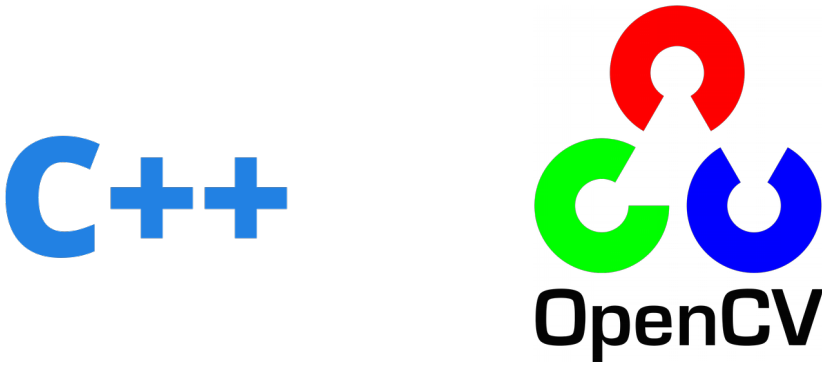
UNSA - CS - ARQUITECTURA DEL COMPUTADOR
Profesor: Pedro Rodríguez Gonzales
Kevin Salazar | 20162013

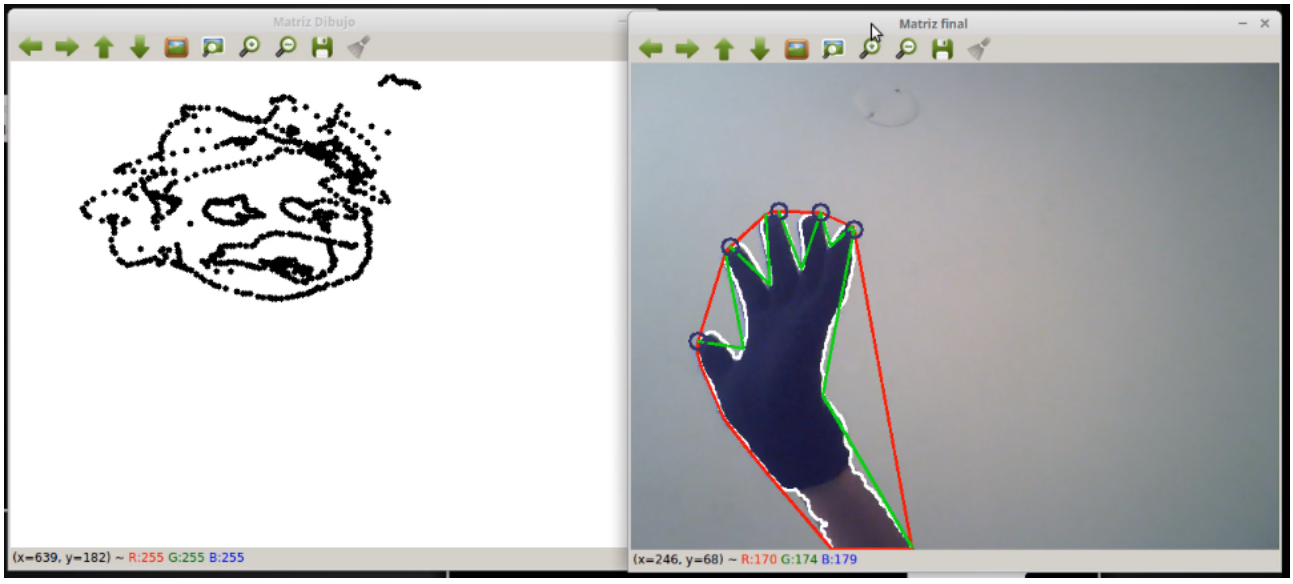
Seguimiento de la mano en tiempo real para dibujar en pantalla usando la cámara web

El presente trabajo del curso lleva como objetivo detectar los dedos, específicamente las yemas, para poder realizar dibujos libremente en nuestra pantalla; generalmente las laptops traen una cámara integrada que gracias a su ubicación permite que al registrar en video nuestra mano se pueda dibujar poniendo esta justo al frente de la pantalla, dando una interacción muy buena con nuestro dibujo.

La programación de esta tarea se ha realizado en C++, usando OpenCV, una librería gráfica de Intel.

