Erosión y Dilatación

Alumno: Josue Samuel Philco Puma

9 de junio de 2025

1. Implementando el programa

Para poder implementar un programa que va a detectar la trayectoria de un objeto, es necesario tener un video o grabación en tiempo real para poder realizarlo, este al momento en que pasa el objeto debe detectar todo el movimiento que va a realizar el objeto. Entonces primero veamos los dos enfoques, un enfoque que detecte la trayectoria del objeto en un video, al momento que funcione este paso ya por teoría debería funcionar con un video en tiempo real. Veamos entonces los dos casos:

1.1. Detección de Objetos mediante un video

Esta primera parte puede servirnos para luego pasar a usar video en tiempo real mediante la cámara web de nuestra laptop. Pero veamos primero los pasos que debemos realizar para lograrlo:

1. Primero debemos leer nuestro video que vamos a utilizar, de ser posible que sea en un fondo de color blanco para un mejor resultado.
2. Con el video deberíamos transformarlo a escala grises para poder aplicarle el proceso de binarización, con ello solo detectará el objeto. Para ello podríamos utilizar las funciones realizadas en prácticas anteriores.
3. Ahora con el video en escala gris deberíamos realizar una pequeña diferencia entre los frames, es decir, la diferencia entre el frame actual con el frame anterior.
4. Una vez que hagamos la diferencia recién aplicamos el proceso de binarización, para ello debemos tener el umbral requerido.
5. Hecho la binarización debemos aplicar una erosión o dilatación (se puede usar las funciones **cv::erode y cv::dilate**). Pero ello depende de como se esta detectando los frames y saber que aplicar.
6. Después hecho ese paso, tendríamos que detectar los contornos del objeto y calcular el centroide del objeto.
7. Una vez hecho, dibujar el centro del objeto y almacenarlos en un vector para obtener su trayectoria.
8. Con esos valores de la trayectoria dibujarlos sobre cada frame y mostrar en una imagen la trayectoria obtenida.

Como se puede ver, son muchos pasos que debemos hacer para poder detectar el objeto, entonces veamos el código que realiza esta parte:

```
Mat convert_gray(const Mat& frame_captured) {
    Mat frame_gray(frame_captured.rows, frame_captured.cols, CV_8UC1);
    int rows = frame_captured.rows;
    int columns = frame_captured.cols;

    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < columns; j++) {
            Vec3b pixel_value = frame_captured.at<Vec3b>(i, j);
            uchar gray_value = (uchar)(0.21 * pixel_value[2] + 0.72 *
                pixel_value[1] + 0.07 * pixel_value[0]);
            frame_gray.at<uchar>(i, j) = gray_value;
        }
    }

    return frame_gray;
}

```

```
Mat binarized_frame(const Mat& frame_gray, int threshold) {
    Mat frame_binarized(frame_gray.rows, frame_gray.cols, CV_8UC1);
    int rows = frame_gray.rows;
    int columns = frame_gray.cols;

    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < columns; j++) {
            uchar pixel_value = frame_gray.at<uchar>(i, j);
            frame_binarized.at<uchar>(i, j) = (pixel_value > threshold) ? 255 :
                0;
        }
    }

    return frame_binarized;
}
```

Esta primera parte del código se va a encargar de convertir todos los frames del video en escala gris y dejándolo listo para el proceso de binarización. Pero más adelante veremos su uso en un ejemplo. Por lo que nos falta el resto que detectará la trayectoria del objeto.

