Reconocimiento de gestos con OpenCV (Open Source Computer Vision Library)

Sistemas de interacción persona-computador

Definición del problema

- El reconocimiento de gestos permite a los seres humanos interactuar naturalmente con una máquina sin usar dispositivos físicos de entrada.

Metodología

- Substracción del fondo
  - Eliminamos el fondo para obtener una región de interés que se corresponda con la mano
- Detección de características de la mano
  - Detección del contorno de la mano
  - Detección de la posición de los dedos de la mano.

➢Para llevar a cabo estas tareas usaremos OpenCV

¿Qué es OpenCV?

- Es una librería de visión artificial y aprendizaje automático
- Se diseñó para acelerar la implantación de sistemas automáticos de percepción en productos comerciales.
- Dispone de más de 2500 algoritmos (Reconocimiento de caras, identificación de objetos, clasificación de comportamiento humano en vídeos, seguimiento de objetos móviles, etc…)
- Tiene interfaces en C, C++, Python, Java y MATLAB y soporta Windows, Linux, Android y MAC OS

La clase Mat

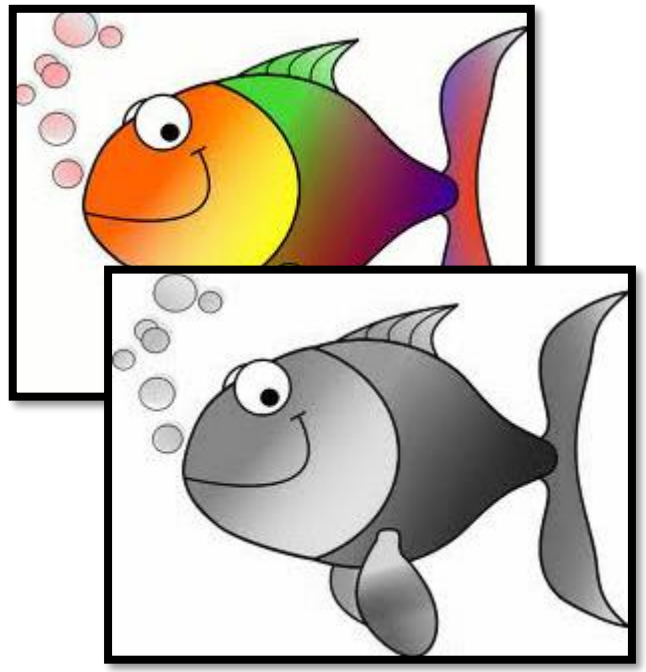
- Es la estructura básica para almacenar imágenes.

```
cv::Mat m(2, 2, CV_8UC3, cv::Scalar(0, 0, 255));
std::cout << "M = " << std::endl << " " << m << std::endl;
```

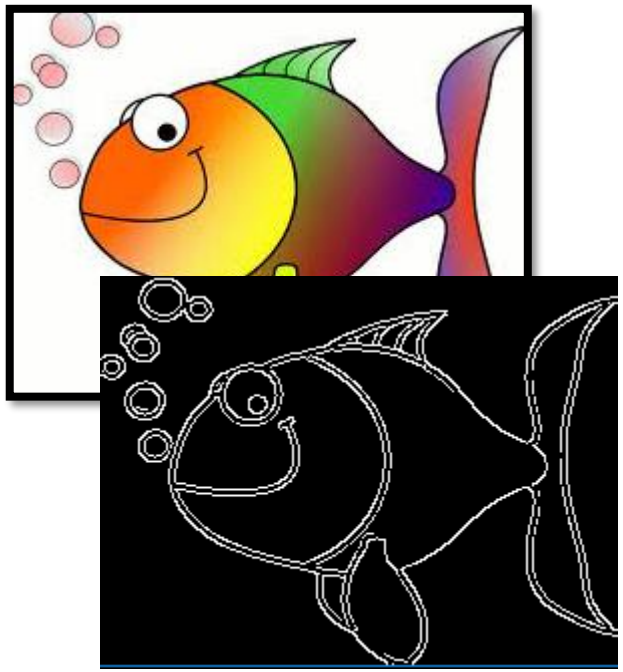
```
M =
 [0, 0, 255, 0, 0, 255;
  0, 0, 255, 0, 0, 255]
```

