ESCUELA POLITÉCNICA SUPERIOR DE CÓRDOBA
Universidad de Córdoba

Departamento de Ingeniería Electrónica y de Computadores.

SOFTWARE PARA EL CONTEO DIFERENCIAL DE MICROFIBRAS NATURALES Y MICROPLÁSTICOS EN IMÁGENES

Autores
Jesús Rioja Bravo
Ezequiel Herruzo Gómez

Julio, 2021

UNIVERSIDAD D CÓRDOBA

Índice

Índice de figuras

1. Introducción

En la actualidad, los textiles producidos a gran escala están formados por dos tipos de microfibras, las llamadas microfibras naturales y los microplásticos. Debido a la gran contaminación que producen los microplásticos al medio ambiente, es conveniente reducir la utilización de este tipo de microfibras, y así conseguir un uso casi exclusivo de microfibras naturales en la producción de textiles. Para conseguir este cometido y conocer los porcentajes de cada uno de los tipos e microfibras nombrados anteriormente que existen en distintos tipos de textiles, hay laboratorios que se encargan de sacar imágenes a nivel microscópico de las microfibras que componen estos textiles.

El problema reside en que, actualmente, una vez se ha obtenido una imagen por microscopio de uno de los textiles, la identificación de los distintos tipos de microfibras se realiza manualmente. Actualmente no hay ningún algoritmo desarrollado para la identificación de microfibras naturales o microplásticos. Así, el propósito de este proyecto es el desarrollo de un algoritmo que consiga diferenciar entre estos dos tipos de microfibras de una manera automática, además de al mismo tiempo obtener un mayor porcentaje de identificación del que actualmente de tiene, que es de un 80 %.

Durante el desarrollo de este documento, primero se explicarán las todas soluciones que se decidieron implementar hasta finalmente dar con la implementación final y que mejor resultado ha proporcionado. Una vez explicada la solución final, se detallarán distintas futuras mejoras al proyecto y, por último, se hablarán de las conclusiones obtenidas durante el desarrollo del proyecto.

2. Primeras implementaciones

En esta sección de procederá a mencionar las diferentes soluciones que se trataron de implementar hasta llegar a la solución final. Se explicará también el funcionamiento de estas posibles soluciones y el porqué fueron finalmente descartadas. Cabe mencionar que todos los algoritmos han sido desarrollados en C++ haciendo uso de OpenCV, librería pensada para el desarrollo de aplicaciones de visión por computador.

2.1. Identificación a partir del punto medio

2.1.1. Idea General

En las imágenes obtenidas por microscopio se puede apreciar la forma de ambos tipos de microfibras. Los microplásticos tienen una forma redondeada, aunque con bastantes imperfecciones, mientras que las microfibras naturales tienen una forma alargada. Algo también bastante remarcable es que, normalmente, los microplásticos son bastante más pequeños que las microfibras naturales. En esta imagen se pueden apreciar cómo son ambos tipos de microfibras.

Figura 1: Imagen obtenida por microscopio.

Por este motivo, la primera solución que se trató de dar a este problema fue a partir de las diferencias que hay entre las formas de ambos tipos de microfibras.
Para poder identificar la forma de cada una de las microfibras em una imagen, lo primero que se ha de hacer es reconocer el contorno de cada una de estas.

Una vez obtenidos los contornos de cada una de las microfibras y conocidas las coordenadas en los ejes X e Y, para cada uno de los contornos se hace lo siguiente:

1. Se encuentran la coordenadas máxima y mínima respecto al eje X.
2. Se calcula el punto medio respecto a esas dos coordenadas.
3. Se define la función de una recta que pase por estos puntos.
4. Se calcula la pendiente de esta recta.
5. Conocida la pendiente, se define la función de una recta que pase por el punto medio calculado anteriormente y que sea perpendicular a la esta (pendiente inversa).
6. Se buscan los dos puntos de el contorno que pertenezcan a esta nueva recta y se calcula su distancia.
7. Se calcula la distancia de estos dos puntos.
8. Se divide la distancia de las coordenadas máxima y mínima entre la distancia de los dos nuevos puntos calculados
9. Se hayan los puntos medios entre el anterior punto medio y las coordenadas máxima y mínima respectivamente, y para ambos nuevos puntos se realizan los pasos 5, 6, 7 y 8.
10. Se realizan las iteraciones del paso 9 que sean necesarias.