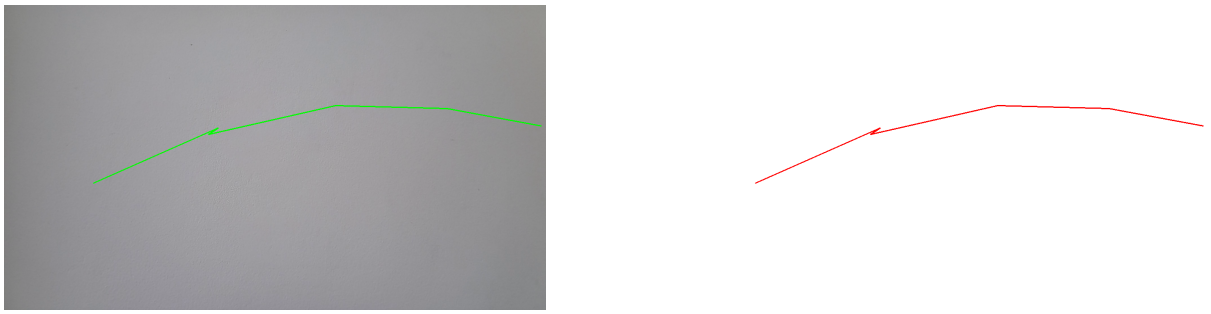
```
Point process_frame_trajectory(const Mat& frame_current, const Mat&
    previous_frame, int threshold, const string& path, int count_frame) {
    Mat frame_binarized, frame_binarized_copy, frame_eroded;

    absdiff(frame_current, previous_frame, frame_binarized);
    frame_binarized = binarized_frame(frame_binarized, threshold);
    frame_binarized_copy = frame_binarized.clone();

    Mat structure = getStructuringElement(MORPH_RECT, Size(5, 5));
    erode(frame_binarized, frame_eroded, structure);

    imwrite(path + "Frames/binarized_frame_" + to_string(count_frame) + ".png",
         frame_binarized_copy);
    imwrite(path + "Eroded/eroded_frame_" + to_string(count_frame) + ".png",
        frame_eroded);
    imshow("Binarized Frame", frame_binarized);

    vector<vector<Point>> contours;
    findContours(frame_binarized, contours, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE)
        ;

    if (!contours.empty()) {
        int contour_index = 0;
        double largest_area = 0;

        for (int i = 0; i < contours.size(); i++) {
            double area = contourArea(contours[i]);
            if (area > largest_area) {
                largest_area = area;
                contour_index = i;
            }
        }

        if (largest_area > 100) {
            Moments m = moments(contours[contour_index]);
            if (m.m00 > 0) {
                return Point((int)(m.m10 / m.m00), (int)(m.m01 / m.m00));
            }
        }
    }

    return Point(-1, -1);
}
```

Ahora si pasamos a la parte de la detección, Esta parte del código es muy importante va a detectar el objeto, además, como dijimos debemos saber si debemos realizar la erosión o la dilatación, por ello también guardamos los frames binarizados para ver frame por frame como es que detecto el objeto al estar binarizado. Una vez hecho eso detectamos los contornos del objeto y obtener su centroide para que dibuje su trayectoria recorrida, esta trayectoria no será el real pero por lo menos nos indica la trayectoria a logrado recorrer lo más preciso posible. Veamos un ejemplo de detecto el objeto por un frame guardado.

(a) Frame binarizado (b) Frame con erosión

Figura 1: Mostrando el frame binarizado (a) y el frame aplicado con erosión (b)

Con este proceso, el frame binarizado tenía unas partes donde podía haber zonas blancas que se detecto que puede ser causante del ruido, pero al aplicar el **proceso de erosión** con un elemento estructurante de 5x5 el frame se puede ya ver como elimina partes de ruido que detecto la binarización. Entonces ahora con ello debemos ver la traza obtenida.



(a) Detección captada en el video (b) Trayectoria obtenida como imagen

Figura 2: Trayectoria en el video (a) y el guardado de la trayectoria en una imagen (b)

Entonces, una vez realizado esto, podemos reutilizar esta misma lógica para mostrar la trayectoria pero utilizando la cámara de nuestra máquina para detectar la trayectoria del objeto en tiempo real.

NOTA: Para aplicar el umbral se debe hacer pruebas en donde lo detecta mejor el objeto, pero para objetos pequeños se recomienda un umbral máximo de 30 para su detección.

1.2. Detección de Objetos mediante cámara

Ahora que tenemos la detección de los objetos en un video, hacerlo con cámara es el mismo proceso anterior, solo sería cambiar el parámetro de la función **VideoCapture** y establecerlo en 0, esto hará que se ponga la cámara operando con OpenCV. Esta parte estaría en el main.

```
1  int main() {
2      VideoCapture cap(0);
3      Mat frame;
4      while (true) {
5          cap >> frame;
6          if (frame.empty()) {
7              cerr << "Error al capturar el frame.\n";Add commentMore actions
8              break;
```

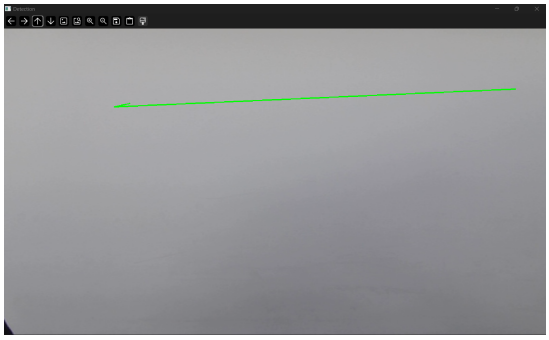
```
        }
        imshow("Webcam", frame);
        if (waitKey(1) == 'q') {
            break;
        }
    }
    cap.release();
    destroyAllWindows();
    return 0;
}
```

