Visión por Computador (2016-2017)
Grado en Ingeniería Informática
Universidad de Granada

Proyecto final:
Detección de ojos en imágenes a color

Ana Puertas Olea
Laura Tirado López

ÍNDICE

Descripción del problema 2

Enfoque de la implementación y eficiencia 2

Experimentos realizados y resultados definitivos 10

Valoración de resultados y conclusiones 12

Referencias 12

Anexo I: Resultados 13

1. Descripción del problema

Nuestro problema consiste en la detección de ojos en imágenes a color. Para resolverlo, nos hemos basado en varios estudios y algoritmos como los que se describen en [1], [2] y [3].

La idea de este proyecto es poder implementar una solución para la detección de ojos lo más acertada posible en base a las referencias anteriormente citadas.

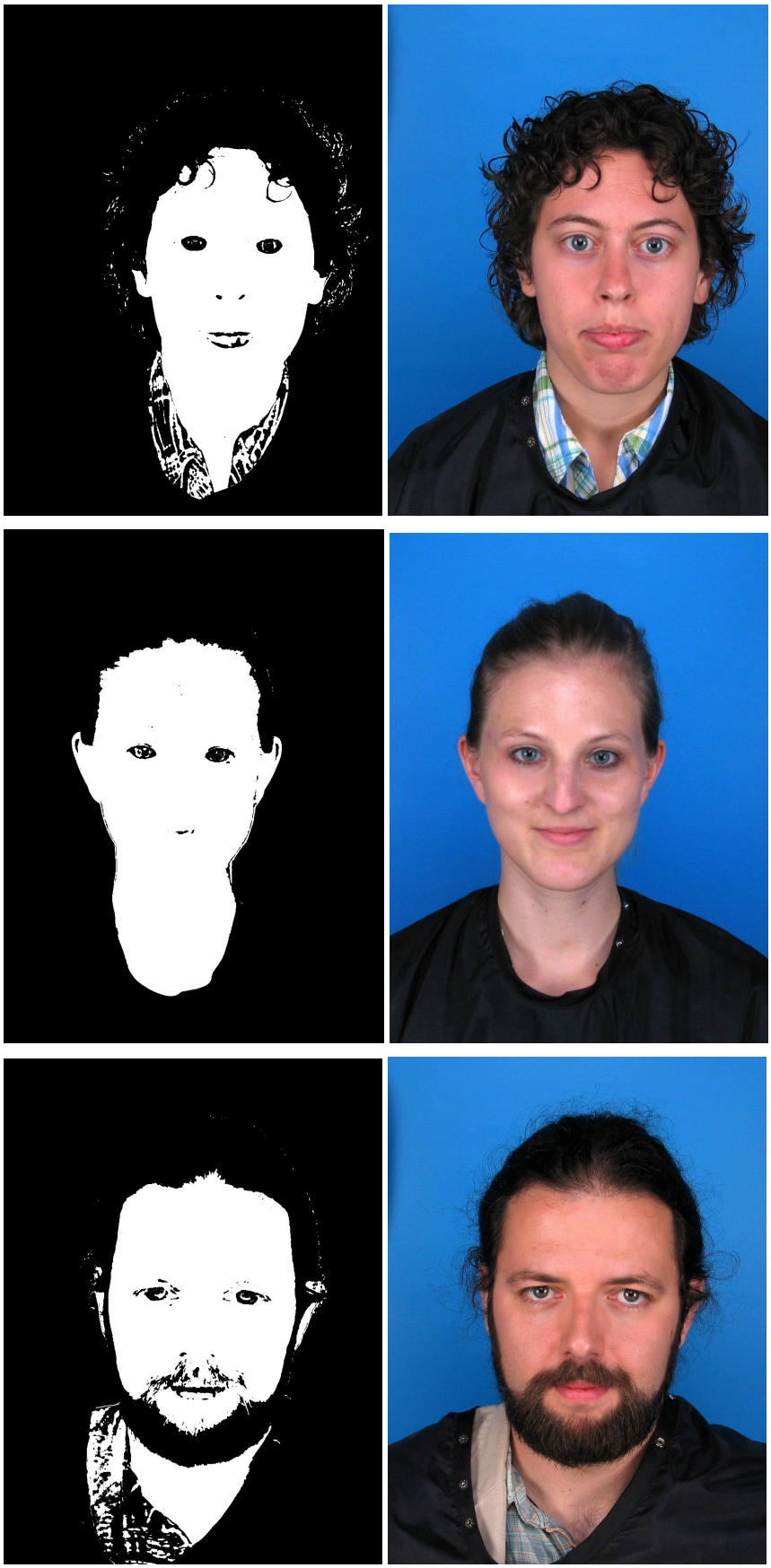
2. Enfoque de la implementación y eficiencia