Si el contorno en el que nos encontramos se trata de un microplástico, al hacer la primera división obtendremos un número cercano a 1, ya que al ser una figura circula, ambas distancias han de ser similares. En las sucesivas divisiones este valor irá aumentando gradualmente, ya que el numerador se mantendrá mientras que el denominador se irá reduciendo.

En cambio, si el contorno de trata de una microfibra natural, pueden darse dos circunstancias distintas. Puede que el resultado de la primera división sea muy alejado de 1, o que en las posteriores divisiones no se mantenga un aumento gradual del resultado. En ambos casos sabremos que el contorno en el que nos encontramos pertenece a una microfibra natural.

2.1.2. Desarrollo

Para conseguir este objetivo lo que se ha realizado en primer lugar ha sido obtener los contornos de las distintas microfibras existentes mediante el uso de una función de OpenCV llamada “FindContours()”. Esta función nos devuelve un vector de vectores de puntos (para cada coordenada), en la que cada posición del vector resultado corresponde a uno de los contornos detectados en la imagen. Nuestro objetivo es que para cada microfibra detectada, esta función nos devuelva su correspondiente contorno, de forma que al final, obtengamos tantos contornos como microfibras hay en la imagen.

Si aplicamos la función “FindContours()” a la imagen original, debido a la calidad de esta, los contornos detectados serán erróneos, ya que habrá fibras por las que se detecte más de un contorno. Para evitar esto, es necesario realizar un post-procesado de la imagen, de manera que al obtener los contornos, haya el mismo número de contornos que de microfibras.
Para detectar los contornos adecuadamente, a la imagen original, tras pasarla a escala de grises, se le ha aplicado, en primer lugar, un suavizado con la función ´´Blur()” y, en segundo lugar, se le ha aplicado el algoritmo de detección de bordes ´´Canny()”, el cual nos devolverá una imagen en la cual solamente se marcan los bordes de las figuras detectadas. El resultado tras aplicar ambos filtros es el siguiente.

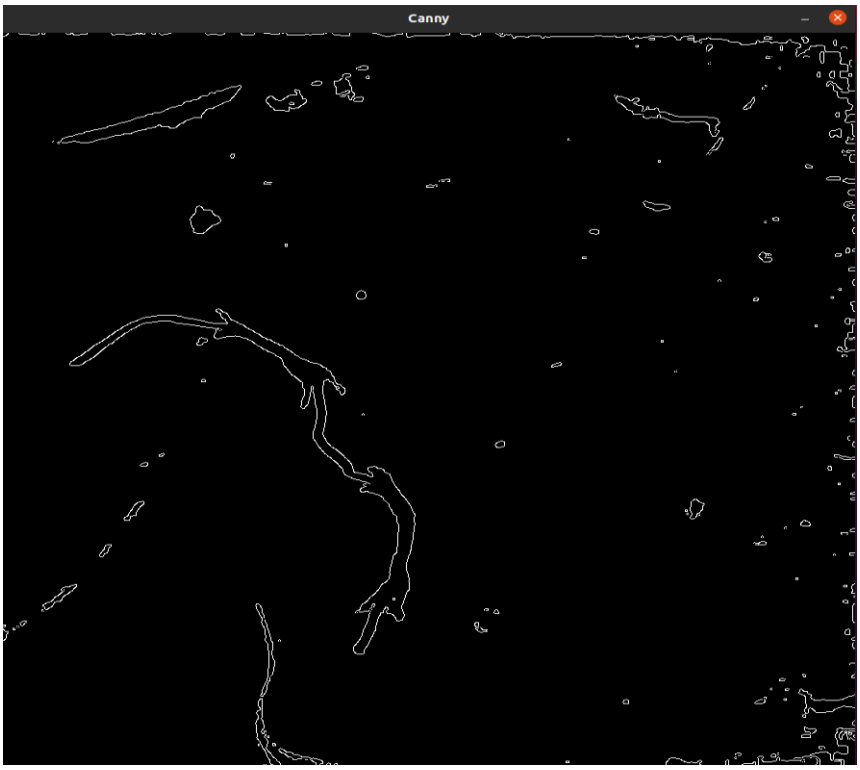

Figura 2: Resultado de aplicar el algoritmo Canny a nuestra imagen.