Cargar, modificar y guardar una imagen…

```
int main(int argc, char** argv)
{
    char* imageName = argv[1];

    Mat image;
    image = imread(imageName, 1);

    if (argc != 2 || !image.data)
    {
        printf(" No image data \n ");
        return -1;
    }

    Mat gray_image;

    // convertimos la imagen en color a escala de grises
    cvtColor(image, gray_image, CV_BGR2GRAY);

    //guardamos la imagen
    imwrite("Gray_Image.jpg", gray_image);

    //creamos las ventanas para mostrar las imágenes
    namedWindow(imageName, CV_WINDOW_AUTOSIZE);
    namedWindow("Gray image", CV_WINDOW_AUTOSIZE);

    //mostramos las imágenes
    imshow(imageName, image);
    imshow("Gray image", gray_image);

    //esperamos a que el usuario pulse una tecla
    waitKey(0);

    return 0;

}
```



Detección de bordes



```
int main(int argc, char** argv)
{
    /// Cargar la imagen
    src = imread(argv[1]);

    if (!src.data)
    {
        return -1;
    }

    /// Convertir la imagen a escala de grises
    cvtColor(src, src_gray, CV_BGR2GRAY);

    /// Crear una ventana
    namedWindow(window_name, CV_WINDOW_AUTOSIZE);

    /// Crear un trackbar para el umbral del detector de bordes Canny
    createTrackbar("Min Threshold:", window_name, &lowThreshold, max_lowThreshold, CannyThreshold);

    /// Detectar los bordes y mostrar
    CannyThreshold(0, 0);

    /// Esperar a que el usuario presione una tecla
    waitKey(0);

    return 0;
}
```

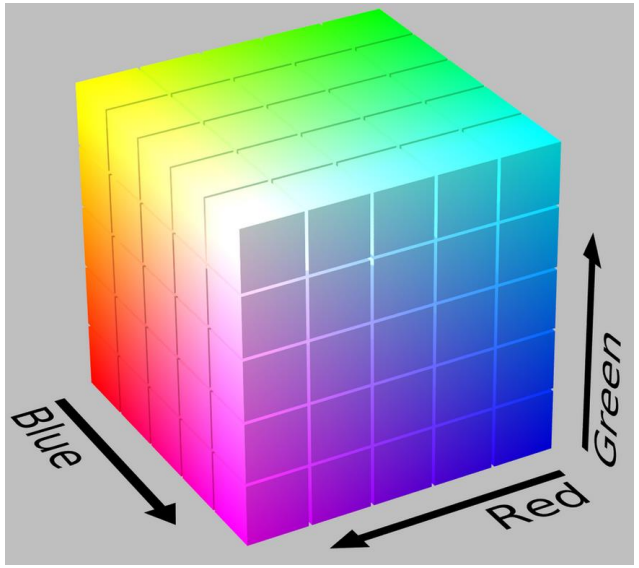
Detección de bordes

```
void CannyThreshold(int, void*)
{
    /// Detector de bordes Canny
    Canny(src_gray, detected_edges, lowThreshold, lowThreshold*ratio, kernel_size);

    imshow(window_name, detected_edges);
}
```

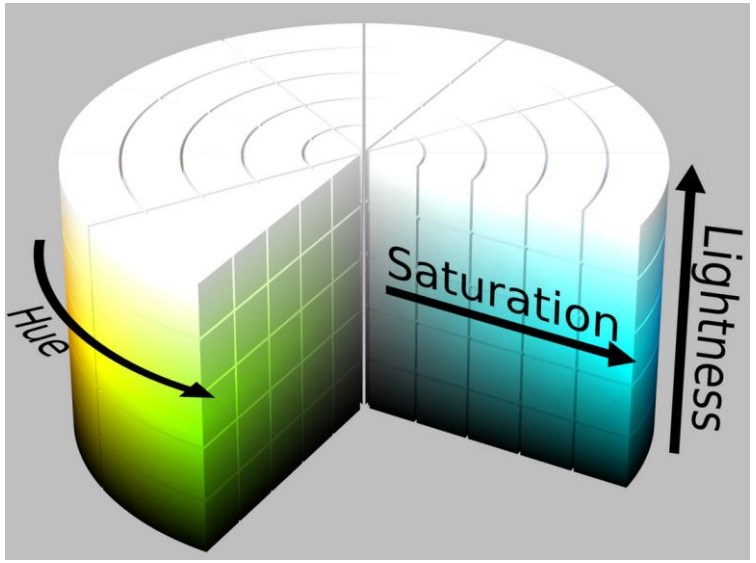
Espacios de color

```
cvtColor(frame, hls_frame, CV_BGR2HLS);
```

