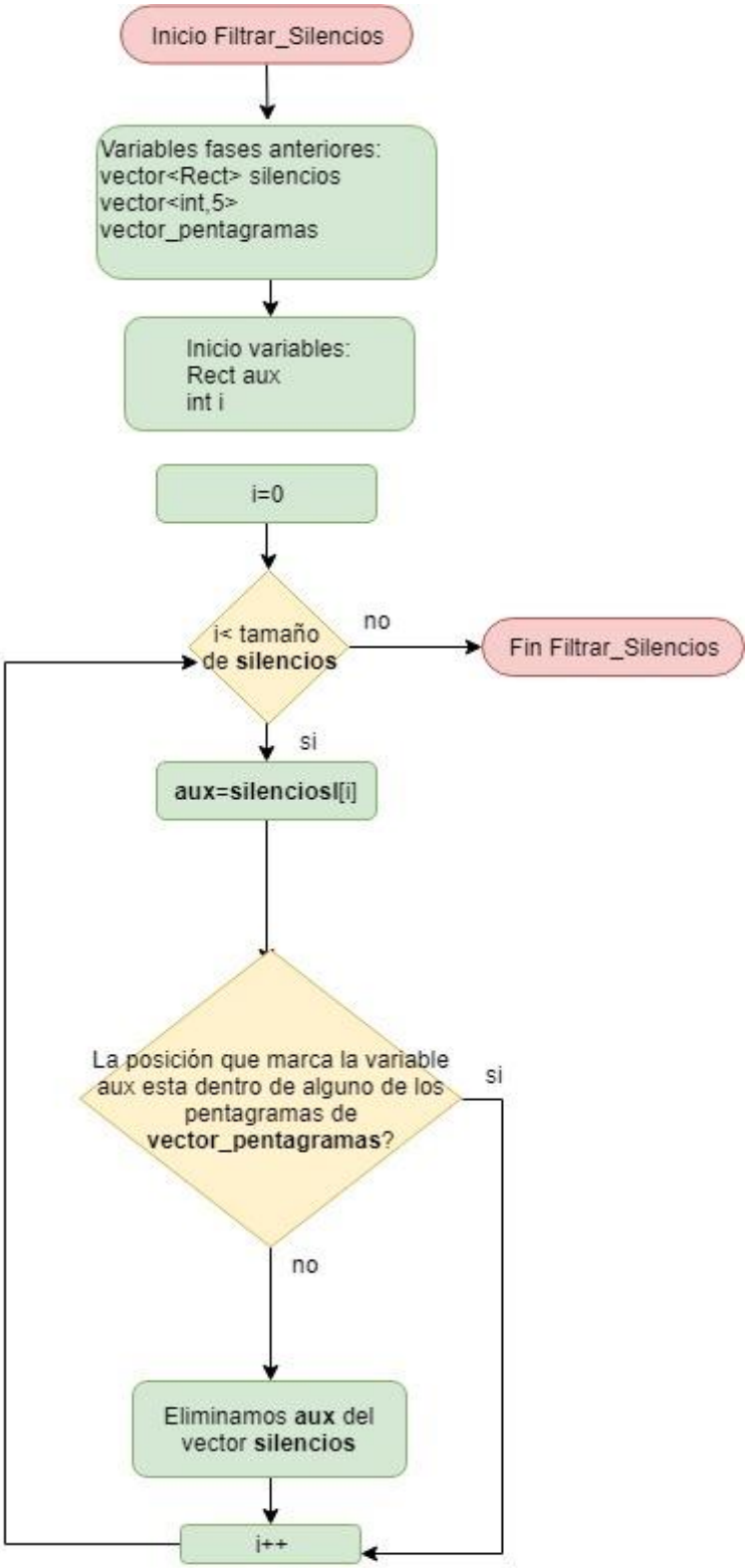


Figura 52. Flujograma 10

6.1.4.7. Filtrado de claves

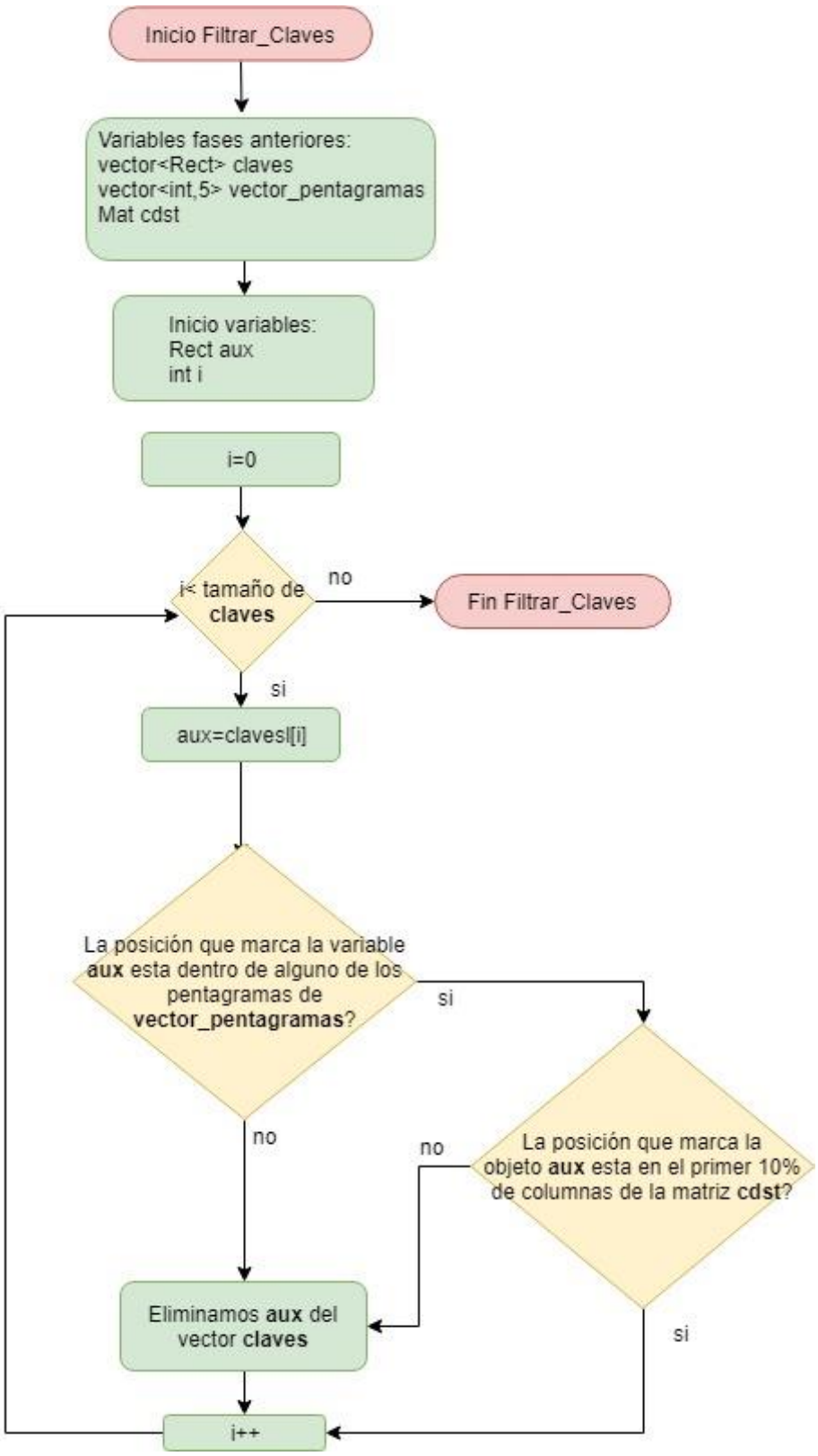


Figura 53. Flujograma 11

6.1.4.8. Falsas notas

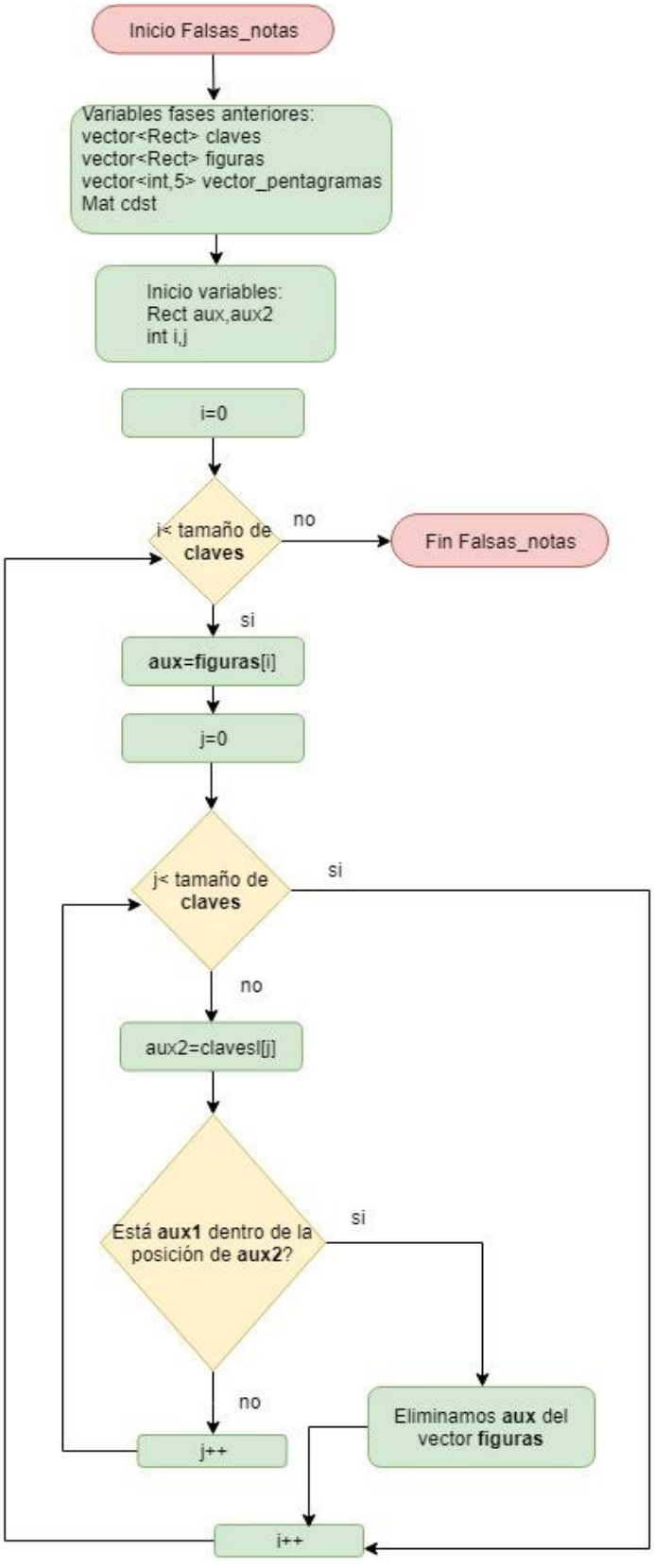


Figura 54. Flujograma 12

6.1.4.9. Clasificación

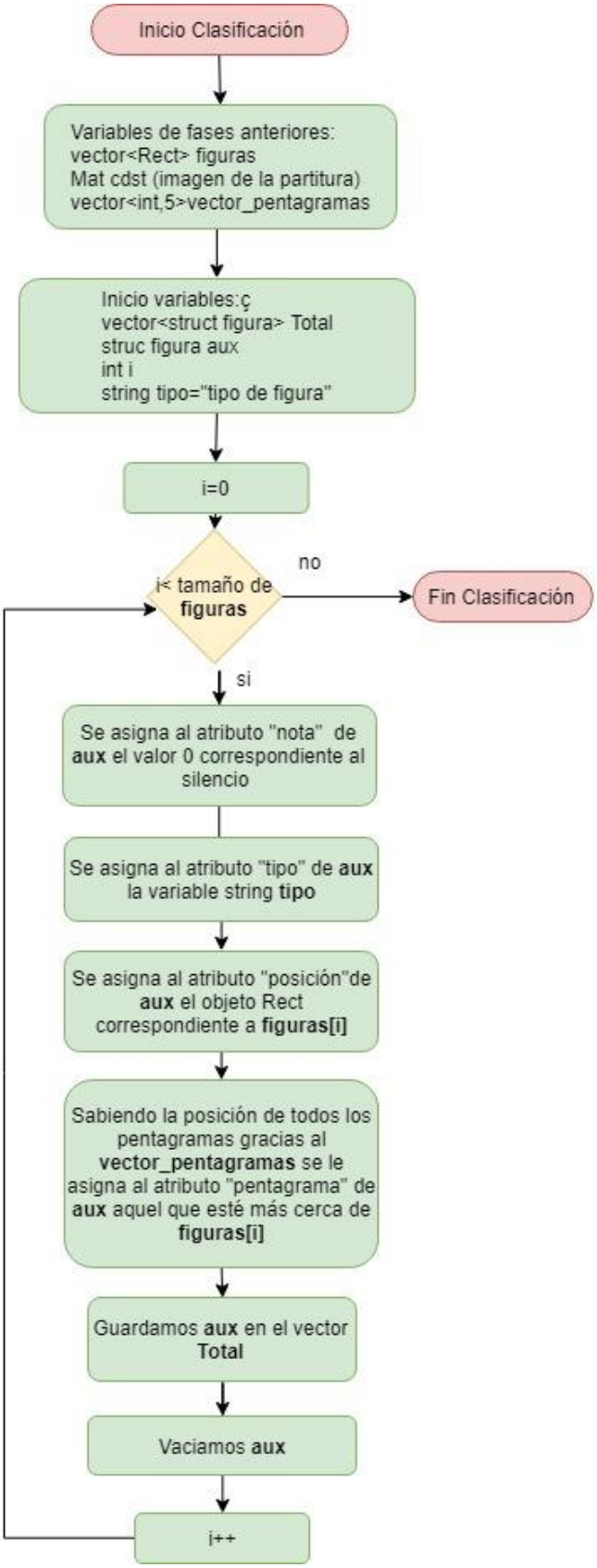


Figura 55. Flujograma 13

6.1.5. Diagrama Fase 5 – Asignación de notas