Una vez se ha realizado el post-procesado de la imagen y aplicado la función ´´FindContours()", se tendrán todos los contornos con sus respectivas coordenadas, por lo que ya solo falta programar las operaciones necesarias para llevar a cabo el proceso mencionado en el apartado anterior.

Finalmente, esta posible solución fue rechazada debido a que, al ponerlo en práctica, las coordenadas que nos proporcionaba la función ´´FindContours()" a veces no eran continuas, por lo que había ciertos puntos que no se estaban contemplando dentro del vector proporcionado por esta.
Este problema hacía que a veces, a la hora de buscar los puntos del contorno que pertenecieran a la recta perpendicular a la formada por las coordenadas máxima y mínima respecto al eje X, no se hallara ningún punto perteneciente a esta, por lo que no se podía realizar el cálculo para encontrar la relación entre las distancias.

2.2. Regresión Lineal

2.2.1. Idea general

La siguiente idea que se llevó acabo fue aplicar regresión lineal a cada uno de los conjuntos de puntos que conforman cada uno de los contornos detectados.

La regresión lineal consiste en, dada una nube de puntos sobre los ejes, aproximar esta a la mejor recta posible. Esto es algo que normalmente se usa en computación para, dados unos valores iniciales llamados datos de entrenamiento, aproximar estos valores a una función que nos ayude a predecir futuros resultados.

Uno de los parámetros más importantes en regresión lineal es el error acumulado. Esto no es más que la suma de las distancias de los puntos originales respecto a la recta generada, de manera que cuánto más dispersos y alejados estén estos puntos, mayor será el error acumulado.

Aplicando regresión lineal a cada uno de los contornos y calculando su error acumulado, se pensó que se podría diferenciar entre ambos tipos de microfibra, ya que, en la teoría, las microfibras naturales tienen una forma alargada similar a una recta, mientras que los microplásticos tienen una forma circular. Partiendo de estos conocimientos, se supuso que el error acumulado de los microplásticos siempre sería mayor que el de las microfibras naturales, ya que tendrían una mayor dispersión respecto a la de estas segundas, por lo que se podría discriminar entre ambos tipos a partir de un umbral determinado.

2.2.2. Desarrollo

Para el desarrollo de este algoritmo se partió del post-procesado realizado con anterioridad, además de que se volvió a hacer uso de la función ´´FindContours()".

A partir de los puntos que conforman cada uno de los contornos, realizando diversos cálculos, se buscó una recta de la forma $Y = NX + M$ que más se aproximara al contorno detectado, donde los valores N y M son calculados a partir de los puntos del contorno.

Una vez obtenida esta recta, se calculó el error acumulado. Para esto, se realizó el sumatorio de la diferencia entre el valor original en Y para cada valor de X y el valor de Y obtenido al sustituir X en la recta calculada.

´ Al observar los distintos errores acumulados para ambos tipos de microfibras, se observó que no existía una diferencia significativa entre ellos, por lo que esta solución fue descartada.
Esto se debe principalmente a dos motivos. El primero es que en la realidad, la forma de las microfibras naturales varía mucho de unas a otras, por lo que la dispersión de puntos es, en muchas ocasiones, mayor a la dispersión obtenida en microplásticos. El segundo motivo es que los microplásticos, en ocasiones, son de un tamaño muy pequeño, por lo que sus contornos están formados por apenas varios puntos, cosa que también dificulta la aplicación de este algoritmo.

2.3. Machine Learning

2.3.1. Idea general

OpenCV tienen implementados diversos de los algoritmos de machine learning más utilizado en la actualidad. Algunos de estos algoritmos son: KNN (K-Nearest Neighbors), SVM (Support-Vector Machine), Árboles de decisión, Random Trees, Regresión Logística...

Debido a esto, antes de dar con la solución final, se trató de implementar alguno de los algoritmos de machine learning que nos proporciona OpenCV.
Debido a que la clasificación que nos interesa es binaria (dos clases) y a la potencia y efectividad de estos algoritmo, se optó por aplicar o SVM o Regresión Logística.

El problema surgió a la hora de tratar de obtener el conjunto de entrenamiento y de test. Estos conjuntos deben de estar compuestos por imágenes independientes de ambos tipos de microfibras, etiquetadas para el conjunto de entrenamiento y no etiquetadas para el de test. Por lo que finalmente, debido a que nuestras imágenes no son compatibles estos algoritmos, y que obtener estos conjuntos de test y de entrenamiento en el formato necesario requeriría de mucho tiempo extra, esta posible solución tuvo que ser descartada.

