Proyecto final Sistemas de Percepción:

"Detección de disparos en una diana"

Gonzalo Cabanas Yuste NIA: 100317371

Daniel Fernández Delgado NIA: 100316780

Jorge Pastor García NIA: 100315924

Índice

1. Introducción

A lo largo de este curso hemos aprendido tanto conceptos teóricos de visón por computador, como conceptos prácticos aplicados a esta tecnología, como puede ser la detección de bordes, procesamiento de imágenes o segmentación… entre otros.

Para realizar este proyecto final, hemos aplicado varias de las técnicas mencionadas anteriormente en el entorno de programación "Visual Studio 2017", ayudados por la librería para el desarrollo de aplicaciones de visión por computador "OpenCv" en su versión 3.3.

El proyecto escogido para implementar ha sido una aplicación que consiste en la detección de los disparos realizados con un arma de aire comprimido a una diana de tiro al blanco. Cuyo objetivo final es el de calcular la puntuación obtenida de la suma de todos los disparos existentes en la diana.

Para ello se han valorado distintas opciones de desarrollo, optando finalmente por la que mejor se adapta a nuestro objetivo final teniendo en cuenta el rendimiento ofrecido.

2. Proceso de diseño y algoritmo final

A lo largo de la realización de este proyecto nos hemos encontrado con distintas alternativas para el desarrollo de este.

En un primer lugar se ideó proceso que consistía en la detección tanto de los círculos de los disparos efectuados, por un lado, así como la detección de los círculos concéntricos que componen el rango de puntuaciones de una diana por el otro. Para ello se utilizó la función "Hough Circle Transform". Pero con la utilización de esta función se producían principalmente dos problemas:

1. **Problema para detectar círculos concéntricos:** La función Hough Circle no permite que dos círculos compartan el mismo centro, por lo que resultaba imposible definir un espacio para cada una de las diez posibles puntuaciones.

2. **Problemas para detectar disparos:** Esta función, en este caso, también nos daba una alta tasa de error, ya que los agujeros que definen las balas en la diana no siempre eran círculos perfectos, por lo que ocasionalmente no era capaz de detectarlos. Además, dependiendo del color del fondo sobre el que realizáramos la fotografía detectaba esos círculos con mayor o menor efectividad.

También se valoraron otros métodos, como trabajar con áreas para detectar los disparos, o con resta de imágenes, pero estas variantes fueron rápidamente desechadas.

Finalmente se optó por un algoritmo que consta de dos procesos como se muestra en la figura1:

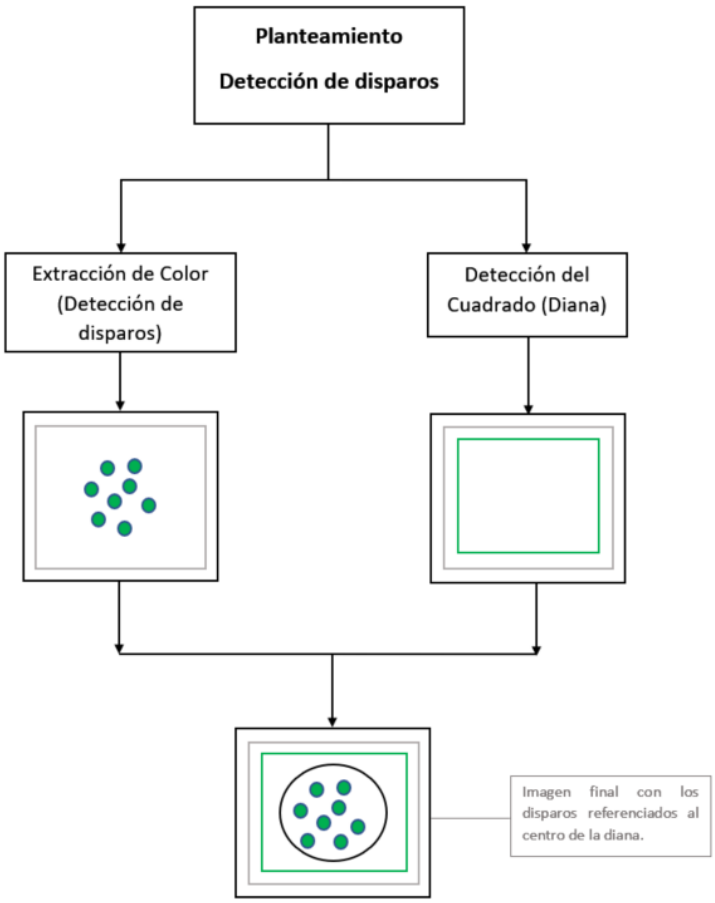

Figura1. Planteamiento de funcionamiento del programa.