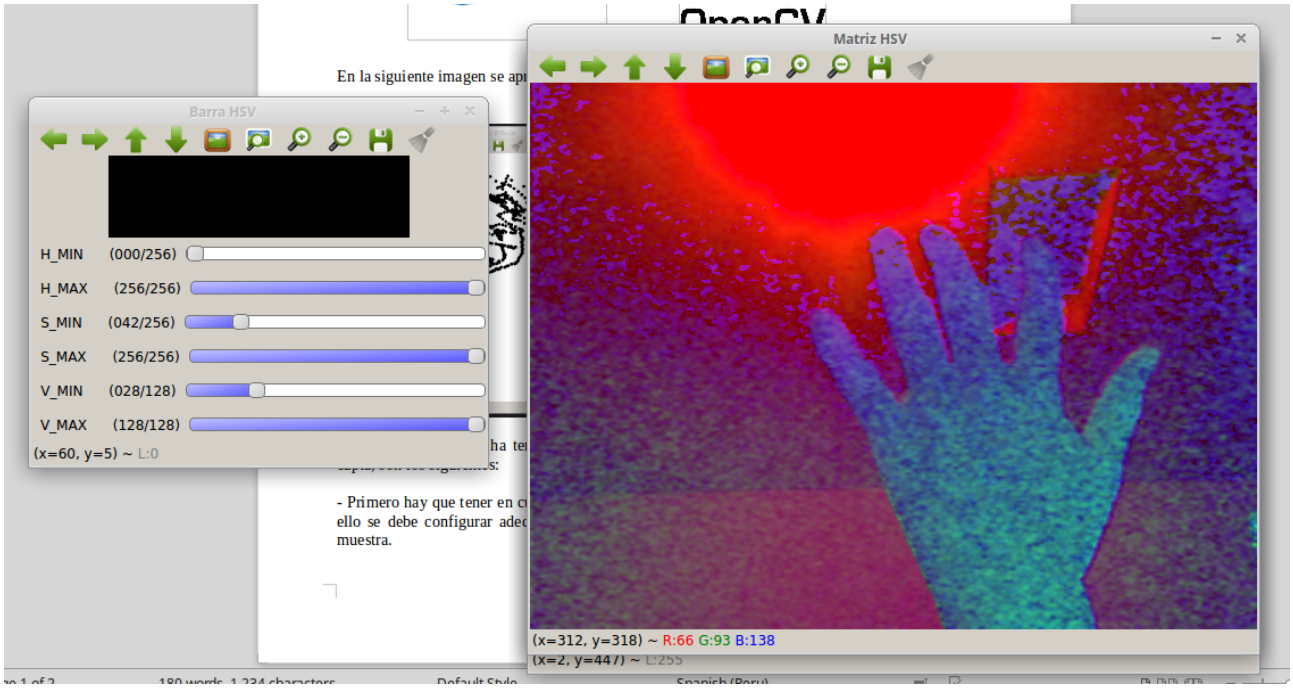


En la siguiente imagen se aprecia un dibujo de cómic hecho con el proyecto ya culminado:

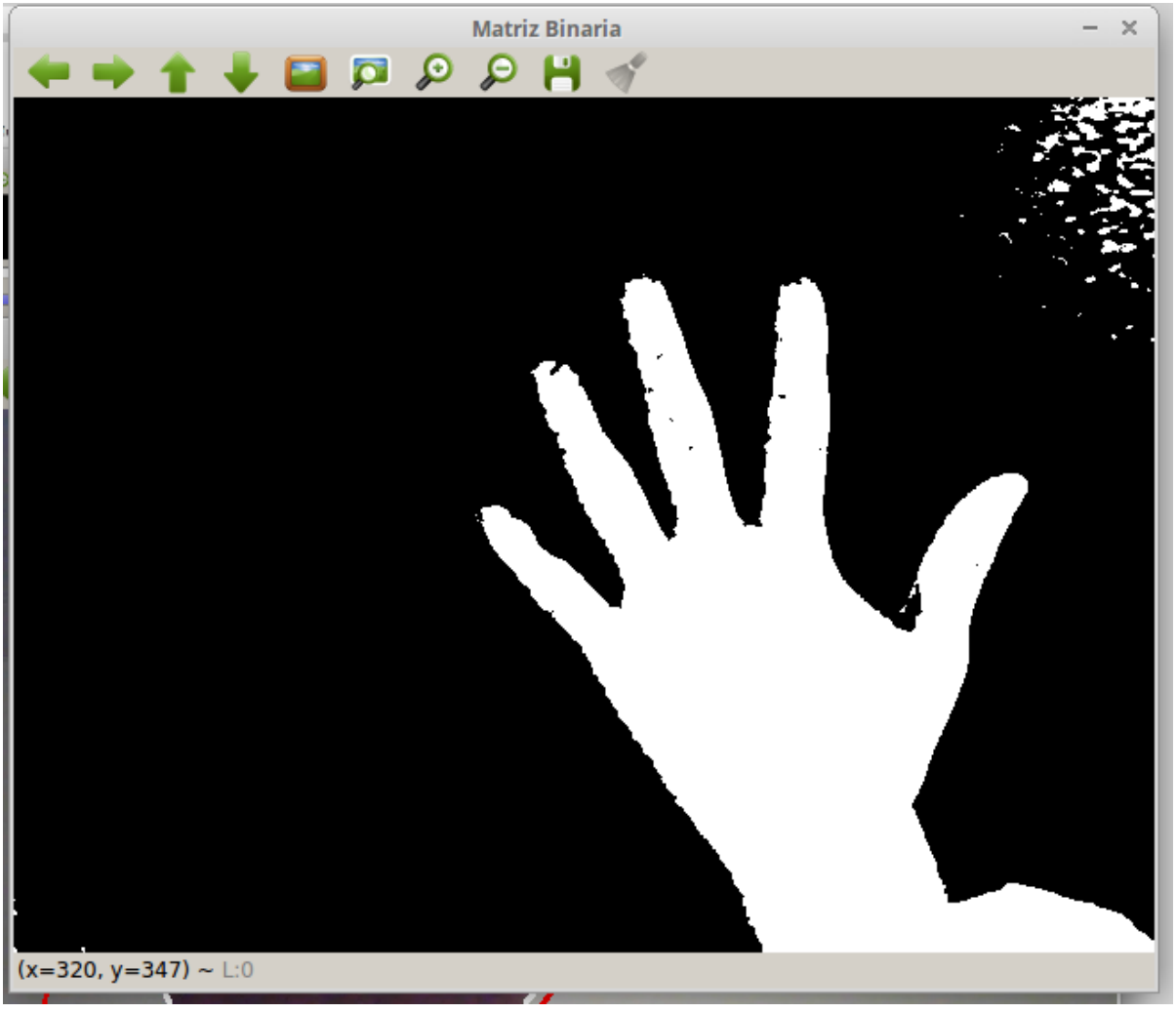


Para lograr dibujar se ha tenido que atravesar ciertos procesos en la imagen que la cámara web capta, son los siguientes:

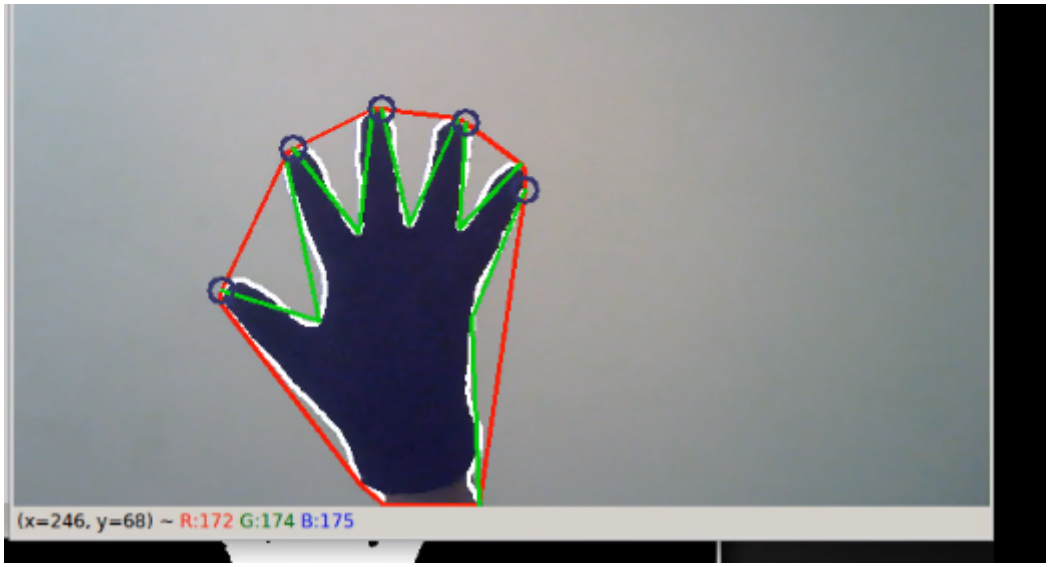
- Primero hay que tener en cuenta que se debe conseguir solamente la silueta de nuestra mano, para ello se debe configurar adecuadamente la barra HSV (matiz, saturación, valor) que el programa, al ejecutarse, muestra.