3. Implementación final - Solución

3.1. Idea General

Finalmente, basándonos en la primera idea de diferenciar ambos tipos de microfibras a partir de su forma, se consiguió desarrollar un algoritmo que pudiera identificar tanto microplásticos como microfibras naturales, clasificándolas correctamente.

Para desarrollar este algoritmo nos basamos en la idea de que, si vamos comprobando los cambios respecto a los ejes X e Y a lo largo de un contorno, para los microplásticos, observaremos que estos cambios deben de ser más o menos similares en ambos ejes. Mientras que para las microfibras naturales, los cambios en ambos ejes distarán mucho de uno respecto al otro.

3.2. Desarrollo

Para el desarrollo de este último algoritmo, se realizaron ciertos cambios en el post-procesado de la imagen y en el método de obtención de contornos.

Durante el post-procesado, se decidió que además de utilizar el método ´´Canny" para la detección de bordes, previamente la imagen se pasaría a imagen binaria, con solamente dos colores (blanco y negro) ya que se comprobó que al aplicar la función ´´FindContours()" a imágenes con este formato, los resultados eran mucho más satisfactorios.
Además de esto, ciertos parámetros de la función ´´FindContours()" fueron modificados, para que así el vector que esta nos devuelve sea más preciso en los puntos que conforman cada uno de estos contornos.

Una vez llevados a cabo estos cambios, se procedió al desarrollo del algoritmo detector de las microfibras, el cual funciona de la siguiente forma:

1. Se divide el contorno en 8 secciones de igual tamaño.
2. Para cada sección, se calcula, a partir de sus puntos inicial y final, los cambios producidos tanto en el eje X como en el eje Y.
3. Una vez calculado el cambio producido en ambos ejes para cada sección, se hace una división de estos dos valores.

Si el resultado de estas divisiones es cercano a 1, significa que el cambio en el eje X y en el eje Y ha sido parecido, mientras que cuanto más se aleje de 1, más habrá cambiado en un eje respecto al otro.
Debido a que los microplásticos poseen una forma redondeada, los cambios producidos en sus secciones respecto a los ejes X e Y serán cercanos a 1 en la mayoría de casos, mientras que en las microfibras naturales, en la mayoría de ocasiones estos cambios serán muy dispares y alejados de 1. De esta forma, se podrá discriminar entre ambos tipos de microfibras
Mencionar también que, tras estudiar las imágenes, se llegó a la conclusión de que las microfibras más pequeñas son siempre microplásticos, por lo que estas serán siempre clasificadas automáticamente como tales, sin tener que pasar por el algoritmo explicado anteriormente. De esta forma, solamente las microfibras de mayor tamaño serán las que pasen por este proceso de clasificación.

Gracias a este sistema, se ha conseguido clasificar correctamente entre ambos tipos de microfibras.

3.2.1. Diagrama de flujo

El diagrama de flujo que sigue nuestro algoritmo es el siguiente.