La implementación se ha realizado en lenguaje C++, y la hemos dividido en dos partes:

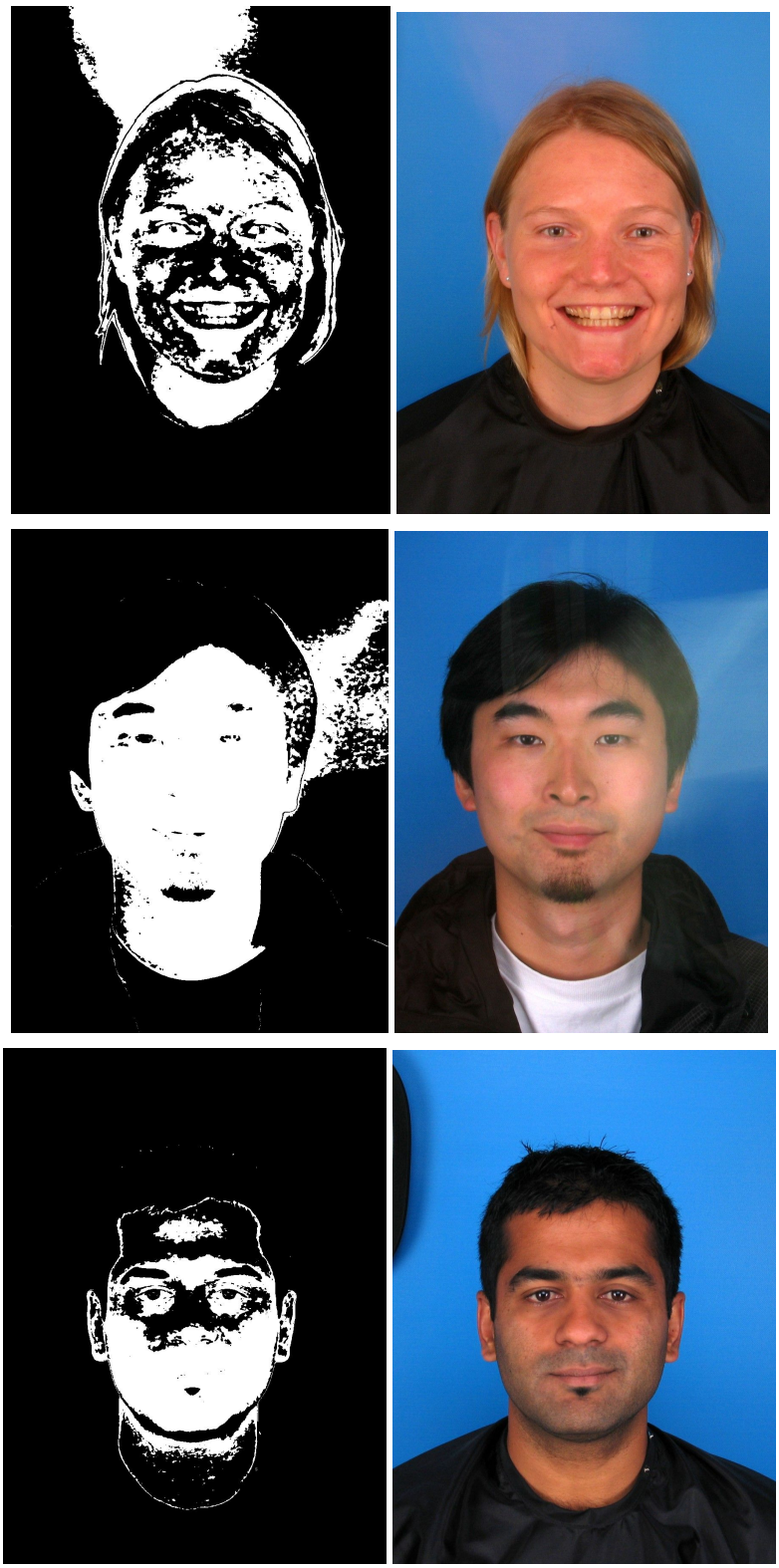
1. Detección de piel → Utilizamos una función en la que mediante un filtro de color detectamos la piel en la imagen previamente alisada con una función Gaussiana para eliminar ruido o altas frecuencias, y posteriormente convertida de RGB a HSV. Obtenemos como resultado una imagen que muestre mostrando en color blanco las zonas de piel y lo que no se considera piel en negro. Para llevarlo a cabo, hemos realizado lo siguiente:

```
Mat deteccionPiel(Mat imag, int indice){
  ....
  int const max_BINARY_value = 255;

 // Aplicamos un filtro Gaussiano para alisar la imagen y eliminar ruido
  GaussianBlur(imag, imag, Size(3,3),70);

  //Convertimos la imagen de RGB a HSV
  cvtColor(imag, imag_hsv, CV_BGR2HSV);

  //Le pasamos la imagen convertida en hsv, y la función deja a negro el píxel que no se
 encuentra en el umbral definido. En caso de que el píxel se encuentre entre los umbrales
 establecidos, el píxel se pone a blanco.
  Scalar umbral_bajo = Scalar(0,48,80); //H,S,V
  Scalar umbral_alto = Scalar(179,150,255);

  //Se devuelve como resultado la imagen filtrada
  Mat salida;
  inRange(imag_hsv,umbral_bajo,umbral_alto,salida);
  imwrite(nombre.c_str(), salida);
  return salida;
}
```