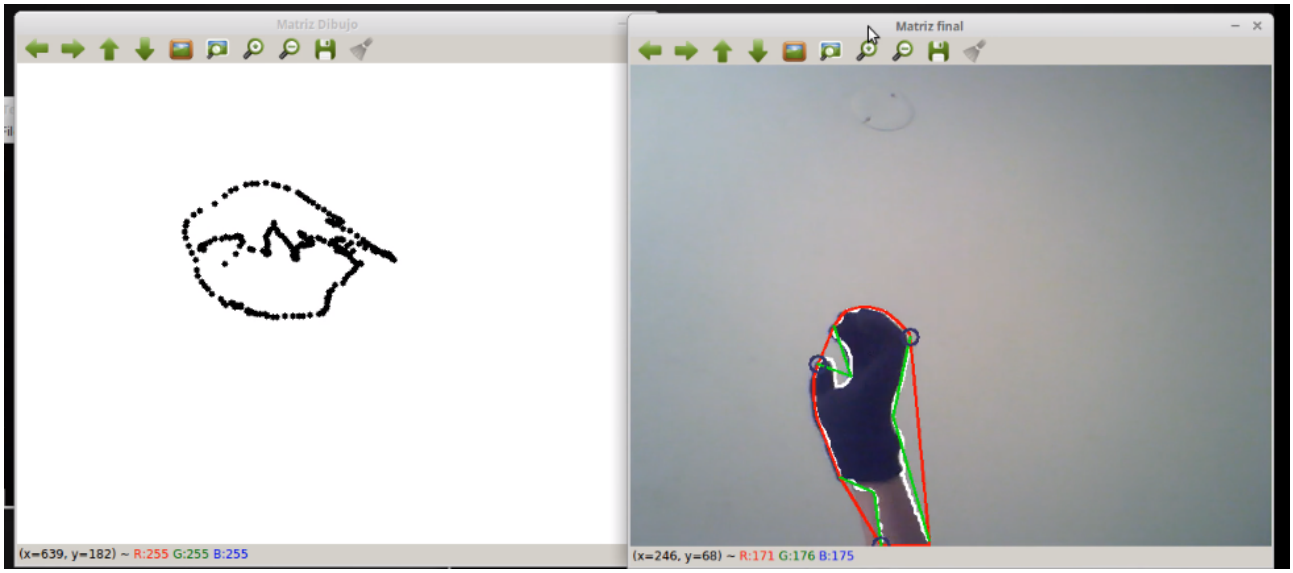
Se muestra la barra configurada y la ventana HSV generada, internamente esta ventana es una matriz de puntos que puede ser transformada a una matriz binaria de dos colores (blanco y negro).



Con esta nueva matriz se puede trazar un contorno a la mano (líneas de color rojo) y dibujar línea verdes a las partes cóncavas.
De esta manera se dibujan círculos azules en la yema de los dedos.