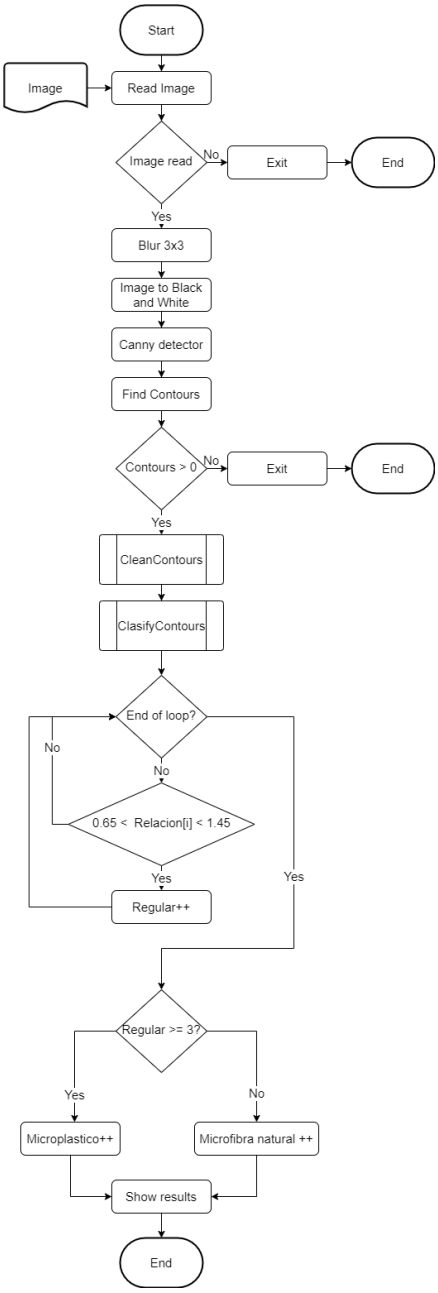

Figura 3: Diagrama de flujo principal.

3.2.2. Código

En esta sección se muestra el código implementado en C++, en bruto.

```
#include "opencv2/imgcodecs.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/photo.hpp"
#include <iostream>
#include <math.h>
#include <vector>


using namespace cv;
using namespace std;


Mat src_gray;
Mat src_bw;
Mat detected_edges;

vector<vector<Point> > contours;
vector<vector<Point>> Distances; // Vector que, para cada contorno, guarda las distancias en los
    ejes de ordenadas y abscisas entre los puntos a tener en cuenta
vector<vector<double> > Relaciones; // Vector que, para cada contorno, guarda la correlacin
    (divisin) entre pares de distancias (ambos ejes)
int actualContour;

int fibras = 0;
int microPlasticos = 0;

RNG rng(12345);

void SetContoursVector(int, void* );
void ShowContoursCords();
void ClasifyContours();
void CleanContours();
void getDistances();


int main( int argc, char** argv )
{

    const char* source_window = "Source";
    CommandLineParser parser( argc, argv, "{@input | imagenes_seleccionadas/img2.jpg | input
        image}" );
    Mat src = imread( samples::findFile( parser.get<String>( "@input" ) ));
    if( src.empty() )
    {
      cout << "Could not open or find the image!\n" << endl;
      cout << "Usage: " << argv[0] << " <Input image>" << endl;
      return -1;
    }

    blur( src, src, Size(3,3));
    cvtColor( src, src_gray, COLOR_BGR2GRAY );

    namedWindow( source_window );
    imshow( source_window, src_gray );
```

```
    src_bw = src_gray > 164;

    actualContour = 0;

    SetContoursVector( 0, 0 );
    ShowContoursCords();

    ClasifyContours();

    int regular = 0;

    for(int i = 0; i < Relaciones.size() ; i++)
    {
        for(int j = 0; j < Relaciones[i].size() ; j++)
        {

            if(Relaciones[i][j] <= 1.45 && Relaciones[i][j] >= 0.45)
                regular++;
        }

        if(regular >= 4)
            microPlasticos++;
        else
            fibras++;
        regular = 0;
    }

    ErrSumValues.clear();
    Distances.clear();
    Relaciones.clear();

    cout << "En esta imagen hay " << fibras << " fibras y " << microPlasticos << " microplsticos."
         << endl;

    waitKey();

    return 0;
}


void SetContoursVector(int, void* )
{
    Mat canny_output;
    vector<Vec4i> hierarchy;

    Canny( src_bw, canny_output, thresh, thresh*1.5 );
    imshow( "Canny", canny_output );

    findContours( src_bw, contours, hierarchy, RETR_LIST, CHAIN_APPROX_NONE );
    Mat drawing = Mat::zeros( src_bw.size(), CV_8UC3 );

    if(contours.size() > 0)
        CleanContours();



    for( int i = 0; i< contours.size(); i++ )
    {
```

```
            Scalar color = Scalar( rng.uniform(0, 256), rng.uniform(0,256), rng.uniform(0,256) );
            drawContours( drawing, contours, (int)i, color, 2, LINE_8, hierarchy, 0 );
    }

    cout << endl << "Hay " << contours.size() << " contornos" << endl << endl;
    imshow( "Contours", drawing );

}

void ShowContoursCords()
{
    for( size_t i = 0; i< contours.size(); i++ )
    {
            cout << "CONTORNO " << i << ": "<< " con " << contours[i].size() << " puntos." << endl;
            for( size_t j = 0; j< contours[i].size(); j++ )
            {
                cout << j << ": " << contours[i][j] << endl;
            }

            cout << endl;

    }
}


void ClasifyContours() //TODO
{
    //cout << "He entrado en ClasifyContours" << endl << endl;
    double aux;
    vector<double> auxRel;

    for(int i = 0; i < contours.size(); i++)
    {

            actualContour = i;
            cout << "CONTORNO " << i << " con " << contours[i].size() << " puntos." << endl;
            /*CalculateData();
            cout << "Equation of best fit is: Y = " << N << " + " << M << "x;" << endl;
            cout << "Error en M: " << (long int)ErrMValues[i]<< endl;
            cout << "Error en N: " << (long int)ErrNValues[i] << endl;
            cout << "Error: " << (long int)ErrSumValues[i] << endl << endl;*/

            getDistances();

            for(int j = 0; j < Distances[i].size(); j++)
            {
                cout << "Distancia " << j << ": X-> " << Distances[i][j].x << " Y-> " <<
                    Distances[i][j].y << endl;
                aux = ((float)Distances[i][j].y)/((float)Distances[i][j].x);
                if(aux < 0)
                    aux *= -1;

                cout << "Relacin " << j << "-> " << aux << endl;
                auxRel.push_back(aux);

            }

            cout << endl;
            Relaciones.push_back(auxRel);
```

```
            auxRel.clear();

    }



}

void getDistances()
{
    vector<Point> auxDistance;
    Point aux;

    int tam = contours[actualContour].size() - 1;


    for( float i = 0; i <= 0.875; i += 0.125 )
    {
        aux.x = contours[actualContour][tam * (i + 0.125)].x - contours[actualContour][tam * i].x;
        aux.y = contours[actualContour][tam * (i + 0.125)].y - contours[actualContour][tam * i].y;
        auxDistance.push_back(aux);
    }


    Distances.push_back(auxDistance);
    auxDistance.clear();



}


void CleanContours()
{
    int maxTam = 0;
    for( int i = 0; i < contours.size(); i++ )
    {
            if( contours[i].size() < 50)
            {
                if( contours[i].size() > 16)
                    microPlasticos++;

                contours.erase(contours.begin() + i);
                i--;

            }
            else if(contours[i].size() > contours[maxTam].size())
                maxTam = i;


    }

    contours.erase(contours.begin() + maxTam);

}
```