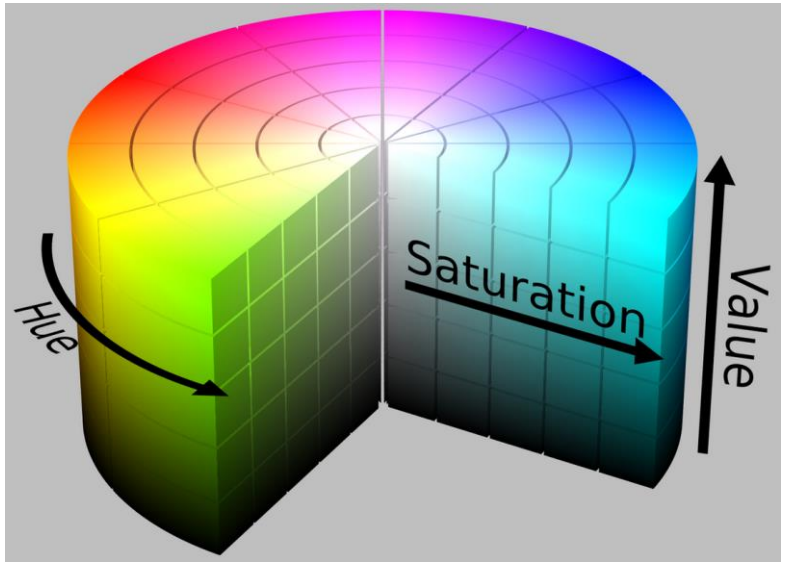
RGB



HLS

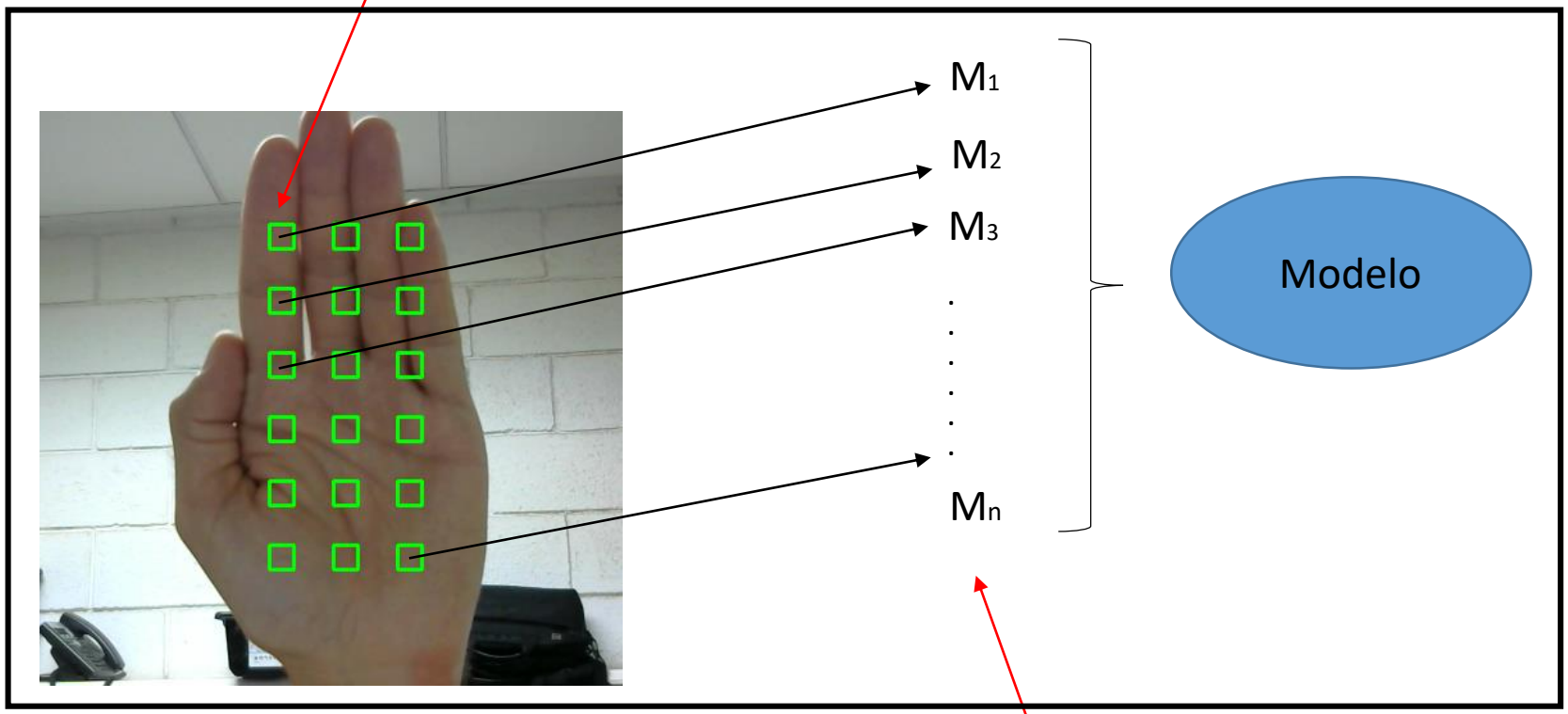


HSV



Substracción de fondo

```
Mat roi = img(Rect(x, y, w, h));
```

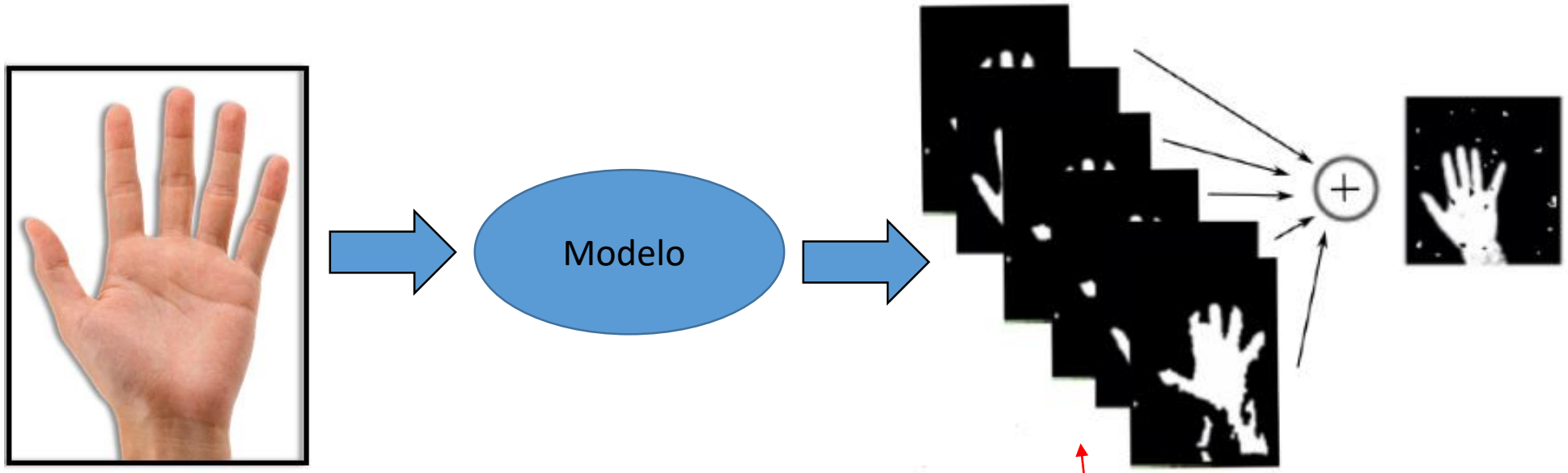



```
Scalar m = mean(roi);
```

Código 1.1

- Convertir la imagen de entrada a HLS
- Para cada muestra
  - Obtener una región de interés de la imagen HLS.
  - Calcular la media de la región de interés