El resultado obtenido, para algunas de las fotos de la base de datos de personas que hemos utilizado, es el siguiente (foto original a la derecha, y foto tras pasar el filtro de detección de piel a la izquierda):



Para la mayoría de las fotos de nuestra base de datos se obtienen resultados en general bastante aceptables. Sin embargo, esto tiene limitaciones, ya que con personas con tonos más oscuros de piel el filtro que utilizamos no detecta bien las zonas de piel (como se esperaba) y, además, algo importante es que al trabajar a nivel de píxel y sin tener en cuenta los gradientes, hace que la iluminación afecte bastante al resultado, como podemos apreciar en las siguientes imágenes:



Una vez tenemos la piel podemos calcular aproximadamente la zona en la que se encuentra la cara, por lo que en la imagen original creamos un marco negro alrededor de la zona en donde se encontraría ésta para descartar zonas de la imagen donde es seguro que no van a aparecer ojos. Para realizarlo lo hemos implementado de la siguiente forma:

```
Mat recortarCara(Mat im_piel, Mat original, int indice){

  ....

  //Buscamos horizontalmente (o a lo ancho) el primer píxel que se podría
  considerar piel
  for(int i=0; i < im_piel.cols/2 && !encontrado; ++i){
        color = im_piel.at<Vec3b>(Point(i, im_piel.rows/2));
        if(color.val[0] == 255 && color.val[1] == 255 && color.val[2] == 255){
              encontrado = true;
              x1 = i; }
  }
  encontrado = false;
  for(int i=im_piel.cols-1; i > im_piel.cols/2 && !encontrado; --i){
        color = im_piel.at<Vec3b>(Point(i, im_piel.rows/2));
        if(color.val[0] == 255 && color.val[1] == 255 && color.val[2] == 255){
              encontrado = true;
              x2 = i; }
  }
  encontrado = false;
  //Buscamos verticalmente (altura) el primer píxel que se podría considerar piel
  for(int i=0; i < im_piel.rows/2 && !encontrado; ++i){
        color = im_piel.at<Vec3b>(Point(im_piel.cols/2, i));
        if(color.val[0] == 255 && color.val[1] == 255 && color.val[2] == 255){
              encontrado = true;
              y1 = i;}
  }
  encontrado = false;
  for(int i=im_piel.rows-1; i > im_piel.rows/2 && !encontrado; --i){
        color = im_piel.at<Vec3b>(Point(im_piel.cols/2, i));
        if(color.val[0] == 255 && color.val[1] == 255 && color.val[2] == 255){
              encontrado = true;
              y2 = i;
        }
  }
  //Enmarcamos la cara con un marco negro
  cara_recortada = Mat::zeros(original.rows, original.cols, CV_8UC3);
  for(int i=y1; i < y2; ++i)
        for(int j=x1; j < x2; ++j)
              cara_recortada.at<float>(i,j) = original.at<float>(i,j);
  imwrite(nombre.c_str(), cara_recortada);
  return cara_recortada;

}
```