3.3. Resultados obtenidos

En esta sección se procederá a mostrar el proceso de ejecución del código desarrollado.

Una vez compilado, se puede proceder a ejecutar el código, diciéndole en el proceso la imagen que se desea utilizar.
Cuando ejecutemos el código, se nos mostrarán varias imágenes, además de la información que aparecerá por consola.

1. Imagen original.

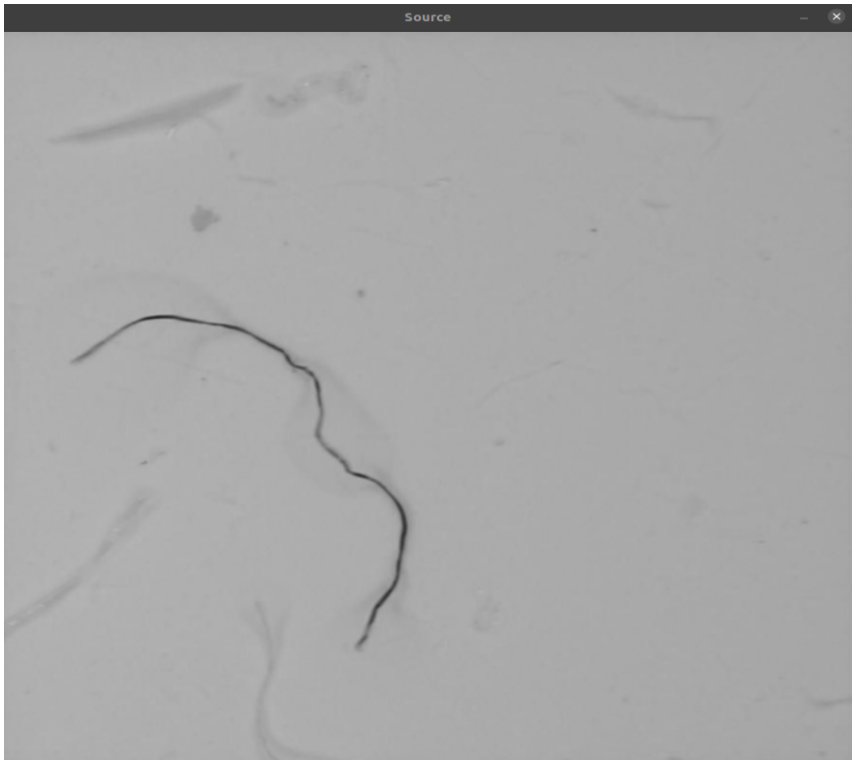

Figura 4: Imagen original.