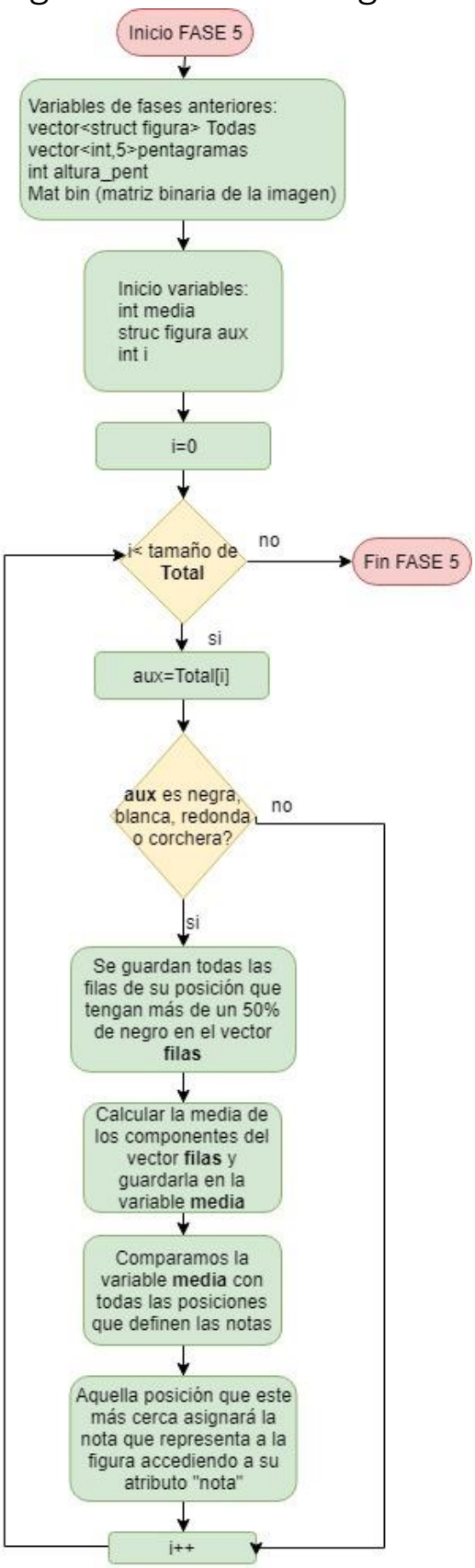


Figura 56. Flujograma 14

6.1.6. Diagrama Fase 6 – Reproducción

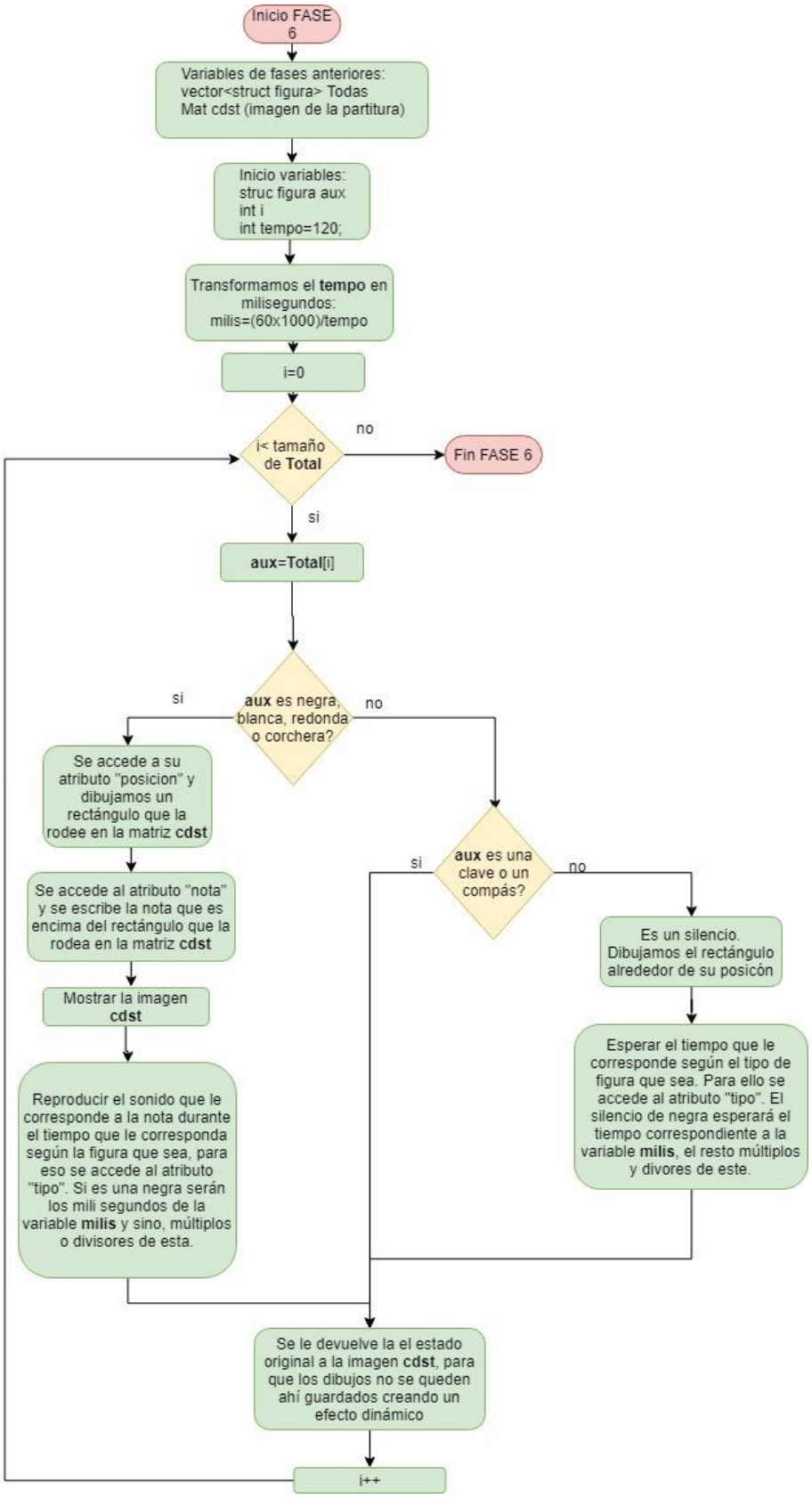


Figura 57. Flujograma 15

6.2. Instalación de VS2015 y las librerías OpenCV 3.1

Tutorial extraído de la referencia bibliográfica número 24.

Lo primero es instalar Visual Studio 2015, es de Microsoft y es gratis:

https://www.visualstudio.com/es-es/downloads/download-visual-studio-vs.aspx

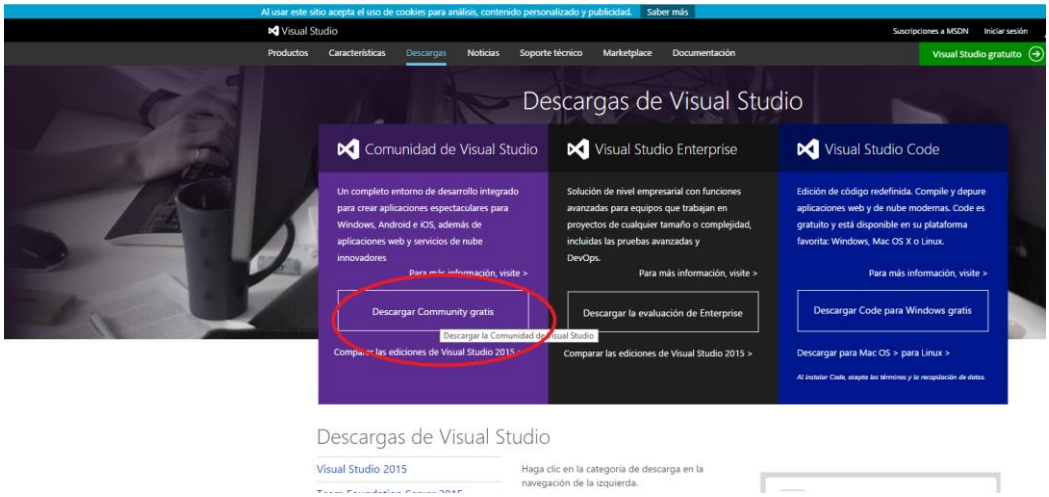


Figura 58. Instalación VS2015– Captura 1

Se descarga el archivo, se ejecuta y se sigue el proceso de instalación el proceso de instalación

Se descargan los archivos de OpenCV 3.1 of Windows desde su página oficial: http://opencv.org/ es la más reciente que podremos encontrar ahora mismo.

