ANEXO 6

```cpp
//*********************************************************************//
//                ANEXO 6. Programa matriz de tamaño                   //
//*********************************************************************//

// Librerias C++ generales
#include <cstdio>
#include <stdlib.h>
#include <iostream>
#include <stdio.h>
#include <ctype.h>
#include <Windows.h>
#include <fstream>
#include <vector>
#include <math.h>

//Librerias OpenCv generales
#include "opencv2/core/core.hpp"
#include "opencv2\core\utility.hpp"
#include "opencv2\core.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv\cv.hpp"
#include "opencv\cv.h"

//Librerias tratamiento de video
#include "opencv2\videoio.hpp"
#include "opencv2\video\background_segm.hpp"
#include "opencv2/video/tracking.hpp"
#include "opencv2/ml/ml.hpp"

// Variales globales relacionadas con el tratamiento de la imagen
const static int SENSITIVITY_VALUE = 25;
const static int BLUR_SIZE = 10;
#define ESCAPE 27
#define KEY_UP 27
#define pi 3.14159

// Variable global para recuadrar el movimiento
Rect Recuadro_movimiento = Rect(0, 0, 0, 0);

// Definir std y cv para las funciones OpenCv
using namespace std;
using namespace cv;

// Variables relacionadas con la zona prohibida
int coordenadaX = 0;
int coordenadaY = 0;
double area = 0;

//*********************************************************************//
// Nombre de la función: conversion()
// Función: Función que transforma los números en formato texto.
//          Se ejecuta cada vez que se necesite representar coordenadas.
// Entradas: Number, número que se quiere transcribir.
// Salidas: SS, número pasado a texto.
//*********************************************************************//
```

```
string conversion(int number)
{
    std::stringstream ss;
    ss << number;
    return ss.str();
}

//*********************************************************************//
// Nombre de la función: conversion2()
// Función: Función que transforma los números en formato texto.
//          Se ejecuta cada vez que se necesite representar áreas.
// Entradas: Number, número que se quiere transcribir.
// Salidas: SS, número pasado a texto.
//*********************************************************************//
string conversion2(double number)
{
    std::stringstream ss;
    ss << number;
    return ss.str();
}

//*********************************************************************//
// Nombre de la función: main()
// Función: Función principal del programa. Tareas que realiza:
//              - Tratamiento previo de imágenes.
//              - Resta de imágenes para detectar el movimiento.
//              - Tratamiento posterior de la resta resultante.
//              - Almacena en un .txt información al pulsar la
//                barra espaciadora
// Entradas: ninguna.
// Salidas: Información en un .txt.
//*********************************************************************//
int main()
{
    puts("A continuacion va a aparecer un video en tiempo real de su webcam con cuadricula. Debe guiar el robot y anotar las areas en cada cuadro.");

    // Matrices donde se guardarán las imágenes que forman del tratamiento
    Mat captura;
    Mat captura_gray;
    Mat frame_fondo;
    Mat frame_fondo_gray;
    Mat diferencia;
    Mat diferencia_editada;

    //Variables para identificar los movimientos
    int i = 0, clase, k = 0;

    // Se almacenarán la información en un .txt
    ofstream file("Matriz_Tamaño.txt");
    if (!file)
    {
        cout << "open file error!";
        return 1;
    }
```

```cpp

    // Cargar el video de la webcam
    VideoCapture video_original(1);

    // Verificación de que el video se ha cargado correctamente
    if (!video_original.isOpened())
    {
        cout << "No se encontro camara.\n";
    }

    // Se carga la imagen patrón
    frame_fondo = cv::imread("capturaf.png");
    cvtColor(frame_fondo, frame_fondo_gray, COLOR_BGR2GRAY);

    // Variable para finalizar la ejecuación de la aplicación al dar a escape
    char keypressed = 0;
    bool exit = false;

    // Espera con el fín de estabilizar la imagen de la cámara
    Sleep(4000);

    // Mientras no se pulse la tecla ESCAPE, la aplicación se ejecuta
    while(exit==false)
    {

        if (GetAsyncKeyState(VK_ESCAPE))
        {
            exit = true;
        }

        // Variables para contornear la zona en movimiento
        Mat copia_diferencia_editada;
        vector< vector<Point> > contorno;
        vector<Vec4i> hierarchy;
        bool movimiento_detectado = false; // Por defecto, no se ha encontrado ↵
           ningún elemento en movimiento

                                            // Variables para diferencias      ↵
                    contornos
        Mat test_data(10, 2, CV_32F);
        double area_baseMIN, area_baseMAX; // Área base de la aspiradora      ↵
          objetivo

                                            // Captura de una imagen del video
        video_original.read(captura);

        // Pasar la imagen a escala de grises
        cvtColor(captura, captura_gray, COLOR_BGR2GRAY);

        // Restar ambas imágenes. Puesto que está todo inmovil salvo el robot, ↵
           la imagen resultante mostrará el movimiento.