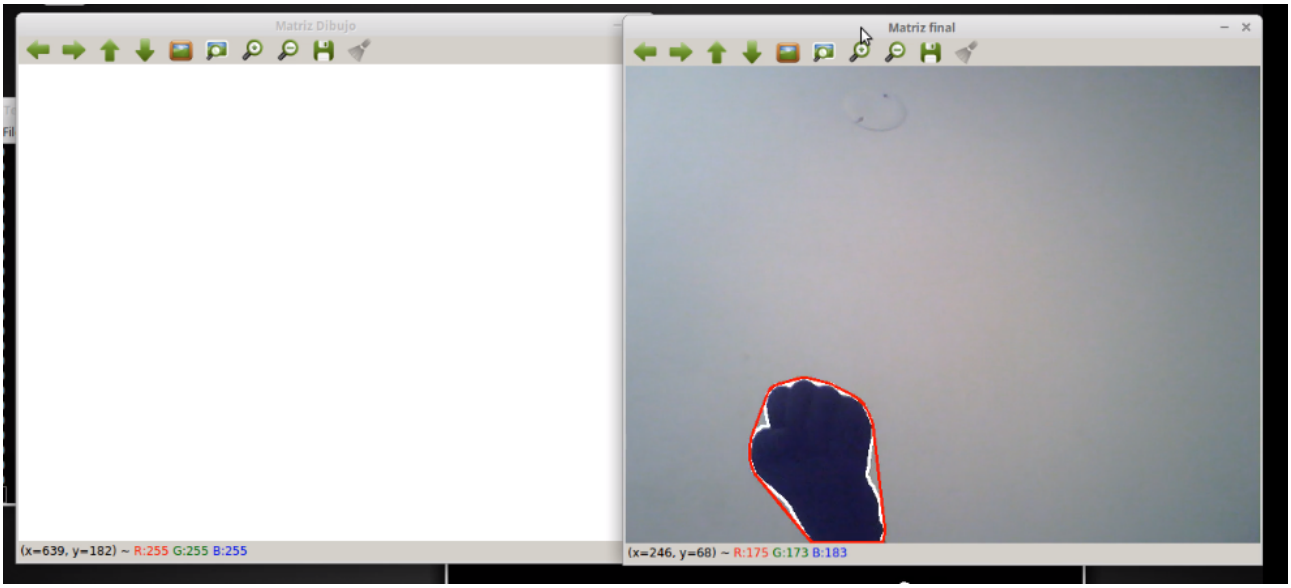


Para llegar hasta este punto se ha encontrado variada documentación en internet.
La forma para conseguir dibujar en otra ventana con estos círculos que se han creado al detectar los dedos es una idea genuina. Consiste en almacenar en vectores los puntos de esta matriz que representan los círculos para luego determinar un único punto representante de los cinco leídos, este punto se usará como lápiz para dibujar en una nueva ventana. Una vez conseguido este único punto el dibujar se logra gracias a copiar la ubicación de este y recrearlo en la ventana, la detección del punto se repite en un bucle (por lo que la replicación se da en tiempo real) bajo ciertas variables que permiten mover el “lápiz” o borrar lo dibujado. Estas variables son dos, la primera establece que si no se detectan exactamente cinco dedos no se seguirá dibujando (permite desplazar el “lápiz” en la pantalla sin dibujar) y la otra que si no se detectan dedos se borre todo lo dibujado.



Se aprecia que no se están detectando cinco dedos, sino dos; por lo tanto, podemos desplazar nuestro lápiz sin dibujar en la ventana.

Al no detectarse dedos todo lo dibujado se borra.



Notar que al ejecutar el programa se generan cinco ventans: del la imagen capturada, de la imagen HSV, de la imagen binaria, de la imagen con lineas rojas y verdes, y la ventana donde se dibuja. Para culminar el programa se presiona la tecla Esc.