Para el correcto funcionamiento de la aplicación es necesario que la imagen adquirida esté realizada sobre un fondo verde para poder realizar la correcta detección de los disparos mediante una separación de color.

Por un lado, se ha desarrollado un método para detectar correctamente estos disparos, por otro se ha detectado la diana y se han referenciado esos disparos al centro de esta, para así poder calcular sus puntuaciones. A continuación, se procede a explicar el funcionamiento del programa en profundidad.

3. Funcionamiento de la aplicación

Como se ha mencionado anteriormente el programa consta de dos funcionalidades principales, la detección de disparos, y la detección de la diana, y una tercera que consiste en combinar los datos obtenidos de las anteriores. Ambas se desarrollarán a partir de la misma imagen de partida, pero se aplican diferentes métodos para cumplir dichos objetivos. Para ello se ha seguido la estructura que ha de seguir cualquier aplicación de visión por computador.

- **Adquisición de la imagen**

  Se realiza una fotografía con la que poder trabajar, la imagen debe cumplir con los requisitos mencionados anteriormente (fondo verde) como se muestra en la figura2.

Figura2. Adquisición de la imagen a procesar.

A continuación, se carga la imagen tanto en color, como en escala de grises y se redimensiona para poder trabajar con ella si no fuera del tamaño apropiado figura3.

```
//Cargar la imagen en blanco y negro
dianaOriginal = imread("definitiva.jpg", CV_LOAD_IMAGE_GRAYSCALE);
//Cargar la imagen en color
dianaRGB = imread("definitiva.jpg", CV_LOAD_IMAGE_ANYCOLOR);
//Las redimensiona para poder trabajar con ella
Size size(1000, 1000);
resize(dianaOriginal, dianaOriginal, size);
resize(dianaRGB, dianaRGB, size);
```

Figura3. Carga de las imágenes de trabajo.

Una vez obtenida la imagen inicial se procede al preprocesamiento de la imagen, para poder así obtener mejores resultados y evitar problemas en la detección de características de la imagen.

- **Preprocesamiento**

Para este caso, será necesario procesar la imagen tanto para la correcta detección posterior del rectángulo y de su centro, como para la detección de disparos, por lo que trabajamos con una imagen binaria a la que se aplica un filtro (GaussianBlur) que mejora la detección de contornos de la imagen como se muestra en la figura4.

```
//La convierte a binaria
threshold(dianaOriginal, dianaBinaria, 70, 255, THRESH_BINARY);
//Filtro para mejorar la detección de contornos
GaussianBlur(dianaBinaria, dianaBinaria, Size(9, 9), 0, 0);
```

Figura4. Preprocesamiento.

Detección de diana y su centro

Con este desarrollo se busca detectar el rectángulo que encierra a los círculos concéntricos de las puntuaciones de la diana, para así poder localizar su centro geométrico en el cual se encontrará la máxima puntuación (10 puntos).

- **Extracción de características**

Tras realizar el preprocesamiento ya tendremos la imagen en condiciones para conseguir detectar el centro de la diana.

En primer lugar, se detectan todos los rectángulos de la imagen mediante "connectedComponents" y se almacenan en un vector "contours". A partir de este vector se calculan los momentos de los rectángulos, para posteriormente utilizarlos en el cálculo de los centros de cada contorno encontrado. Estos centros se almacenarán en el vector "centros".

```
//Calcular momentos
vector<Moments>momentos(contours.size());
for (int i = 0; i <contours.size(); i++)
{
    momentos[i] = moments(contours[i], false);
}
//Detectar centros
vector<Point2f> centros(contours.size());
for (int i = 0; i < contours.size(); i++)
{
    centros[i] = Point2f(momentos[i].m10 / momentos[i].m00, momentos[i].m01 / momentos[i].m00);
}
```

Figura5. Calculo del centro del rectángulo.

Tras esto, se halla de entre todos los rectángulos detectados el cuadrado que conforma la diana. Para ello, se van comparando los componentes del vector "contours" hasta obtener el de mayor área. Este, a excepción de imágenes con grandes errores, será siempre el de la diana, por lo que almacenaremos la posición de su centro y su área para utilizarlos a la hora de realizar el cálculo de las puntuaciones.

```
double areaRectangulo = 0;
double areaTemporal;
int posicionRectangulo;

for (int i = 0; i < contours.size(); i++)
{
    areaTemporal = contourArea(contours[i]);
    if (areaTemporal > areaRectangulo)
    {
        areaRectangulo = areaTemporal;
        posicionRectangulo = i;
    }
}
//Centro de la diana
double centroX = centros[posicionRectangulo].x;
double centroY = centros[posicionRectangulo].y;
```

Figura6. Calculo del rectángulo y su centro.