Substracción de fondo




```
inRange(src, low, up, dst);
```

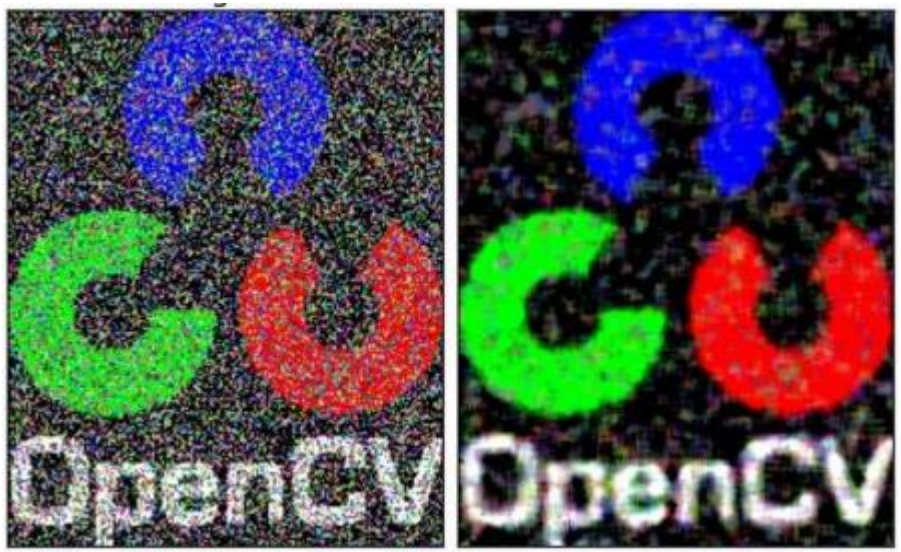
Código 1.2

- Convertir la imagen de entrada a HLS
- Para cada media
  - Obtener el rango, comprobando que esté en [0,255].
  - Crear una imagen binaria con los puntos que estén dentro del rango.
  - Ir sumando las imágenes binarias.

Reducir el ruido

- Filtro de la mediana

```
medianBlur(bgmask, bgmask, 5);
```



Reducir el ruido

- Operaciones morfológicas
  - Dilatación

```
Mat element = getStructuringElement(MORPH_RECT,
    Size(2 * dilation_size + 1, 2 * dilation_size + 1),
    Point(dilation_size, dilation_size));
```

```
dilate(src, dst, element);
```

  - Erosión

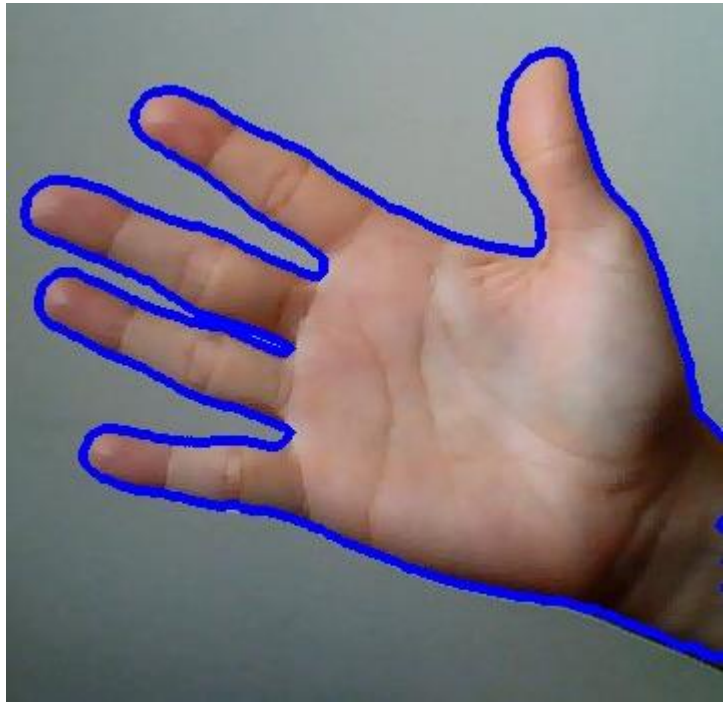
```
erode(src, dst, element);
```

Código 2.1

- Aplicar operaciones morfológicas y/o filtro de la mediana para limpiar el ruido de la imagen antes de detectar la mano

Detección del contorno de la mano

```
findContours(img, contours, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE);
```

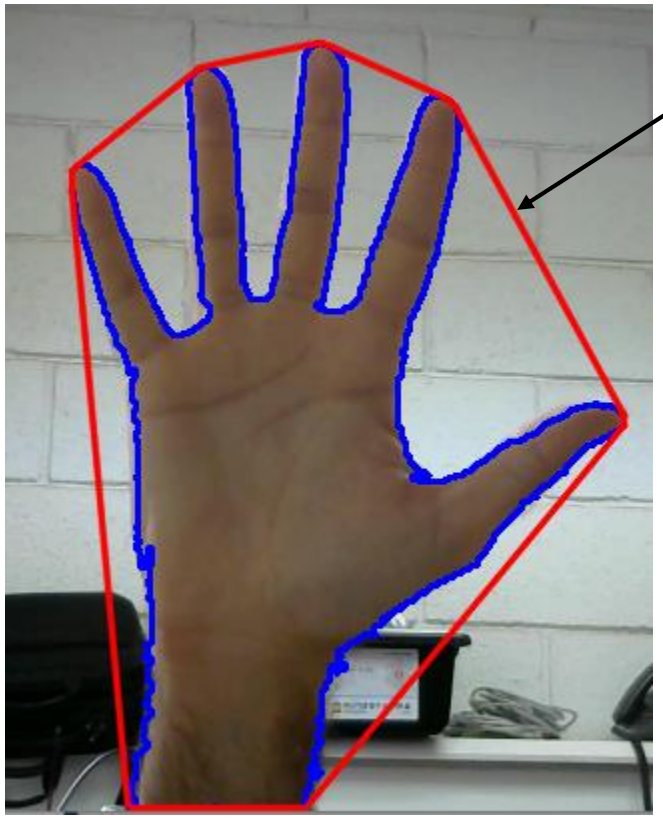


```
drawContours(output_img, contours, index,
                cv::Scalar(255, 0, 0), 2, 8,
                vector<Vec4i>(), 0, Point());
```