Se ejecuta el .exe que se ha descargado, lo que hace es descomprimir los archivos en una carpeta con el nombre opencv en la ruta que seleccionada, lo más recomendable es hacerlo en c:/ porque habrá que configurar la ruta a las carpetas en Visual Studio. Quedaria c:/opencv.

Para la configuración se deberá tener en cuenta las etiquetas de cada versión del programa: vc9 = Visual Studio 2008, vc10 = Visual Studio 2010, vc12 = Visual Studio 2013, vc14 = Visual Studio 2015. En este tutorial al querer instalar Opencv sobre Visual Studio 2015 se necesitarán los archivos de la carpeta vc14 que se verá más adelante más adelante.

Configuramos el patch de Windows: panel de control/sistema/configuracion avanzada del sistema y variables de entorno. Se pulsa en patch y en editar:

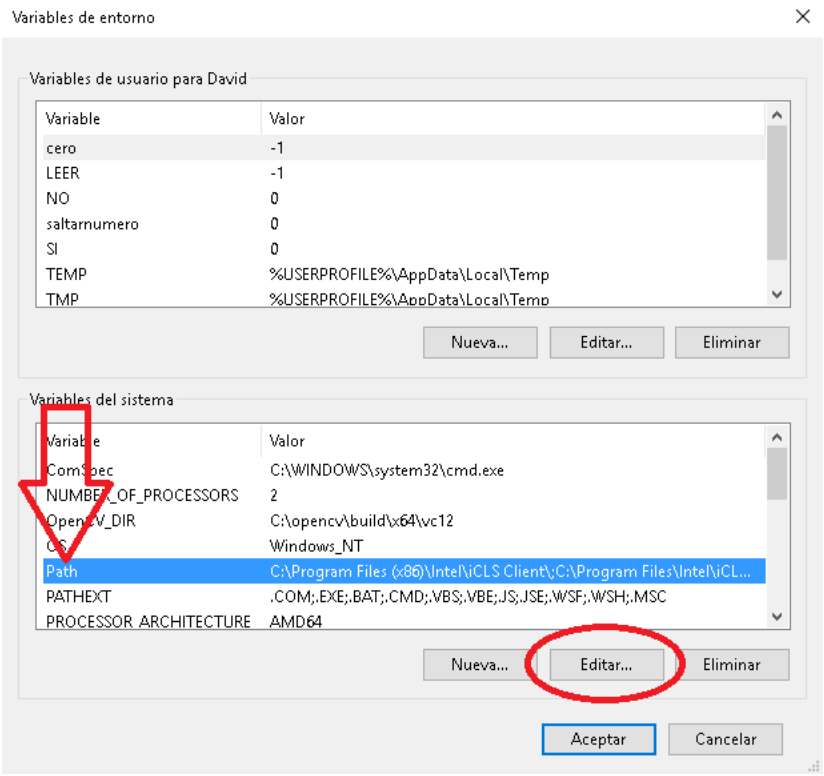


Figura 59. Instalación OpenCV – Captura 1

Se hace clic en nuevo y se pone :

```
C:\opencv\build\x64\vc14\bin
```

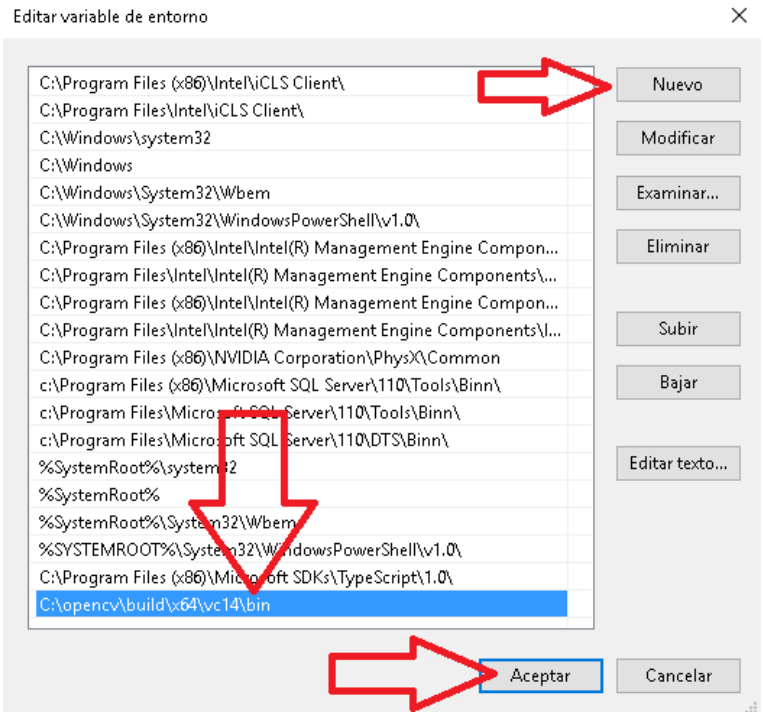


Figura 60. Instalación OpenCV – Captura 2

Luego se hace clic en aceptar/aceptar/aceptar y se sale del panel de control.

Ahora en Visual Studio 2015 se instalan las opciones para poder crear proyectos en C++ desde consola win32:

- Iniciar Visual Studio 2015, y se accede a file/New/project.
- En el desplegable de la izquierda clic en Templates/Other
- Languages/Visual C++ y en la derecha Install Visual C++ 2015
- Tools for Windows Desktop y clic en ok.

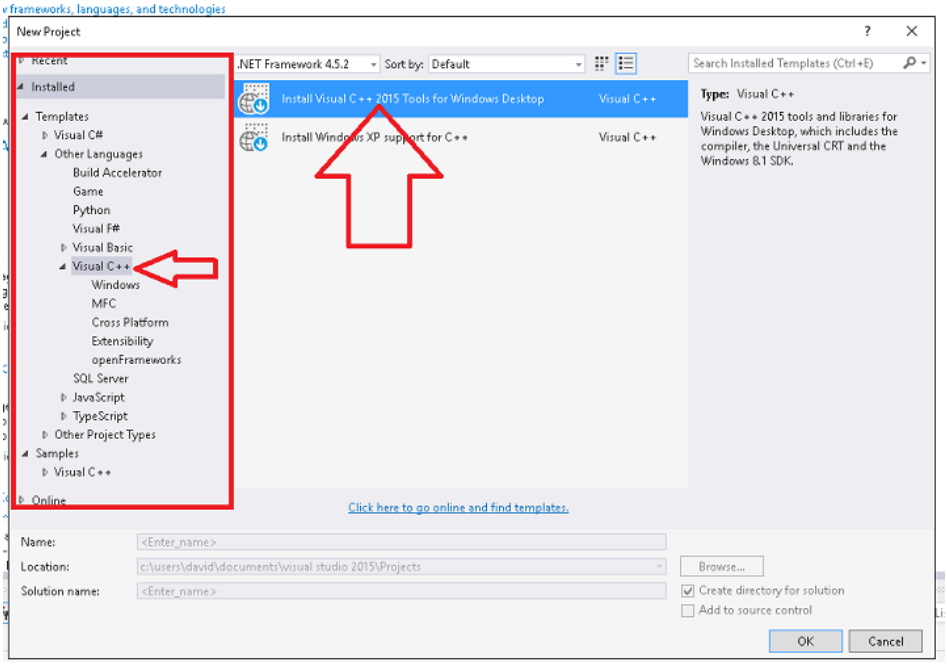


Figura 61. Instalación OpenCV – Captura 3

Aparece una ventana nueva y se hace clic en Install. Saldrá otra nueva ventana que pedirá cerrar Visual Studio, se cierra y clic en Retry, luego en Next, y luego en UPDATE. Empieza la instalación y cuando termine se hace clic en Close.