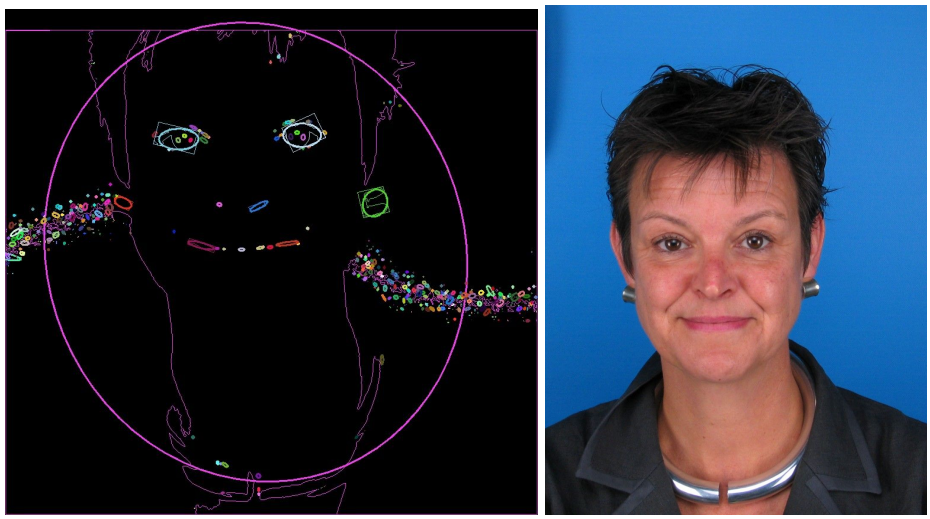
2. Detección de candidatos y elección → A partir de la imagen con la cara ya enmarcada, buscamos posibles contornos cerrados de forma circular o elipsoidal que serían los posibles candidatos para ser los ojos. Una vez tenemos todos los posibles candidatos, los comparamos en parejas para determinar si podrían ser posibles ojos o no. Para ello tenemos en cuenta que su posición sea más o menos centrada, que estén a la misma altura y a la misma distancia del centro de la imagen aproximadamente, es decir, que sean simétricos. Además de esto, comprobamos que el área de los contornos candidatos sea más o menos de un tamaño similar.