2. Imagen una vez aplicado el filtro Canny para la detección de bordes.

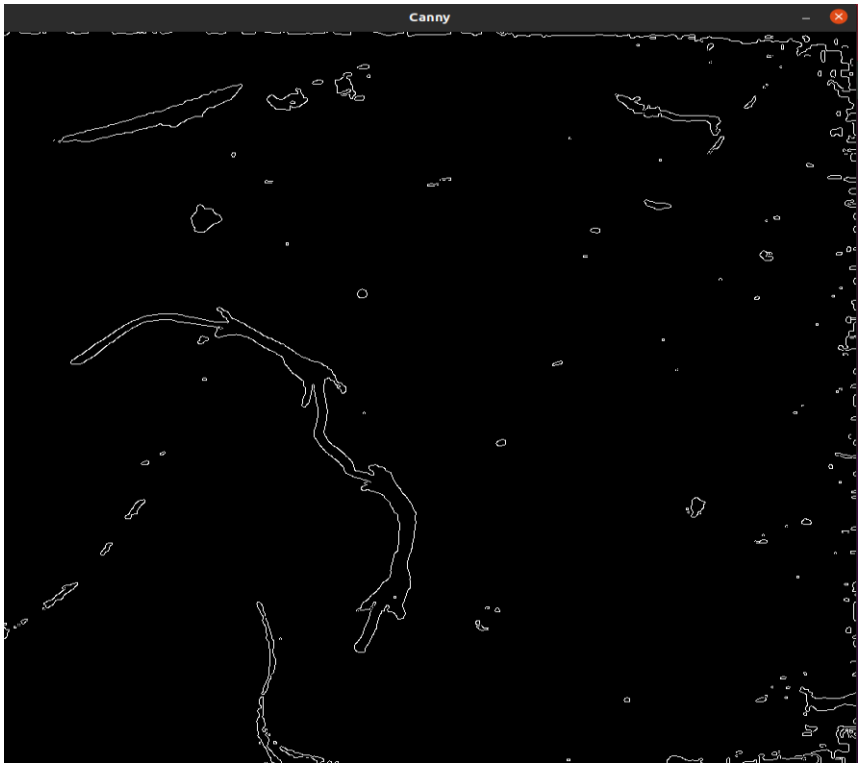

Figura 5: Imagen Canny.