Ahora cuando se cree un nuevo proyecto de C++ siguiendo los pasos igual que antes, se tiene la opción de crear un proyecto Win32 Console Application.

Ahora se procede a crear un proyecto y configurarlo para **Opencv en Visual Studio 2015**. Se accede a File/new/project y a Other languages/Visual C++ y se hace clic en Win32 Console Application. Abajo en Name se escribe el nombre del proyecto y clic en OK.

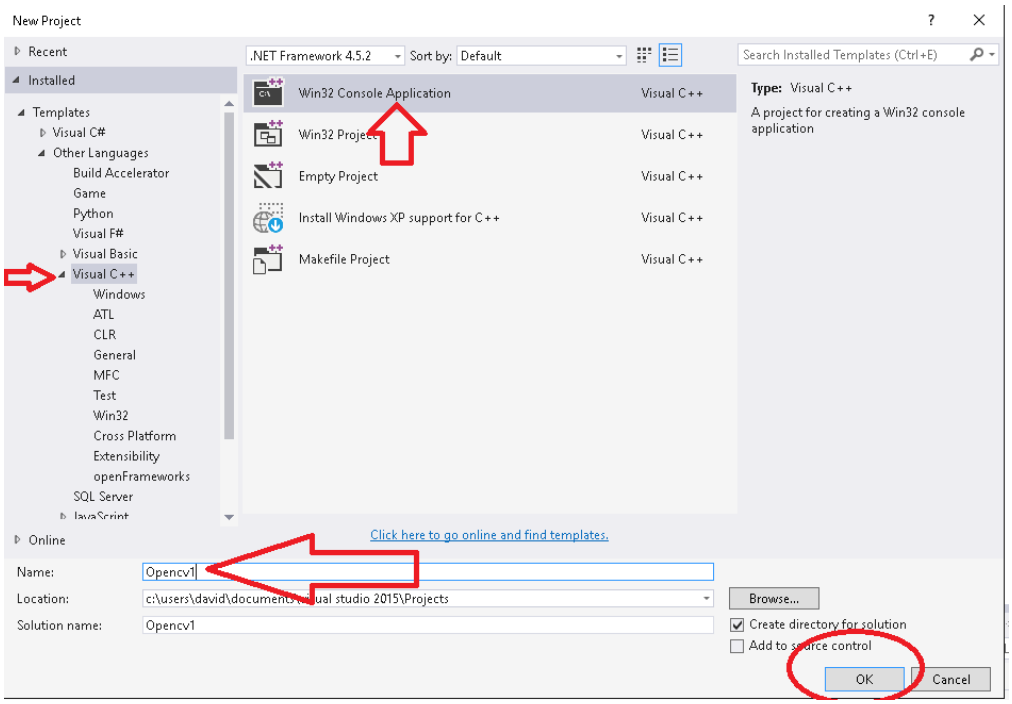


Figura 62. Instalación OpenCV – Captura 4

En la nueva ventana que se abre se hace clic en Next, en la siguiente se marca Empty project y clic en Finish.

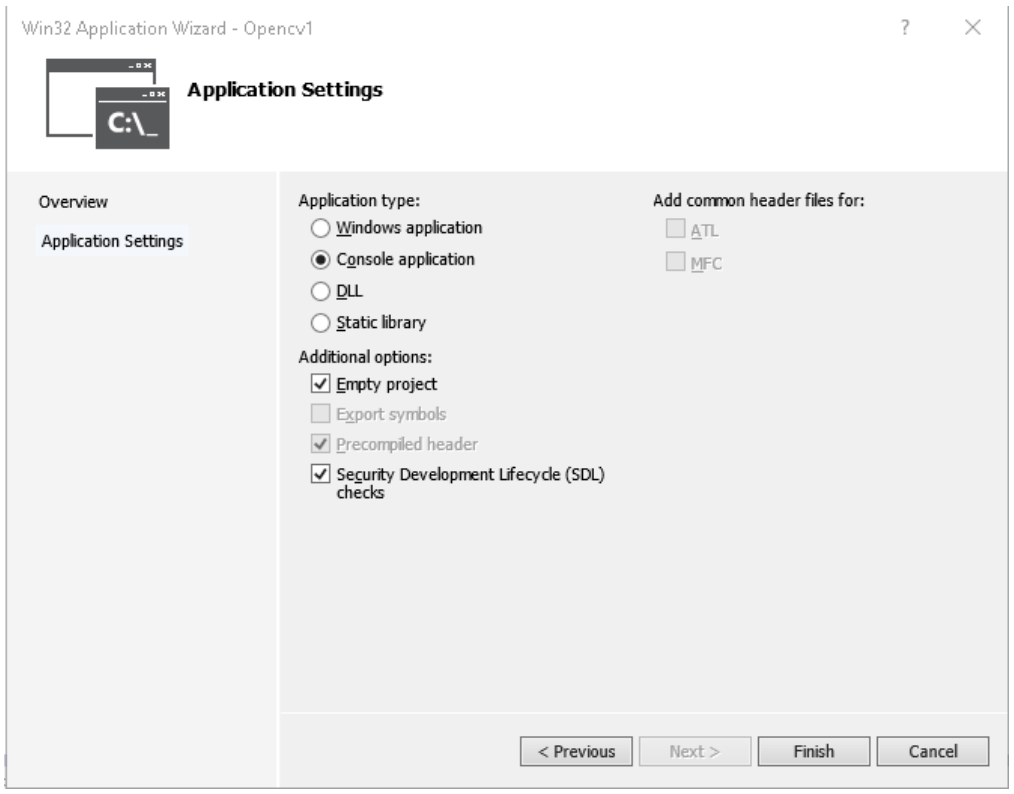


Figura 63. Instalación OpenCV – Captura 5

El proyecto ya está creado, ahora se configurará para OpenCV

Si no está abierta se abre la ventana de Soluction Explorer, accediendo a en View/Solution Explorer. En esa ventana clic derecho en el nombre del proyecto y luego en Properties.