Detección de disparos

Para la detección de disparos se ha decidido realizar una segmentación por colores de la imagen, para ello hemos elegido un fondo verde, y trabajar en el espacio de colores HSV, ya que en este espacio de colores la detección del verde es mucho más efectiva.

En primer lugar, se ha definido el espacio de color HSV, y se han umbralizado los valores de verde a detectar. Como se observa en la figura7.

```
cvtColor(dianaRGB, img_hsv, CV_BGR2HSV);

Mat H(img_hsv.size(), CV_8UC1);
Mat S(img_hsv.size(), CV_8UC1);
Mat V(img_hsv.size(), CV_8UC1);
Mat array_channels[] = { H,S,V };

split(img_hsv, array_channels);

threshold(H, H, 40, UCHAR_MAX, CV_THRESH_TOZERO);
threshold(H, H, 75, UCHAR_MAX, CV_THRESH_BINARY_INV);
threshold(S, S, 67, UCHAR_MAX, CV_THRESH_TOZERO);
threshold(S, S, 255, UCHAR_MAX, CV_THRESH_BINARY_INV);
threshold(V, V, 45, UCHAR_MAX, CV_THRESH_TOZERO);
threshold(V, V, 189, UCHAR_MAX, CV_THRESH_BINARY_INV);
```

Figura7. Espacio HSV y umbralización del verde.

A continuación, se procede a umbralizar el color negro en el espacio de colores RGB para eliminar la diana para la posterior detección de disparos. Combinando ambos procesos obtenemos una imagen con los disparos aislados, pero con cierto ruido en algunas partes de la imagen, lo cual nos puede llevar a error en las siguientes operaciones.

Para eliminar ese ruido se realizan una serie de erosiones, dilataciones y filtros a la imagen de valor previamente definido, de la forma que se observa en la figura8.

```
int erosion_size = 6;
Mat element = getStructuringElement(cv::MORPH_ELLIPSE,
    cv::Size(2 * erosion_size + 1, 2 * erosion_size + 1),
    cv::Point(erosion_size, erosion_size));

// Apply erosion or dilation on the image
erode(resultadoF, resultadoF, element);
dilate(resultadoF, resultadoF, element);
erode(resultadoF, resultadoF, element);
dilate(resultadoF, resultadoF, element);
bitwise_and(resultadoF, B, resultadoF);
blur(resultadoF, resultadoF, Size(3, 3));
erode(resultadoF, resultadoF, element);
dilate(resultadoF, resultadoF, element);
```

Figura8. Transformaciones para la eliminación del ruido.

Como resultado de este proceso, se obtiene una imagen con los disparos bien diferenciados y con el fondo de la diana completamente eliminado, como se observa en la figura9.

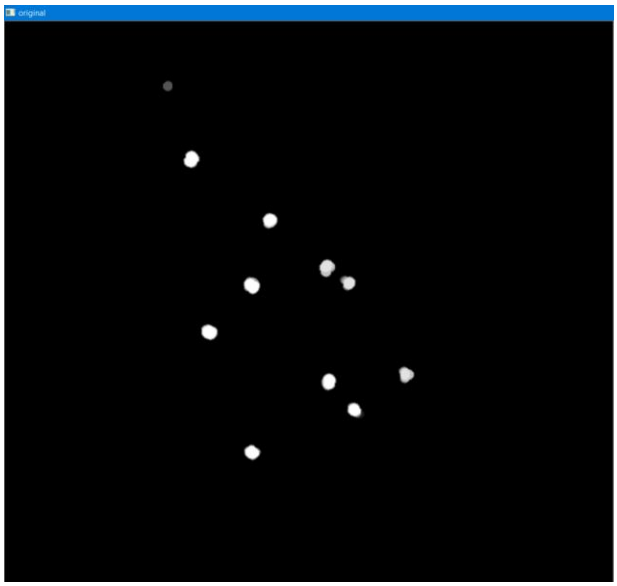

Figura9. Umbralización para obtener disparos.

Una vez en este punto, se procede a la diferenciación de objetos (disparos), para ello se realiza un labeling de cada disparo, y se almacenan los centroides. Para ello utilizaremos la función connectedComponentsWithStats como se observa en la figura 10.

```
int numeroDisparos;
Mat labels;
Mat stats;
Mat centrosDisparos;
int connectivity = 8;

numeroDisparos = connectedComponentsWithStats(resultadoF, labels, stats, centrosDisparos, connectivity);
```

Figura10. Labeling disparos.