A continuación el código:

```
// `pkg-config --cflags --libs opencv` , se usa al momento de compilar

#include <iostream>

#include <opencv/cv.h>
#include <opencv/highgui.h>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/video/background_segm.hpp>
#include <opencv2/opencv.hpp>


using namespace std;
using namespace cv;

// configuración global

const string nombre_barraHSV = "Barra HSV"; // Valor de Saturación de Matiz(Hue)

/*
int H_MIN = 0;
int H_MAX = 256;
int S_MIN = 0;
```

```
int S_MAX = 256;
int V_MIN = 0;
int V_MAX = 256;
*/
int H_MIN = 0;
int H_MAX = 256;
int S_MIN = 42;
int S_MAX = 256;
int V_MIN = 28;
int V_MAX = 128;

void createTrackbars() //creamos la ventana para configurar las barras
{
   namedWindow(nombre_barraHSV,0);
   createTrackbar( "H_MIN", nombre_barraHSV, &H_MIN, H_MAX);
   createTrackbar( "H_MAX", nombre_barraHSV, &H_MAX, H_MAX);
   createTrackbar( "S_MIN", nombre_barraHSV, &S_MIN, S_MAX);
   createTrackbar( "S_MAX", nombre_barraHSV, &S_MAX, S_MAX);
   createTrackbar( "V_MIN", nombre_barraHSV, &V_MIN, V_MAX);
   createTrackbar( "V_MAX", nombre_barraHSV, &V_MAX, V_MAX);
}

int indice_mayor_contorno(vector<vector<Point> >contornos);

int main()
{
   VideoCapture webcam(0);  // iniciamos la cámara integrada

     Mat matriz_1(Size(640, 480),CV_8UC3); // matriz de unsigned int de 8 bits y de 3 canales (RGB)
   Mat matriz_2(Size(640, 480),CV_8UC3);
   Mat matriz_3(Size(640, 480),CV_8UC3, CV_RGB(255,255,255));
   Mat matriz_2_f(Size(640, 480),CV_8UC3);
   Mat matriz_3_f(Size(640, 480),CV_8UC3);


   createTrackbars(); // creamos la barraHSV

   if(webcam.isOpened())
      do{
         webcam >> matriz_1; // llenamos la matriz con los datos de la cámara
         webcam >> matriz_2;

         Size kernel_tam;
                  kernel_tam.height = 3;
                  kernel_tam.width = 3;
                  double sigma = 0.3*(3/2 - 1) + 0.8; // según documentación
                   GaussianBlur(matriz_1,matriz_1,kernel_tam,sigma,0.0,4);  // reducimos ruido de la captura

         Mat matriz_hsv(Size(640, 480),CV_8UC3);
```

```
            cvtColor(matriz_1,matriz_hsv,CV_RGB2HSV); // matriz_hsv ahora contiene matriz_1 el canal que HSV buscamos

            Mat matriz_binaria(Size(640, 480),CV_8UC1); // 1 solo canal (binario)
             inRange(matriz_hsv,Scalar(H_MIN,S_MIN,V_MIN),Scalar(H_MAX,S_MAX,V_MAX),matriz_binaria); // en la matriz_binaria los pixeles blancos representan los elementos dentro del rango y los negros fuera del rango

            Mat Erode(Size(640, 480),CV_8UC1);
            erode(matriz_binaria, Erode, Mat(), Point(-1,-1));

            Mat matrizb_dilatada(Size(640, 480),CV_8UC1);
             dilate(Erode, matrizb_dilatada, Mat(), Point(-1,-1),2); // dilatar significa añadir pixeles a los bordes de la imagen

            vector<Vec4i>  hierarchy; //jerarquía de los contornos detectados
            vector<vector<Point> > contornos; //matriz que contiene los puntos de controno en la matriz image

                        findContours(matrizb_dilatada.clone(), contornos, hierarchy, CV_RETR_TREE , CV_CLOCKWISE, Point(0, 0)); // toma la matriz dilatada y guarda los contornos en las variables correspondientes

            //hallando los datos necesarios para hacer el dibujo del poligono del contorno de las manos
            if(contornos.size() > 0){
                  vector<vector<Point> > contornos_(contornos.size()); //contornos_ contendra todos los contornos presentes en forma de data
                              vector<vector<Vec4i> > defectos(contornos.size()); // defectos contendrá los "hundimientos" de los contornos

                  vector<vector<int> > contornos__int(contornos.size());

                  //buscamos el contorno perteneciente a las manos (el mayor)
                  int indice_manos = indice_mayor_contorno(contornos);

                     //ahora dibujaremos las lineas de contorno basados es los datos de contornos y de las jerarquias de estos

                  for(int i = 0; i != contornos.size(); i++){
                     convexHull(Mat(contornos[i]), contornos_[i], false);//convexHull guarda los contornos presentes como data
                     convexHull(Mat(contornos[i]), contornos__int[i], false);