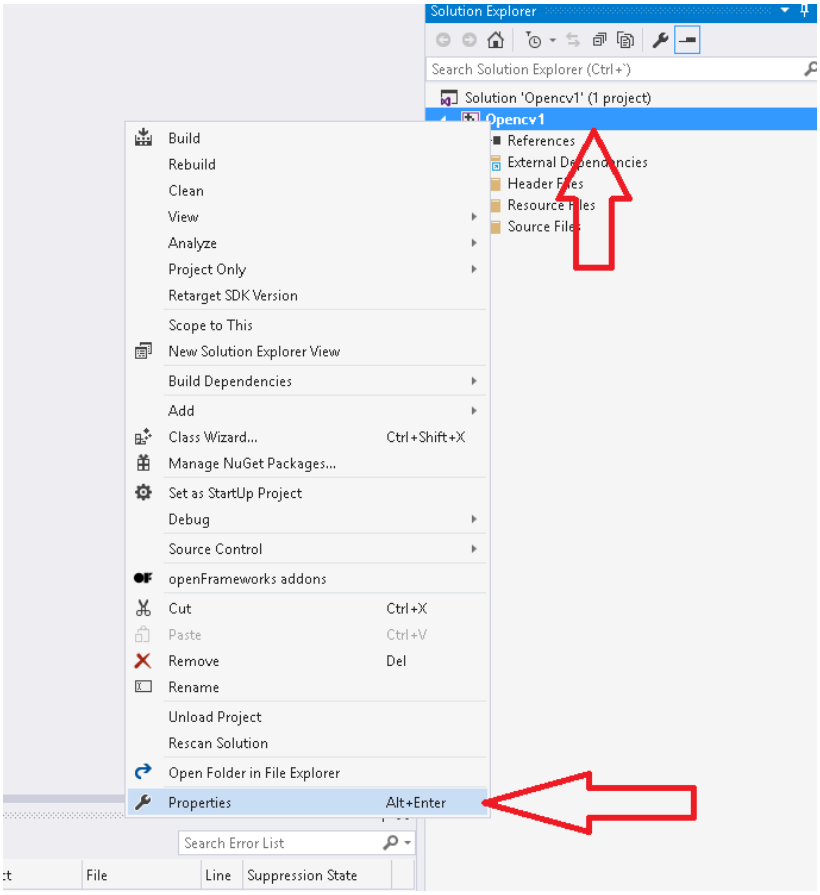


Figura 64. Instalación OpenCV – Captura 6

En la ventana nueva en el menú de la izquierda clic en VC++ Directories, en la derecha en executable Directories. Damos clic y se verá una flecha a la derecha, se pulsa y clic en Edit..

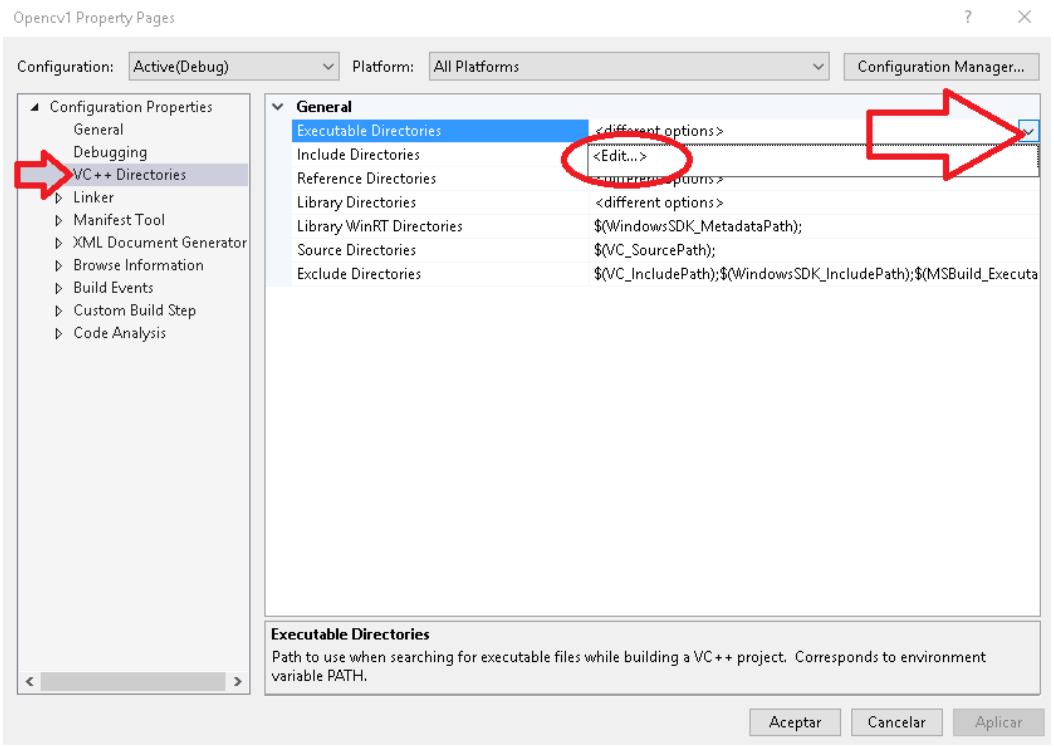


Figura 65. Instalación OpenCV – Captura 7

Se abrirá una nueva ventana y se clicará en el icono de la carpeta para agregar un nuevo directorio, luego clicamos en el botón que aparece con los puntos suspensivos para buscar el directorio. Se debe de agregar la carpeta C:\opencv\build\x64\vc14, siempre que se haya dejado la carpeta de Opencv en c:/. También se puede pegar la ruta directamente sin tener que buscarla. Luego clic en ok y luego en aplicar.

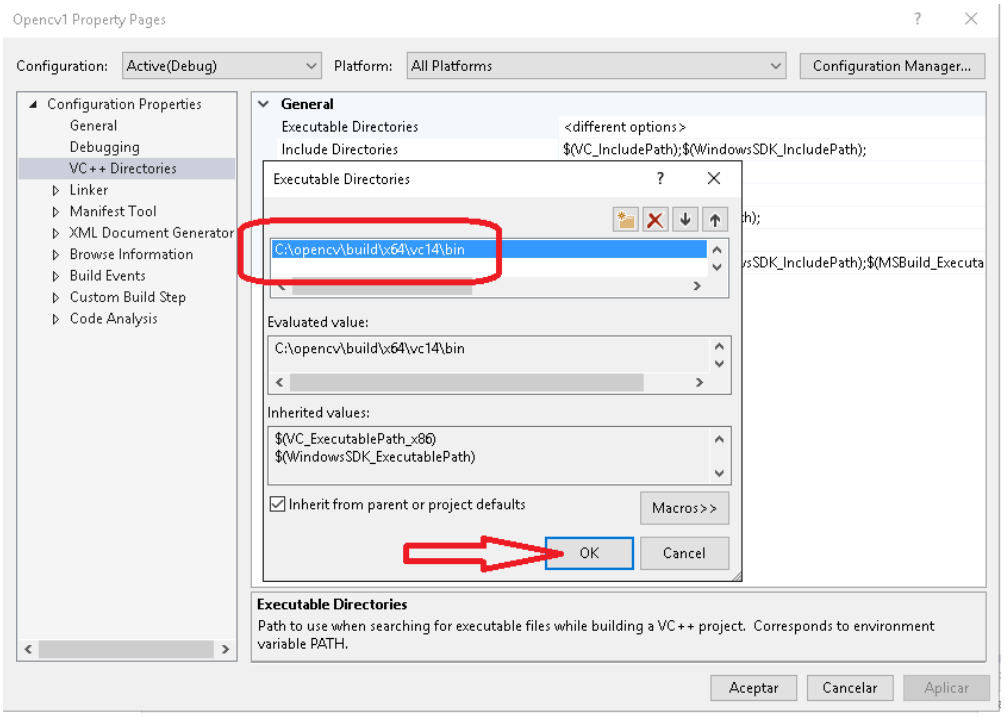


Figura 66. Instalación OpenCV – Captura 8

Más abajo en Library Directories se realiza la misma acción, pero la ruta será hacia la carpeta Lib:

```
C:\opencv\build\x64\vc14\lib
```