Finalmente, para comprobar que todo se ha calculado correctamente, se marcan los objetos identificados en el labeling, y se muestran los centroides. Ver figura11 y figura12.

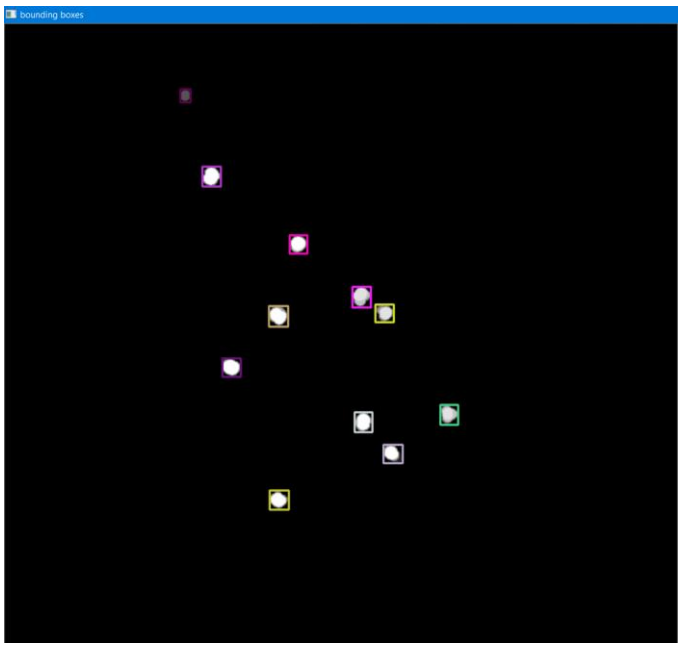

Figura11. Disparos detectados.

```
number_of_labels: 13
stats:
[0, 0, 910, 914, 826567;
 237, 88, 14, 18, 194;
 267, 193, 25, 27, 509;
 385, 286, 24, 25, 451;
 470, 356, 25, 28, 523;
 501, 380, 25, 24, 429;
 357, 382, 26, 28, 545;
 294, 453, 25, 25, 500;
 589, 516, 24, 27, 458;
 473, 526, 24, 27, 490;
 512, 570, 26, 25, 468;
 358, 632, 26, 26, 463;
 54, 866, 14, 14, 143]
centroids:
[454.7652337923, 456.6127996883495;
 243.7113402061856, 96.5;
 278.9744597249509, 206.2612966601179;
 396.4035476718403, 297.8824833702882;
 481.8317399617591, 368.9598470363289;
 513.2517482517483, 390.6946386946387;
 369.5357798165138, 395.097247706422;
 305.75, 464.806;
 599.8602620087336, 528.5545851528384;
 484.5938775510204, 539;
 524.1260683760684, 582.1688034188035;
 369.9244060475162, 644.5939524838013;
 60.35664335664335, 872.6573426573426]
```

Figura12. Labeling y centroides.

Unificación de procesos

Tras finalizar los procesos de detección del centro de la diana, y de los disparos, se procede a hacer el cálculo de las puntuaciones obtenidas.

Para ello se tiene en cuenta las dimensiones de la diana real, y se saca una proporción de medidas con respecto al rectángulo calculado con anterioridad, referenciando esa proporción al centro de la diana, como se puede observar en la figura13.

```
double longitudRectangulo;
longitudRectangulo = sqrt(areaRectangulo);

double uno, dos, tres, cuatro, cinco, seis, siete, ocho, nueve, diez;
uno = (longitudRectangulo*7.8) / 17;
dos = (longitudRectangulo * 7) / 17;
tres = (longitudRectangulo*6.2) / 17;
cuatro = (longitudRectangulo*5.4) / 17;
cinco = (longitudRectangulo*4.6) / 17;
seis = (longitudRectangulo*3.8) / 17;
siete = (longitudRectangulo * 3) / 17;
ocho = (longitudRectangulo*2.2) / 17;
nueve = (longitudRectangulo*1.4) / 17;
diez = (longitudRectangulo*0.6) / 17;
```

Figura13. Relación entre medidas reales y de la imagen.