3. Contornos de mayor tamaño encontrados.

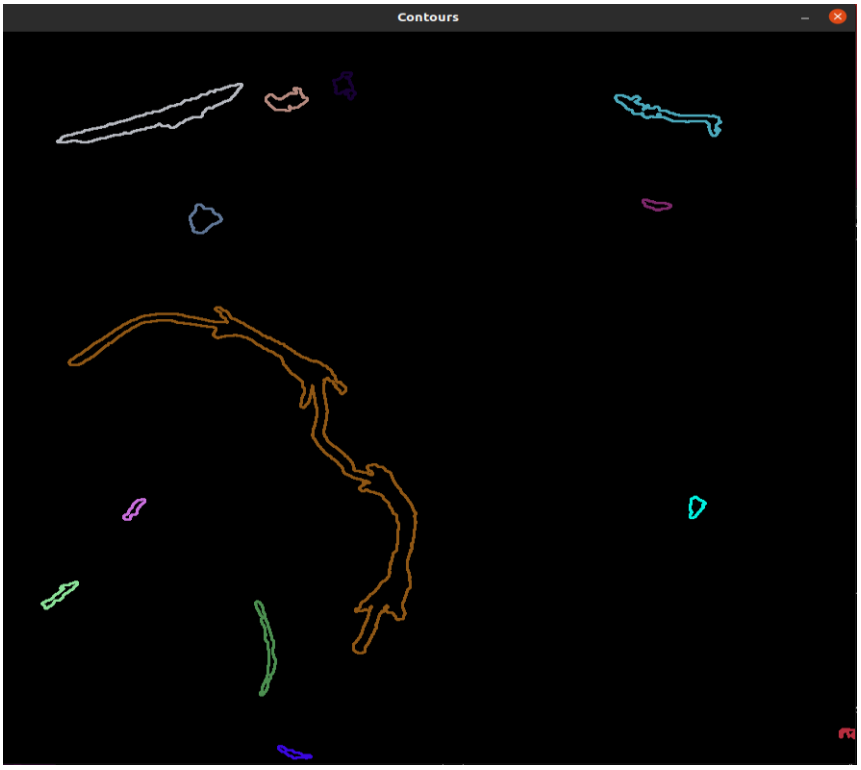

Figura 6: Contornos detectados.

4. Número de contornos detectados, además de los puntos que conforman cada uno de estos.

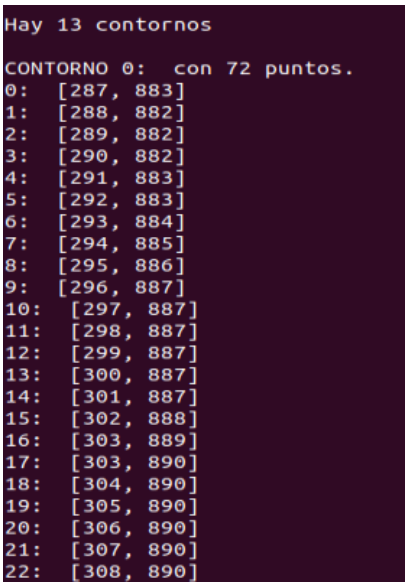

Figura 7: Forma y cantidad de contornos.

5. Información recopilada para cada uno de los contornos.

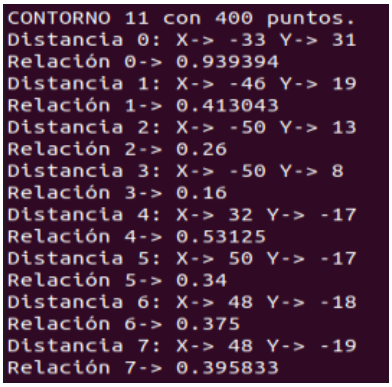

Figura 8: Datos obtenidos.

6. Resultados obtenidos.

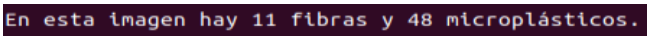

Figura 9: Resultados obtenidos.

4. Posibles mejoras

Este algoritmo desarrollado, aunque nos proporciona lo que necesitamos con una buena tasa de acierto y es funcional, puede ser mejorado.
Una de las mejoras más notable y que no sería difícil de implementar es la forma de mostrar el resultado, ya que actualmente se muestra simplemente mediante una línea de texto por consola.
Una forma más visual de mostrar este resultado sería mostrando todos los contornos detectados detectados en la imagen, coloreando todos los microplásticos de un mismo color y todas las microfibras naturales de otro.

Además de mejorar el algoritmo existente, se pueden desarrollar nuevos algoritmos más precisos y que proporcionen más información.
Este algoritmo simplemente clasifica entre microfibras naturales y microplásticos, sin embargo, dentro de cada uno de estos tipos existen miles compuestos que no se están teniendo en cuenta. La identificación y clasificación de todos estos tipos de microfibras naturales y microplásticos se podrían hacer a partir del color de estos, ya que hacerlo a partir de su forma no sería viable, ya que muchos tipos podrían coincidir en esta.

5. Conclusión

La visión por computador es algo muy usado en la actualidad en muchísimos ámbitos debido a su versatilidad y eficacia. Dentro del campo de la visión, se pueden desarrollar algoritmos de complejidades baja o media, como el aquí desarrollado, que simplemente realiza algunos cálculos a partir de unos contornos detectados para reconocer la forma de diferentes tipos de microfibra, hasta algoritmos (machine learning) de enorme complejidad capaces de reconocer en tiempo real y en movimiento los rostros, junto a sus características, de cientos de personas simultáneamente.
Finalmente, destacar que el algoritmo que aquí se ha desarrollado no tiene por qué ser utilizado simplemente para el propósito aquí mostrado, si no que con pocas modificaciones, puede ser utilizado para la detección e identificación de formas en cualquier otro ámbito.