Con este main ya estaríamos usando nuestra cámara web de nuestra máquina. Entonces, simplemente aplicamos lo mismo para detectar el objeto y su trayectoria, por lo que el main quedaría de esta manera:

```
int main() {
    VideoCapture video(0);

    string trajectory_path = "D:/UNSA EPCC/7mo semestre/Computacion Grafica/
        Unidad 2/";
    string frame_binarized_path = trajectory_path + "Frames/";
    string frame_eroded_path = trajectory_path + "Eroded/";

    Mat frame_captured, frame_gray, previous_frame_gray;
    vector<Point> trajectory;

    namedWindow("Detection", cv::WINDOW_AUTOSIZE);
    int threshold_value = 100;
    int frame_count = 0;

    while (true) {
        video >> frame_captured;
        if (frame_captured.empty()) break;

        resize(frame_captured, frame_captured, Size(1280, 720));
        frame_gray = convert_gray(frame_captured);

        if (!previous_frame_gray.empty()) {
            Point center = process_frame_trajectory(frame_gray,
                previous_frame_gray, threshold_value, trajectory_path,
                frame_count);
            if (center.x != -1 && center.y != -1) {
                trajectory.push_back(center);
                circle(frame_captured, center, 5, Scalar(255, 0, 0), -1);
            }
        }

        previous_frame_gray = frame_gray.clone();
        frame_count++;

        for (size_t i = 1; i < trajectory.size(); i++) {
            line(frame_captured, trajectory[i - 1], trajectory[i], Scalar(0,
                255, 0), 2);
        }

        imshow("Detection", frame_captured);
        if (waitKey(30) >= 27) break;
    }

    video.release();

    Mat trajectory_image(720, 1280, CV_8UC3, Scalar(255, 255, 255));
    for (size_t i = 1; i < trajectory.size(); i++) {
        line(trajectory_image, trajectory[i - 1], trajectory[i], Scalar(0, 0,
            255), 2);
```

```
    }

    imshow("Result trajectory", trajectory_image);
    imwrite(trajectory_path + "trajectory_image.png", trajectory_image);

    waitKey(0);
    destroyAllWindows();

    return 0;
}
```

Entonces, no podriamos mostrarlo que esta funcionando como debe, pero se puede aplicar la detección con un objeto a tener en la mano y detectar su trayectoria.



(a) Detección captada en el video

(b) Trayectoria obtenida como imagen

Figura 3: Trayectoria en el video (a) y el guardado de la trayectoria en una imagen (b)

Como se menciono anteriormente, no es una detección tan precisa que se diga, pero se intenta mostrar la trayectoria del objeto para ver si por lo menos lo detecta. Igual se puede ir ajustando el umbral para obtener el umbral deseado.

2. Codigo completo

Acá estaría todo el código completo de este sistema de detección de objetos.

```
#include <vector>
#include <opencv2/opencv.hpp>
using namespace std;
using namespace cv;

Mat convert_gray(const Mat& frame_captured) {
    Mat frame_gray(frame_captured.rows, frame_captured.cols, CV_8UC1);
    int rows = frame_captured.rows;
    int columns = frame_captured.cols;

    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < columns; j++) {
            Vec3b pixel_value = frame_captured.at<Vec3b>(i, j);
            uchar gray_value = (uchar)(0.21 * pixel_value[2] + 0.72 *
                pixel_value[1] + 0.07 * pixel_value[0]);
            frame_gray.at<uchar>(i, j) = gray_value;
        }
    }

    return frame_gray;
}

Mat binarized_frame(const Mat& frame_gray, int threshold) {
    Mat frame_binarized(frame_gray.rows, frame_gray.cols, CV_8UC1);
```

```
    int rows = frame_gray.rows;
    int columns = frame_gray.cols;

    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < columns; j++) {
            uchar pixel_value = frame_gray.at<uchar>(i, j);
            frame_binarized.at<uchar>(i, j) = (pixel_value > threshold) ? 255 :
                0;
        }
    }

    return frame_binarized;
}

Point process_frame_trajectory(const Mat& frame_current, const Mat&
    previous_frame, int threshold, const string& path, int count_frame) {
    Mat frame_binarized, frame_binarized_copy, frame_eroded;

    absdiff(frame_current, previous_frame, frame_binarized);
    frame_binarized = binarized_frame(frame_binarized, threshold);
    frame_binarized_copy = frame_binarized.clone();

    Mat structure = getStructuringElement(MORPH_RECT, Size(5, 5));
    erode(frame_binarized, frame_eroded, structure);

    imwrite(path + "Frames/binarized_frame_" + to_string(count_frame) + ".png",
         frame_binarized_copy);
    imwrite(path + "Eroded/eroded_frame_" + to_string(count_frame) + ".png",
        frame_eroded);
    imshow("Binarized Frame", frame_binarized);

    vector<vector<Point>> contours;
    findContours(frame_binarized, contours, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE)
        ;

    if (!contours.empty()) {
        int contour_index = 0;
        double largest_area = 0;

        for (int i = 0; i < contours.size(); i++) {
            double area = contourArea(contours[i]);
            if (area > largest_area) {
                largest_area = area;
                contour_index = i;
            }
        }

        if (largest_area > 100) {
            Moments m = moments(contours[contour_index]);
            if (m.m00 > 0) {
                return Point((int)(m.m10 / m.m00), (int)(m.m01 / m.m00));
            }
        }
    }

    return Point(-1, -1);
}

int main() {
    // VideoCapture video(0); // Para c mara en vivo
    VideoCapture video("D:/UNSA EPCC/7mo semestre/Computacion Grafica/Unidad 2/
        Videos/video2.mp4");