De esta manera, se posibilita hallar el rango de puntuaciones en función de la distancia a la que se encuentre el centroide de un disparo detectado, con el centro de la diana, guardando la suma de las puntuaciones en una variable para obtener el total. Como se observa en la figura14.

```
double distancia;
double distanciaX;
double distanciaY;
int puntuacion = 0;

for (int i = 1; i < numeroDisparos; i++)
{
    double centroDisparoX = centrosDisparos.at<double>(i, 0);
    double centroDisparoY = centrosDisparos.at<double>(i, 1);

    distanciaX = (centroX - centroDisparoX);
    distanciaY = (centroY - centroDisparoY);
    distancia = sqrt(pow(distanciaX, 2) + pow(distanciaY, 2));

    cout << endl << "Disparo numero " << i << endl;

    if (distancia < diez)
    {
        cout << "10 puntos" << endl;
        puntuacion = puntuacion + 10;
    }
    if (distancia > diez && distancia < nueve)
    {
        cout << "9 puntos" << endl;
        puntuacion = puntuacion + 9;
    }
```

Figura14. Calculo de las puntuaciones.

Finalmente, el resultado obtenido es el mostrado en la figura15 y figura16.

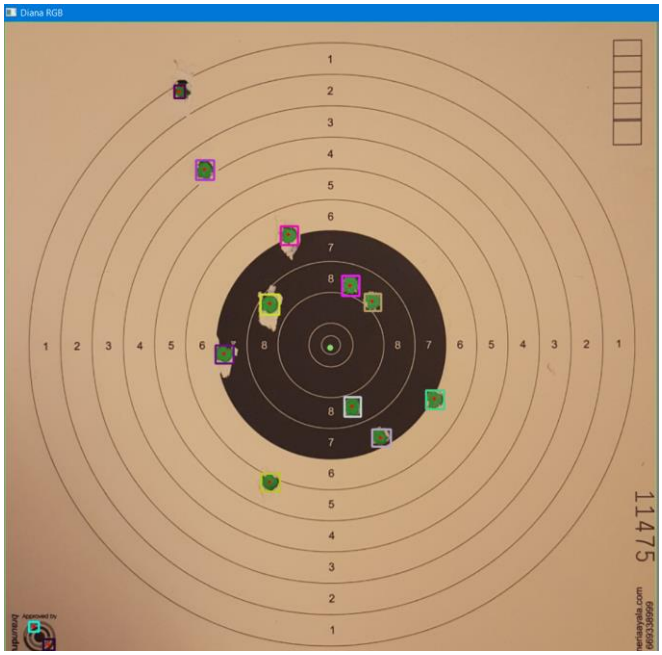

Figura15. Localización de y centro de la diana.

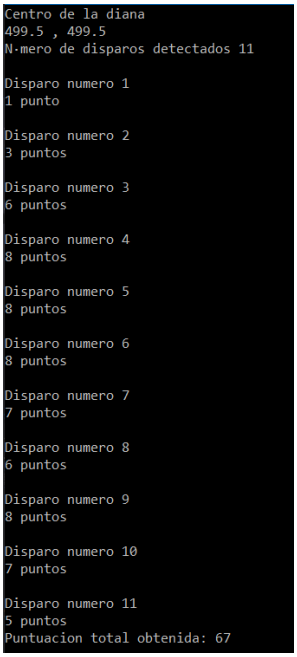

Figura16. Calculo de puntuaciones

Se observa que la aplicación localiza perfectamente todos los disparos, y dibuja el rectángulo que encierra la diana, así como el centro de esta. Además de calcular las puntuaciones correctamente.

4. Posibles mejoras y conclusiones

La aplicación implementada realiza su función correctamente, aunque en ciertos entornos puede sufrir ciertas complicaciones. Algunas de las futuras posibles mejoras a implementar pueden ser:

- Mejoras en entornos con mala iluminación: El programa trabaja bien cuando la imagen es tomada en entornos de buena iluminación, cuando esto no ocurre es posible que no detecte del todo bien algún disparo, por lo que una posible mejora es modificar el programa para hacerlo menos sensible a cambios de iluminación.

- Detección de disparos: La detección de disparos es válida cuando estos están separados entre sí. Cuando dos disparos coinciden parcialmente en un mismo agujero, el programa presenta dificultades para diferenciar los dos (o más disparos) y los identifica como uno solo, por lo que otra posible mejora sería solucionar ese aspecto.

- Fondo de la diana: El programa está diseñado para localizar los disparos con un fondo verde para así optimizar los resultados, una mejora sería poder detectarlos con cualquier otro fondo, o simplemente sin tener que tener uno de referencia.

Para concluir cabe destacar lo aprendido en esta práctica en cuanto a visión por computador. Durante el desarrollo del programa se ha aprendido tanto a realizar una aplicación real aplicando los conceptos vistos en clase, así como otros adquiridos para solventar los problemas que se iban presentando. En definitiva, se han afianzado estos conceptos para poder así iniciarnos en este tipo de tecnología que es la visión por computador.