        absdiff(captura_gray, frame_fondo_gray, diferencia);

        // La diferencia obtenida se difumina, de forma que el contorno se    ↵
          vuelve más grueso
        blur(diferencia, diferencia_editada, cv::Size(BLUR_SIZE, BLUR_SIZE));
```

```
        // Umbralizar
        // Se hace notar más la diferencia obtenida, la cual la se pinta de color blanco (255)
        // Con una sensibilidad inferior a 20 aparece ruido.
        // Con una sensibilidad superior a 20 se reduce el contorno de la diferencia
        threshold(diferencia_editada, diferencia_editada, SENSITIVITY_VALUE, 255, THRESH_BINARY);

        // Visualización de resultados
        imshow("Detección del movimiento", diferencia);
        imshow("Detección del movimiento tratado", diferencia_editada);

        //¿Dónde se ha producido el movimiento?
        // Se copia la parte del video donde sólo se ve el movimiento en blanco
        diferencia_editada.copyTo(copia_diferencia_editada);

        // Se buscan contornos. Serán muy nitidos tras el tratamiento anterior.
        findContours(copia_diferencia_editada, contorno, hierarchy, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE);

        //¿Se ha encontrado algun contorno?
        if (contorno.size()>0)
            movimiento_detectado = true;
        else
            movimiento_detectado = false;

        // Si se ha detectado algún contorno, se procede a su estudio
        if (movimiento_detectado)
        {
            // Se analiza el tamaño de los contornos encontrados
            for (size_t i = 0; i<contorno.size(); i++)
            {
                // Lectura del área y del perímetro
                area = contourArea(contorno[i], false);

                // Se recuadra el contorno del movimiento para obtener su centro de coordenadas
                Recuadro_movimiento = boundingRect(contorno[i]);

                coordenadaX = Recuadro_movimiento.x + Recuadro_movimiento.width / 2;
                coordenadaY = Recuadro_movimiento.y + Recuadro_movimiento.height / 2;

                if (area > 1000)
                {
                    cout << "ROBOT DETECTADO" << endl;
                    // Dibujamos su contorno

                    drawContours(captura, contorno, i, Scalar(255), 2);

                    // Se dibuja el centro del rectángulo en el video original
```

```
                    circle(captura, Point(coordenadaX, coordenadaY), 2, Scalar(0, 255, 0), 2);
                    line(captura, Point(coordenadaX, coordenadaY), Point(coordenadaX, coordenadaY - 5), Scalar(0, 255, 0), 2);
                    line(captura, Point(coordenadaX, coordenadaY), Point(coordenadaX, coordenadaY + 5), Scalar(0, 255, 0), 2);
                    line(captura, Point(coordenadaX, coordenadaY), Point(coordenadaX - 5, coordenadaY), Scalar(0, 255, 0), 2);
                    line(captura, Point(coordenadaX, coordenadaY), Point(coordenadaX + 5, coordenadaY), Scalar(0, 255, 0), 2);

                    // Se escriben las coordenadas en el video
                    putText(captura, "(" + conversion(coordenadaX) + "," + conversion(coordenadaY) + ")", Point(coordenadaX, coordenadaY), 1, 1, Scalar(255, 0, 0), 1);
                    putText(captura, "Area:" + conversion2(area) , Point(coordenadaX+20, coordenadaY+20), 1, 1, Scalar(255, 255, 0), 1);