                           convexityDefects(Mat(contornos[i]), contornos__int[i], defectos[i]);// llenamos los datos de defectos
                     if(indice_manos == i){
                           drawContours(matriz_2, contornos, indice_manos, CV_RGB(255,255,255), 2, 8, hierarchy, 0, Point());//2 maneja el color y 8 que tan delgada es la linea, 0 es el maximo nivel de contornos.
                              drawContours(matriz_2, contornos_, indice_manos, CV_RGB(255,0,0), 2, 8, hierarchy, 0, Point()); //el anterior dibuja el contorno de las manos, y este dibuja un contorno_ poligonal de las manos pues toma los datos hallados antes
                     }
```

```
        }

            //ahora dibujaremos los defectos o "hundimientos" que se marcan al comparar ambos contornos, estos defectos viene a ser el espacio entre los dedos
        int indice_inicio;
        Point punto_inicio;
        int indice_final;
        Point punto_final;
        int indice_entre_dedos;
        Point punto_entre_dedos;

        Point arr[5];

        Point p;
        int cuenta = 0;
        int time = 0;
        for(int i = 0; i != contornos.size(); i++){
              if (contornos[i].size() < 300) continue;// discriminamos los contornos inadecuados, cuyas matrices sean muy pequeñas
            vector<Vec4i>::iterator it = defectos[i].begin();

            while(it != defectos[i].end()){
               Vec4i& defecto = (*it);
               if(indice_manos == i){
                  indice_inicio = defecto[0];
                  punto_inicio.x = contornos[i][indice_inicio].x;
                  punto_inicio.y = contornos[i][indice_inicio].y;


                  indice_final = defecto[1];
                  punto_final.x = contornos[i][indice_final].x;
                  punto_final.y = contornos[i][indice_final].y;

                  indice_entre_dedos = defecto[2];
                  punto_entre_dedos.x = contornos[i][indice_entre_dedos].x;
                  punto_entre_dedos.y = contornos[i][indice_entre_dedos].y;
                   float distancia = defecto[3] / 256; // distancia entre el contorno y entre_dedos, 256 tomado de documentacion

                     if(distancia > 20 && distancia < 80){ //distancia razonable, segun numeros de documentacion
                     line(matriz_2, punto_inicio, punto_entre_dedos, CV_RGB(0,255,0), 2);
                     line(matriz_2, punto_final, punto_entre_dedos, CV_RGB(0,255,0), 2);
                     circle(matriz_2, punto_inicio, 8, Scalar(110,50,50), 2);
                     arr[cuenta] = punto_inicio;
                     cuenta++;

                  }
                  time++;
                  cout << cuenta << '\n';
                  if (cuenta == 0 && time >= 20)
                     matriz_3.setTo(Scalar(255,255,255));
```

```
                    if (cuenta == 5){
                        p.x = (arr[1].x + arr[2].x + arr[3].x)  / 3;
                        p.y = (arr[1].y + arr[2].y + arr[3].y) / 3;
                        cout << "x: " << p.x << ", y: " << p.y << '\n';
                        line(matriz_3, p, p, Scalar(0,0,0), 5);
                    }
                }
                it++;
            }
        }

        line(matriz_3, p, p, Scalar(0,0,0), 5);
        imshow("Captura", matriz_1); // mostramos la matriz
        imshow("Captura dilatada", matrizb_dilatada); // mostramos la matriz
                        imshow("Matriz HSV", matriz_hsv);
                        imshow("Matriz Binaria", matriz_binaria);
                        flip(matriz_2, matriz_2_f, 1);
                        flip(matriz_3, matriz_3_f, 1);
                        imshow("Matriz final", matriz_2_f);
                        imshow("Matriz Dibujo", matriz_3_f);
    }

    }while(waitKey(1) != 27);// muestra la matriz por 10 ms mientras la tecla presionada sea diferente a Esc(27)

  else
    cerr << "No se pudo abrir la cámara =(\n";

  return 0;
}

int indice_mayor_contorno(vector<vector<Point> >contornos)
{
  int indice = -1;
  int tam_contorno = 0;
  int tam;
  for(int i=0; i!=contornos.size(); i++){
    tam = contornos[i].size();
    if( tam > tam_contorno){
      tam_contorno = tam;
      indice = i;
    }
  }
  return indice;
}
```

Finalmente, otro dibujo.