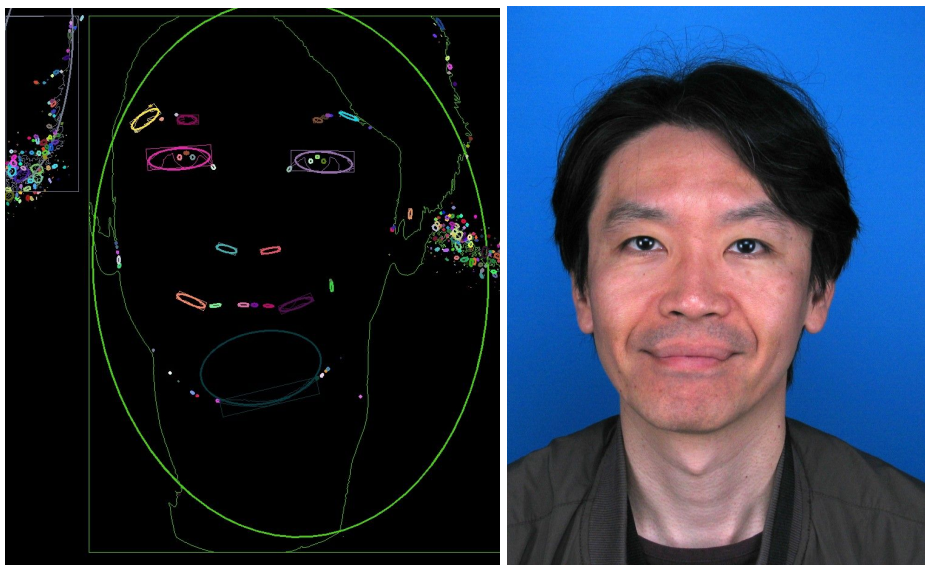
```
Mat deteccionOjos(Mat imag, int indice){

        ....

        //Detectamos la piel y recortamos la cara para facilitar la búsqueda
        Mat piel = deteccionPiel(imag, indice);
        copia_color = recortarCara(piel, imag, indice);
        cvtColor(copia_color,copia,CV_BGR2GRAY);

        //Buscamos los contornos en la imagen
        threshold(copia, imagen_contornos, 85, 255, THRESH_BINARY);
        findContours(imagen_contornos, contornos,  jerarquia, CV_RETR_TREE,
                        CV_CHAIN_APPROX_SIMPLE, Point(0,0));

        nombre = "/home/alumno/Escritorio/prep_contornos/";
        imwrite(nombre+convertir_numero(indice)+jpg, imagen_contornos);

        // Buscamos los rectángulos y elipses adecuados a cada contorno
         vector<RotatedRect> minRect(contornos.size());
         vector<RotatedRect> minEllipse(contornos.size());

         for(int i = 0; i < contornos.size(); i++){
                 minRect[i] = minAreaRect(Mat(contornos[i]));
                         if(contornos[i].size() > 5)
                                 minEllipse[i] = fitEllipse(Mat(contornos[i]));
                }

         // Dibujamos los contornos en una imagen junto con las elipses y rectángulos
         Mat contornos_finales = Mat::zeros(imagen_contornos.size(), CV_8UC3);
         for(int i = 0; i< contornos.size(); ++i){

                Scalar color = Scalar(rng.uniform(0,255),rng.uniform(0,255),rng.uniform(0,255));
                drawContours(contornos_finales,contornos,i,color,1,8,vector<Vec4i>(),0,Point());
                ellipse(contornos_finales,minEllipse[i],color,2,8);
                Point2f puntos_rec[4];
                minRect[i].points( puntos_rec);
                for(int j = 0; j < 4; j++)
                line(contornos_finales, puntos_rec[j], puntos_rec[(j+1)%4], color,1,8);
                }

         nombre = ruta+convertir_numero(indice)+jpg;
```

```
 imwrite(nombre.c_str(),contornos_finales);

 // Una vez tenemos todos los candidatos posibles, buscamos los mejores
        Mat salida = imag;
        Point2f centro_ojo1, centro_ojo2, centro_mejor1, centro_mejor2;
Size2f tam_1, tam_2, tam_mejor1, tam_mejor2;
int ojo_1, ojo_2, mejor_candidato1, mejor_candidato2;

mejor_candidato1 = 0;
mejor_candidato2 = 0;

centro_mejor1 = minEllipse[mejor_candidato1].center;
centro_mejor2 = minEllipse[mejor_candidato2].center;
tam_mejor1 = minEllipse[mejor_candidato1].size;
tam_mejor2 = minEllipse[mejor_candidato2].size;

//Recorremos la lista de candidatos y los comprobamos entre ellos
for(int i=0; i < minEllipse.size()-1; ++i){

        centro_ojo1 = minEllipse[i].center;
        tam_1 = minEllipse[i].size;
        for(int j=0; j < minEllipse.size(); ++j){

                centro_ojo2 = minEllipse[j].center;
                tam_2 = minEllipse[i].size;

                int distancia_x_centro_ojo1 = abs(centro_ojo1.x - imag.cols/2);
                int distancia_x_centro_ojo2 = abs(centro_ojo2.x - imag.cols/2);
                int distancia_y_centro_ojo1 = abs(centro_ojo1.y - imag.rows/2);
                int distancia_y_centro_ojo2 = abs(centro_ojo2.y - imag.rows/2);
                int distancia_x_centro_mejor1 = abs(centro_mejor1.x - imag.cols/2);
                int distancia_x_centro_mejor2 = abs(centro_mejor2.x - imag.cols/2);