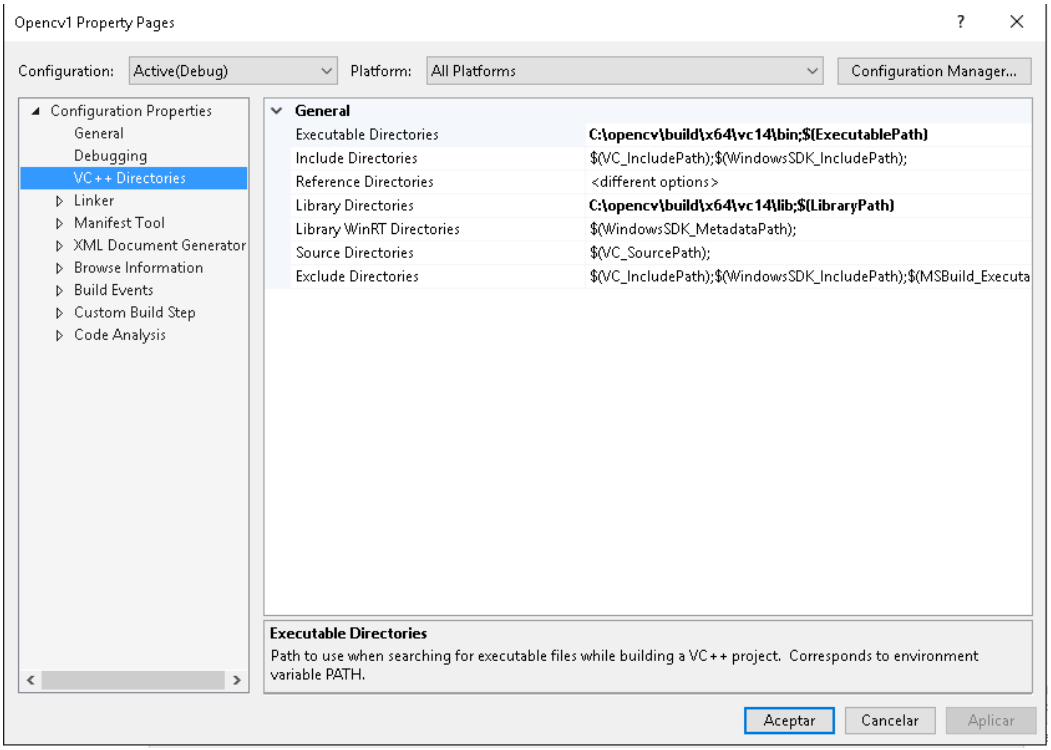


Figura 67. Instalación OpenCV – Captura 9

Otra vez a la izquierda se clicla Linker/general y en Additional Library Directories se agragará la carpeta .lib otra vez:

```
C:\opencv\build\x64\vc14\lib
```

Ahora en Linker/Input en additional Dependencies se agrega el nombre de los archivos con extension .lib de la carpeta lib agregada en el paso anterior, como el modo es Debug son todos los que su nombre termina en d. En versiones anteriores de Opencv había bastante archivos, pero en esta versión solo hay uno así que se tendrá que agregar opencv_world310d.lib . Esta es la única diferencia para configurar debug y reléase. En el caso de configurar release añadiremos opencv_world310.lib .

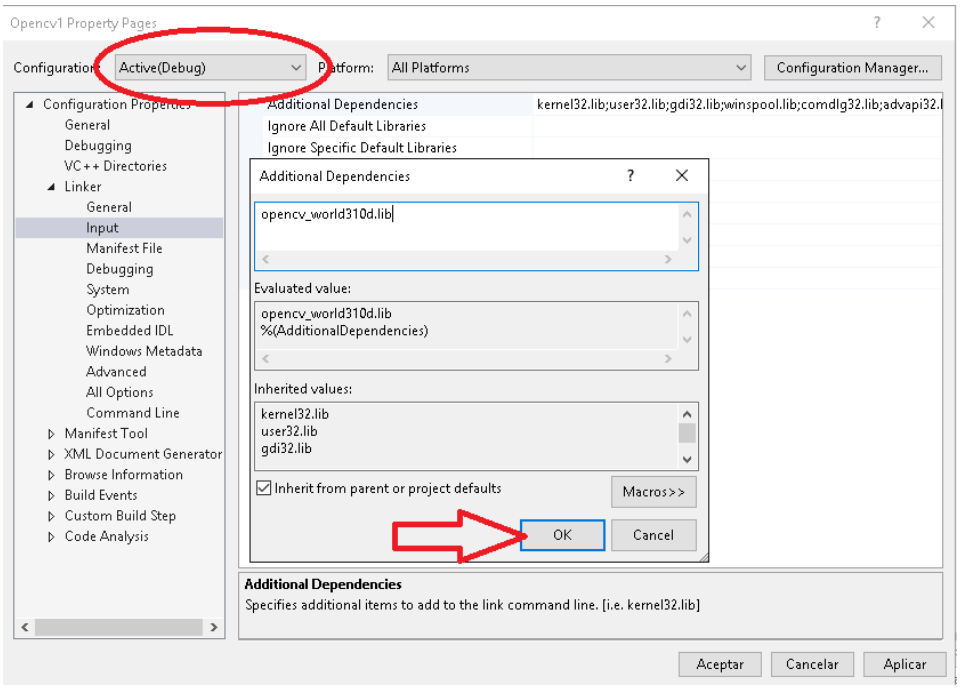


Figura 68. Instalación OpenCV – Captura 10

Clic en aplicar y en aceptar.

Vamos a crear el primer archivo de nuestro proyecto. En la ventana de Solution Explorer clic derecho en Source Files/add/new ítem, clic en Visual C++ y en la derecha C++ File(.cpp) se añade el nombre Main.cpp por ejemplo.