Código 3.1

- Detección de los contornos de la imagen
- Obtener el contorno más largo
- Mostrar el contorno más largo

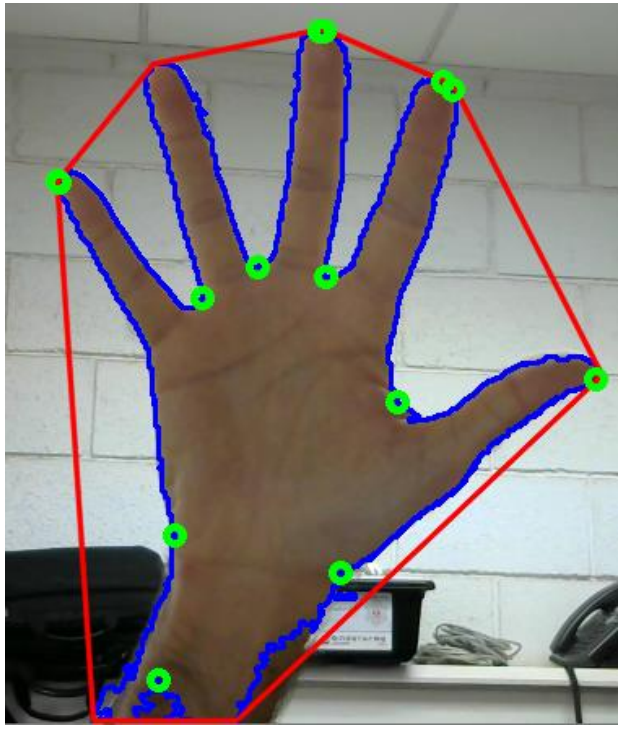
Convex hull



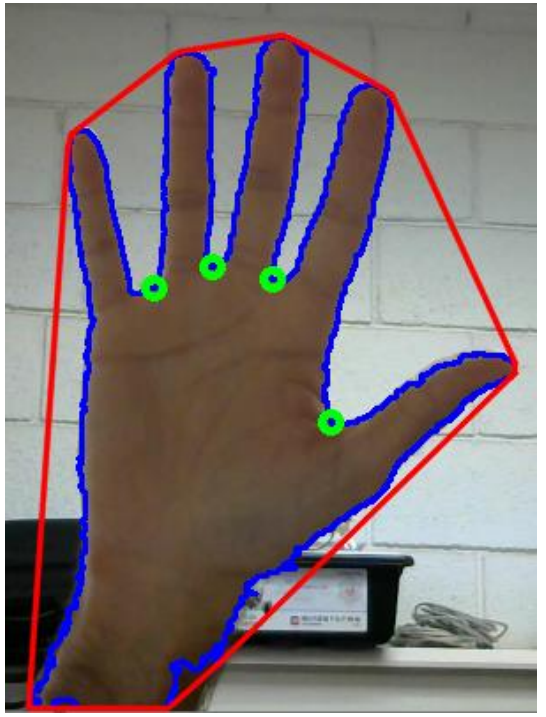
```
vector<int> hull;
convexHull(contours[index],hull);
```

Convexity Defects

```
circle(output_img, f, 5, Scalar(0, 255, 0), 3);
```



Filtrado



```
vector<Vec4i> defects;

convexityDefects(contours[index], hull, defects);
```

Convexity Defects

- Estructura de cada defecto de convexidad
  - punto inicial
  - punto final
  - punto más lejano
  - distancia aproximada del contorno al punto más lejano

Código 3.2

- Filtrar los defectos de convexidad para quedarnos con los que correspondan a la unión de los dedos.
  - Se puede hacer uso de la función *getAngle*