                int distancia_y_centro_mejor1 = abs(centro_mejor1.y - imag.rows/2);
                int distancia_y_centro_mejor2 = abs(centro_mejor2.y - imag.rows/2);
                int distancia_y_ojos = abs(centro_ojo1.y-centro_ojo2.y);
                bool candidatos_mitad_superior = centro_ojo1.y <= imag.rows/2 &&
               centro_ojo2.y <= imag.rows/2;
                bool candidatos_mitad_derecha_izquierda = (centro_ojo1.x >=
               imag.cols/2 && centro_ojo2.x <= imag.cols/2) || (centro_ojo1.x <=
               imag.cols/2 && centro_ojo2.x >= imag.cols/2);
                bool mejor_candidato1_mas_centrado_x = distancia_x_centro_ojo1 <
               distancia_x_centro_mejor1;
                bool mejor_candidato2_mas_centrado_x = distancia_x_centro_ojo2 <
               distancia_x_centro_mejor2;
                bool mejor_candidato1_mas_centrado_y = distancia_y_centro_ojo1 <
               distancia_y_centro_mejor1;
                bool mejor_candidato2_mas_centrado_y = distancia_y_centro_ojo2 <
               distancia_y_centro_mejor2;
                bool alturas_similares = distancia_y_ojos <= 5;
                bool misma_altura = centro_ojo2.y == centro_ojo1.y;
                bool distancias_centro_similares = abs(distancia_x_centro_ojo1 -
```

```
                    distancia_x_centro_ojo2) <= 100;
                    bool areas_similares = abs(tam_1.area()-tam_2.area()) < 5;
                    bool area1_correcta = tam_1.area() >= 600 && tam_1.area() <= 3500;
                    bool area2_correcta = tam_2.area() >= 600 && tam_2.area() <= 3500;

                    //CONDICIÓN PARA POSIBLES CANDIDATOS A OJOS: Si los
                    candidatos están a la misma altura por encima de la mitad de la
                    imagen, cada uno en una mitad derecha o izquierda, más o menos
                    a la misma distancia del centro de la imagen y el área de ambos
                    candidatos es similar los seleccionamos
                    if(i != j
                    && candidatos_mitad_superior
                    && candidatos_mitad_derecha_izquierda
                    && mejor_candidato1_mas_centrado_x
                    && mejor_candidato2_mas_centrado_x
                    && mejor_candidato1_mas_centrado_y
                    && mejor_candidato2_mas_centrado_y
                    && (alturas_similares || misma_altura)
                    && distancias_centro_similares
                    && area1_correcta
                    && area2_correcta
                    && areas_similares){

                            mejor_candidato1 = i;
                            mejor_candidato2 = j;

                            centro_mejor1 = minEllipse[mejor_candidato1].center;
                            centro_mejor2 = minEllipse[mejor_candidato2].center;
                            tam_mejor1 = minEllipse[mejor_candidato1].size;
                            tam_mejor2 = minEllipse[mejor_candidato2].size;
                    }
            }
    }

    //Dibujamos dos círculos en las zonas seleccionadas para ser los ojos
    Scalar color = Scalar( rng.uniform(0, 255), rng.uniform(0,255), rng.uniform(0,255) );
    circle(salida, centro_mejor1,40, Scalar(0,255,0),3);
    circle(salida, centro_mejor2,40, Scalar(0,255,0),3);

    return salida;
}
```