                    // Representación de una cuadrícula
                    line(captura, Point(0, 0), Point(650, 0), Scalar(0, 255, 255), 2);
                    putText(captura, "100", Point(10, 110), 1, 1, Scalar(0, 255, 255), 2);
                    line(captura, Point(0, 100), Point(650, 100), Scalar(0, 255, 255), 2);
                    putText(captura, "200", Point(10, 210), 1, 1, Scalar(0, 255, 255), 2);
                    line(captura, Point(0, 200), Point(650, 200), Scalar(0, 255, 255), 2);
                    putText(captura, "300", Point(10, 310), 1, 1, Scalar(0, 255, 255), 2);
                    line(captura, Point(0, 300), Point(650, 300), Scalar(0, 255, 255), 2);
                    putText(captura, "400", Point(10, 410), 1, 1, Scalar(0, 255, 255), 2);
                    line(captura, Point(0, 400), Point(650, 400), Scalar(0, 255, 255), 2);
                    putText(captura, "500", Point(10, 510), 1, 1, Scalar(0, 255, 255), 2);
                    line(captura, Point(0, 500), Point(650, 500), Scalar(0, 255, 255), 2);
                    putText(captura, "600", Point(10, 610), 1, 1, Scalar(0, 255, 255), 2);
                    line(captura, Point(0, 600), Point(650, 600), Scalar(0, 255, 255), 2);

                    line(captura, Point(0, 0), Point(0, 650), Scalar(0, 255, 255), 2);
                    putText(captura, "100", Point(110, 10), 1, 1, Scalar(0, 255, 255), 2);
                    line(captura, Point(100, 0), Point(100, 650), Scalar(0, 255, 255), 2);
                    putText(captura, "200", Point(210, 10), 1, 1, Scalar(0, 255, 255), 2);
                    line(captura, Point(200, 0), Point(200, 650), Scalar(0,
```

```
                    255, 255), 2);
                    putText(captura, "300", Point(310, 10), 1, 1, Scalar(0, 255, 255), 2);
                    line(captura, Point(300, 0), Point(300, 650), Scalar(0, 255, 255), 2);
                    putText(captura, "400", Point(410, 10), 1, 1, Scalar(0, 255, 255), 2);
                    line(captura, Point(400, 0), Point(400, 650), Scalar(0, 255, 255), 2);
                    putText(captura, "500", Point(510, 10), 1, 1, Scalar(0, 255, 255), 2);
                    line(captura, Point(500, 0), Point(500, 650), Scalar(0, 255, 255), 2);
                    putText(captura, "600", Point(610, 10), 1, 1, Scalar(0, 255, 255), 2);
                    line(captura, Point(600, 0), Point(600, 650), Scalar(0, 255, 255), 2);

                    // Al pulsar la barra espaciadora, se escribe en el .txt la información necesaria
                    if (GetAsyncKeyState(VK_SPACE))
                    {
                        if (waitKey(50) >= 0)
                        {
                            file << "INFO TAMAÑO " << endl;
                            file << "X:  " << coordenadaX << endl;
                            file << "Y:  " << coordenadaY << endl;
                            file << "Area: " << area << endl;
                        }
                    }
                }
            }
        }

        // Representación
        imshow("Localización del movimiento", captura);

        //Si pulsamos escape, salimos
        keypressed = waitKey(1);

    }

    //Cerramos ficheros de texto
    file.close();

    //Liberamos memoria
    destroyAllWindows();

    //Fin del programa
    return 0;
}
```