```

```
    string trajectory_path = "D:/UNSA EPCC/7mo semestre/Computacion Grafica/
        Unidad 2/";
    string frame_binarized_path = trajectory_path + "Frames/";
    string frame_eroded_path = trajectory_path + "Eroded/";

    Mat frame_captured, frame_gray, previous_frame_gray;
    vector<Point> trajectory;

    namedWindow("Detection", cv::WINDOW_AUTOSIZE);
    int threshold_value = 100;
    int frame_count = 0;

    while (true) {
        video >> frame_captured;
        if (frame_captured.empty()) break;

        resize(frame_captured, frame_captured, Size(1280, 720));
        frame_gray = convert_gray(frame_captured);

        if (!previous_frame_gray.empty()) {
            Point center = process_frame_trajectory(frame_gray,
                previous_frame_gray, threshold_value, trajectory_path,
                frame_count);
            if (center.x != -1 && center.y != -1) {
                trajectory.push_back(center);
                circle(frame_captured, center, 5, Scalar(255, 0, 0), -1);
            }
        }

        previous_frame_gray = frame_gray.clone();
        frame_count++;

        for (size_t i = 1; i < trajectory.size(); i++) {
            line(frame_captured, trajectory[i - 1], trajectory[i], Scalar(0,
                255, 0), 2);
        }

        imshow("Detection", frame_captured);
        if (waitKey(30) >= 27) break;
    }

    video.release();

    Mat trajectory_image(720, 1280, CV_8UC3, Scalar(255, 255, 255));
    for (size_t i = 1; i < trajectory.size(); i++) {
        line(trajectory_image, trajectory[i - 1], trajectory[i], Scalar(0, 0,
            255), 2);
    }

    imshow("Result trajectory", trajectory_image);
    imwrite(trajectory_path + "trajectory_image.png", trajectory_image);

    waitKey(0);
    destroyAllWindows();

    return 0;
}
```

Referencias

[1] OpenCV. *Erosion and Dilation — OpenCV 3.4 documentation*. 2018. Disponible en: `https://docs.opencv.org/3.4/db/df6/tutorial_erosion_dilatation.html`. [Último acceso: 9 de junio de 2025].

[2] MathWorks. *Detecting a Cell Using Image Segmentation - MATLAB & Simulink*. 2024. Disponible en: `https://www.mathworks.com/help/images/detecting-a-cell-using-image-segmentation.html`. [Último acceso: 9 de junio de 2025].

[3] Stack Overflow. *How to use erode and dilate function in OpenCV*. 2013. Disponible en: `https://stackoverflow.com/questions/17329932/how-to-use-erode-and-dilate-function-in-opencv`. [Último acceso: 9 de junio de 2025].

[4] OpenCV. *Image Thresholding — OpenCV-Python Tutorials*. 2023. Disponible en: `https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html`. [Último acceso: 9 de junio de 2025].

[5] DataFlair. *Image Conversion using OpenCV*. 2020. Disponible en: `https://data-flair.training/blogs/image-conversion-using-opencv/`. [Último acceso: 9 de junio de 2025].