El resultado de los contornos obtenidos, para algunas de las fotos de la base de datos de personas, es el siguiente (foto original a la derecha, y foto contornos a la izquierda):



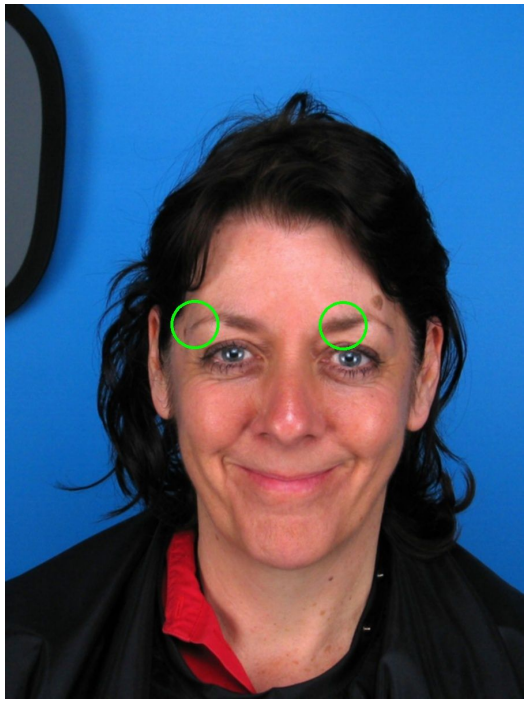


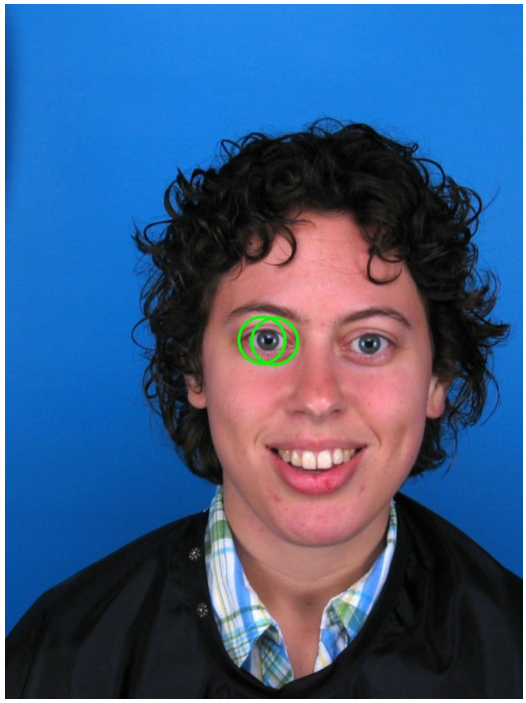
3. Experimentos realizados y resultados definitivos

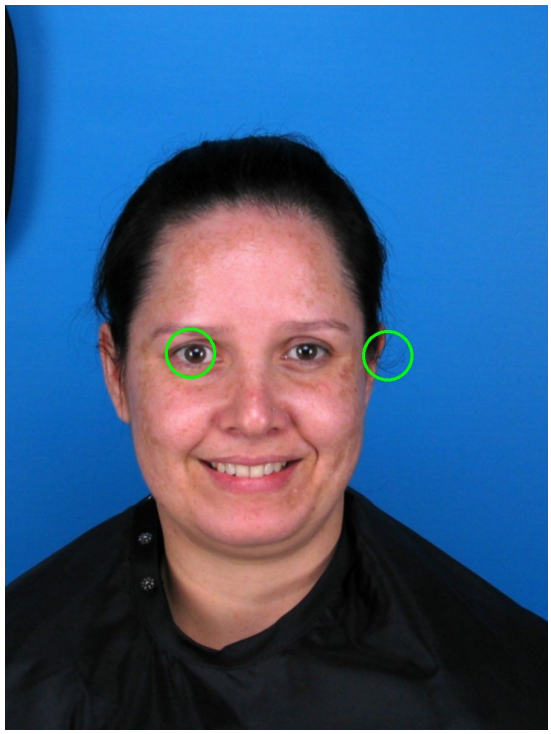
Hemos utilizado una base de datos de 131 imágenes en color en la que se muestran los rostros de personas sobre un fondo azul. En las imágenes hay mujeres y hombres de distintas regiones del mundo y con distintos tonos de piel, lo cual nos ha dificultado poder encontrar un filtro para la piel válido para los distintos casos. Además, de cada individuo se muestran varias fotos con varias expresiones faciales (serios, sonriendo...) con lo que la forma de los ojos varía en cada foto según la expresión.