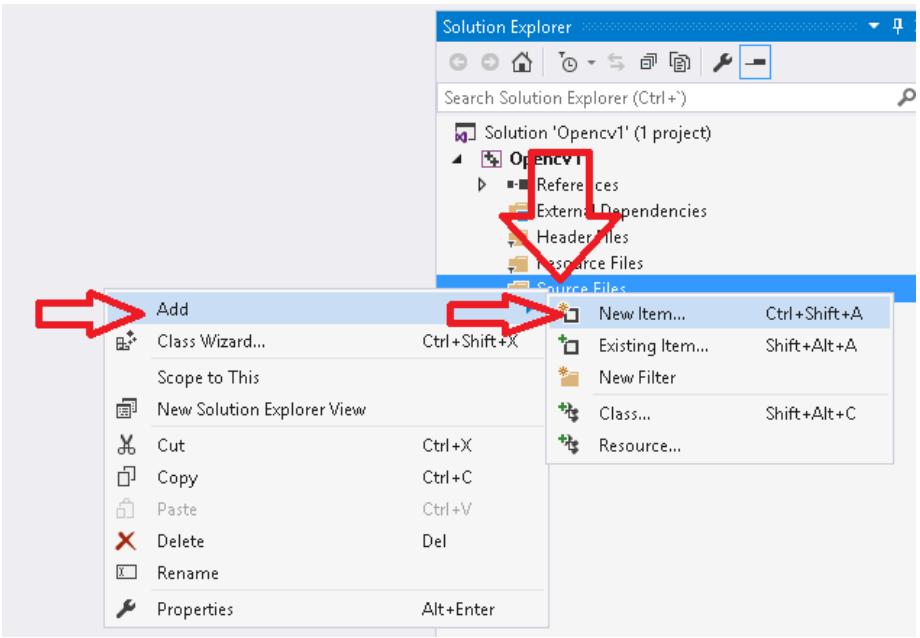


Figura 69. Instalación OpenCV – Captura 11

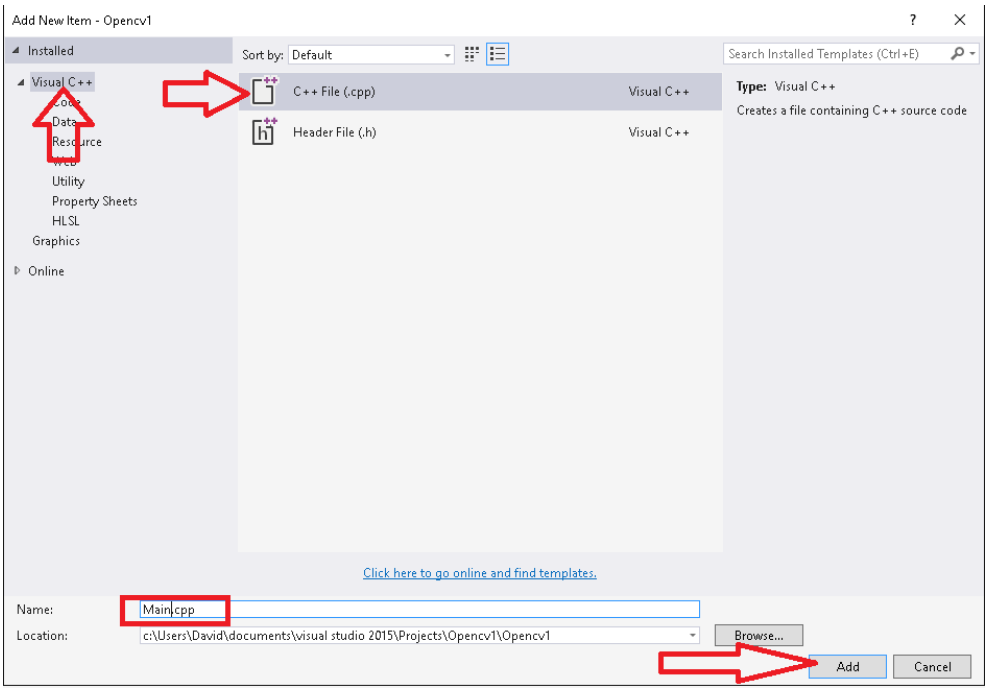


Figura 70. Instalación OpenCV – Captura 12

Clic derecho en nombre del proyecto en la venta de solution Explorer u Propierties. Ahora a la izquierda aparece C/C++ que antes no aparecía. Se repite el proceso: C/C++, general y a la derecha en additional Include Directories se agrega la carpeta include:

```
C:\opencv\build\include
```

Aplicar y aceptar.

Ahora se crea una plantilla para poder usarla cuando se quiera sin tener que realizar el proceso completo cada vez. Se accede a Export Template, preguntará si queremos guardar y clic en Yes. En la nueva ventana se clicará en Porject template y Next.

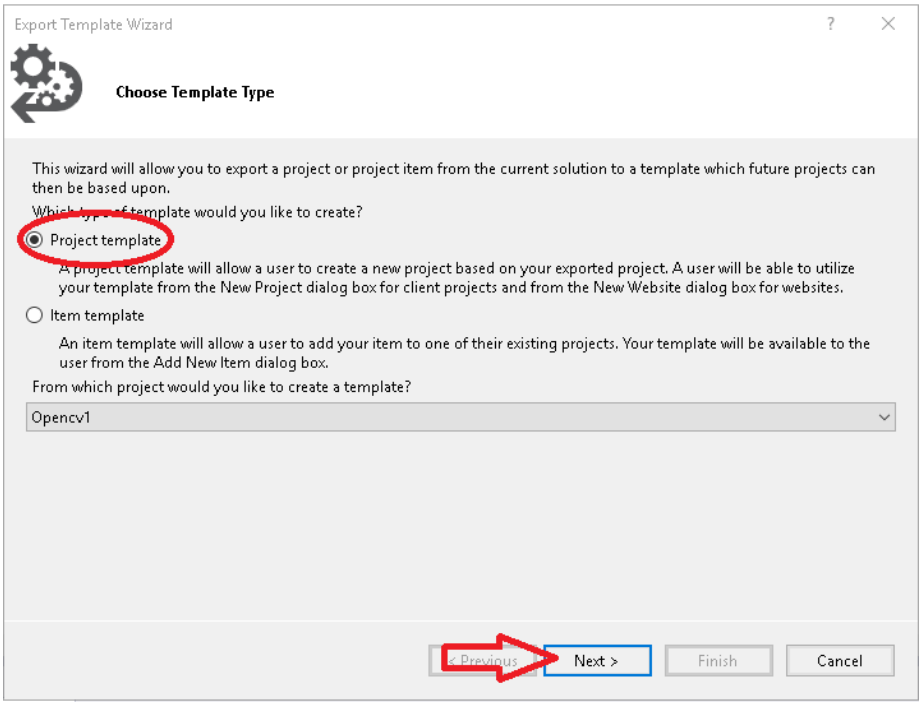


Figura 71. Instalación OpenCV – Captura 13

En la siguiente ventana se completa con el nombre y una descripción, y clic en Finish.

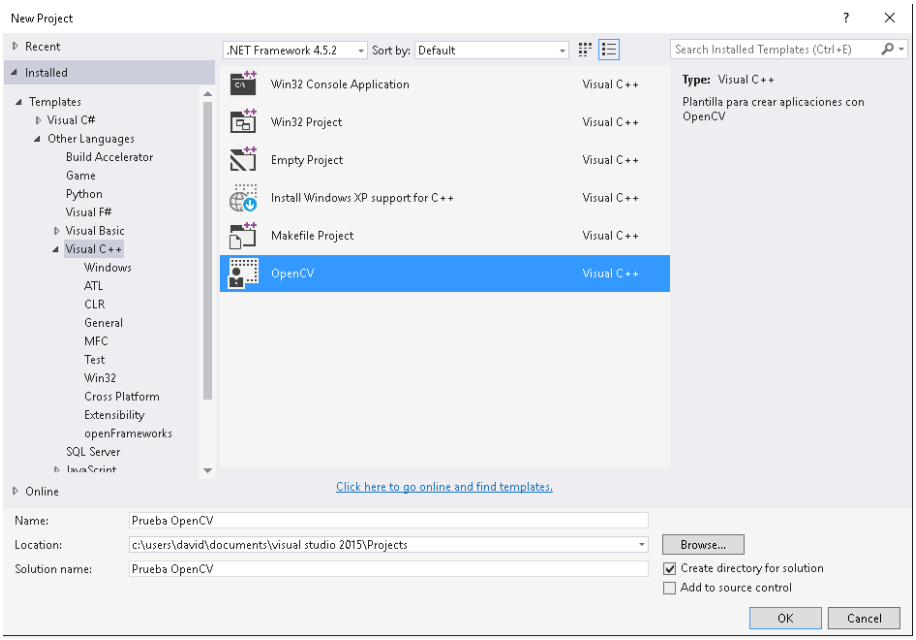


Figura 72. Instalación OpenCV – Captura 14

Y así finaliza la instalación.