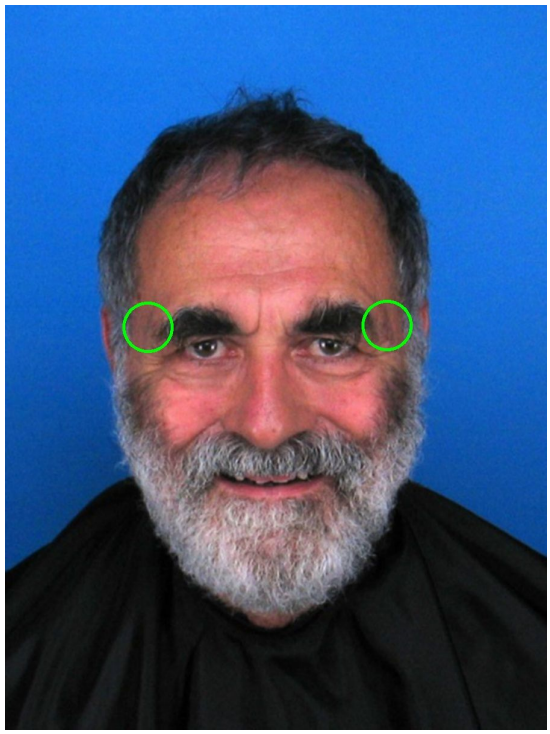
Se obtienen buenos resultados en 79 de 131 imágenes (en torno al 60%). Entre ellos se encuentran los siguientes:

Sin embargo, hay resultados que no salen bien:









Hemos añadido un anexo al final para mostrar más resultados.

4. Valoración de resultados y conclusiones

El problema que hemos abarcado es un problema más complejo de lo que pueda parecer porque influyen muchas variables: el color de la piel, la forma del rostro, la iluminación, la expresión de la cara, etc. Para poder trabajar mejor aislamos nuestro problema a encontrar los ojos en imágenes en la que se muestra el rostro de una persona sobre un fondo monocromático.

Nuestro enfoque se ha basado en la búsqueda de los ojos a partir de la detección de piel en la imagen y contornos entorno a ella. Esto también añade otra dificultad que existen contornos que cumplen las condiciones de los ojos sin serlo, como las cejas o la nariz, provocando falsos positivos. Además tenemos que añadir el hecho de que no existe un filtro para la piel que sea universal, por lo que para unos tonos de piel nuestro algoritmo funcionará correctamente mientras que en otros casos nos darán errores.

De los experimentos que hemos realizado obtuvimos un 60% de acierto, encontrando ambos ojos correctamente. Del 40% restante, en algunas ocasiones sólo se localizó un ojo o las cejas, lo cual consideramos que no es un error tan grave, mientras que en otros casos simplemente no se localizaron.

Podemos concluir que los resultados obtenidos no son tan buenos como esperábamos, quizá debido al enfoque que hemos utilizado. Hay otros enfoques basados en los cambios de gradiente alrededor de la zona de los ojos empleando técnicas de aprendizaje automático que probablemente den resultados mejores al combinar ambas técnicas de visión por computador y aprendizaje automático.

5. Referencias

1. http://ai2-s2-pdfs.s3.amazonaws.com/ca75/55e9ee231cabc3191ddf55372c1d66926f0a.pdf
2. http://www.mil.ufl.edu/publications/fcrar09/Eye_Detection_Tanmay_Rajpathak_fcrar_09.pdf
3. https://hal-ujm.archives-ouvertes.fr/ujm-00374366/document
4. http://docs.opencv.org/trunk/df/d9d/tutorial_py_colorspaces.html
5. http://answers.opencv.org/question/3300/skin-detection/
6. http://www.academia.edu/5704209/Skin_Detection_for_Face_Recognition_Based_on_HSV_Color_Space
7. http://www.wseas.us/e-library/conferences/2010/Vouliagmeni/CSECS/CSECS-01.pdf
8. http://bytefish.de/blog/opencv/skin_color_thresholding/
9. https://www.codeproject.com/Articles/808416/Adaptive-Skin-Color-Detection
10. https://www.researchgate.net/publication/228629812_Skin_segmentation_using_multiple_thresholding_-_art_no_60610F
11. http://docs.opencv.org/2.4/doc/tutorials/imgproc/shapedescriptors/find_contours/find_contours.html
12. http://docs.opencv.org/2.4/doc/tutorials/imgproc/shapedescriptors/bounding_rotated_ellipses/bounding_rotated_ellipses.html

Anexo I: Resultados