24 de marzo de 2019

Entrega 2 - Segmentación

Robótica y Percepción Computacional

Alumnos:

Luciano García Giordano (150245)
Gonzalo Flórez Arias (150048)
Salvador González Gerpe (150044)

Universidad Politécnica de Madrid
ETSI Informáticos

Índice

1. Método. Solución propuesta. Algoritmo de segmentación. Descripción del código desarrollado

A la hora de implementar la parte de segmentación, hemos llegado a tomar una serie de decisiones concretas con el objetivo de lograr los mejores resultados posibles. En este sentido, parecen claras las dos vías por las que se puede optimizar un algoritmo de segmentación: mejorar la calidad de la segmentación y mejorar los tiempos de ejecución.

Nosotros tuvimos, ya desde el primer momento, claro que lo importante sería lograr una buena calidad en la segmentación, y que a partir de ahí el objetivo sería mejorar los tiempos que tarda el algoritmo en proporcionar la segmentación de cierta imagen (ya que en el robot real esto resulta ser importante).

Nuestra primera medida, frente al "diseño" original del algoritmo, fue cambiar el espacio de colores de RGB a HSV. A esto unimos el que finalmente nos quedamos solo con H y S, dejando fuera la V (para tratar de evitar problemas con los cambios de iluminación). Esto resultó ser definitivamente importante para lograr un buen clasificador.

Para etiquetar las imágenes se usó el programa proporcionado por el profesor, que permite colorear un vídeo abierto con OpenCV, por lo que tomamos algunas imágenes y las coloreamos, tratando de obtener datos variados para la línea, el suelo y las marcas. Con esto construimos nuestro dataset de entrenamiento, con unos 30000 píxeles (lo consideramos una cantidad muy elevada, pero hemos obtenido buenos resultados con ello). Almacenamos, de todas las imágenes coloreadas, los canales H y S de los píxeles coloreados para de ahí entrenar el clasificador.

Respecto al clasificador, aunque la idea inicial era usar un clasificador euclídeo, de manera similar al desarrollado en la asignatura de Reconocimiento de Formas, pensamos que sería ventajoso buscar algo un poco más complejo pero que pudiera separar mejor las clases. Probamos, usando bibliotecas como sklearn, distintas opciones: primero un clasificador gaussiano de tipo Naïve Bayes, luego una red neuronal de 3 capas, luego una de dos capas (haciendo diversas pruebas con distintos números de nodos en cada capa). Nos quedamos finalmente con una red neuronal de 2 capas con 3 nodos en cada una de ellas. Utilizamos activación logística y una tasa de aprendizaje α de 0.05. Con ello obtenemos unos resultados con altos porcentajes de acierto (99.5 % de acierto usando un 90 % de datos de training y un 10 % de datos de test), pero asegurando a la vez una velocidad de ejecución rápida, que se comentará más adelante.

Así, al ejecutar el programa se ejecuta el entrenamiento pertinente, y posteriormente se va leyendo el vídeo que se deseé segmentar y calculando dicha segmentación, coloreando y mostrando el resultado a la vez. Es posible también hacer esa lectura directamente a partir de una cámara conectada al ordenador.

Pasan a comentarse ahora las mejoras llevadas a cabo desde la primera entrega de segmentación:

- Gracias a un uso más adecuado de la biblioteca de sklearn, se logró que la ejecución del algoritmo de segmentación fuera paralelizable. De esa manera, no perdemos demasiada potencia de cálculo por utilizar un algoritmo más complejo, como son las redes de neuronas artificiales.
- Se pasó a obtener los frames del vídeo de forma reducida. De esa manera, utilizamos para segmentación los píxeles de la imagen de 4 en 4, reduciendo el coste computacional al 1/16 del original. Según nuestras observaciones, casi no se pierde calidad en la segmentación pero se aumenta mucho la velocidad de cómputo.
- Se redujeron los nodos en cada capa de la red neuronal, logrando así una velocidad de cómputo mayor sin perder mucha calidad en la segmentación.
- Se implementó la parte opcional del seguimiento de la pelota (explicado a continuación).
- Como medida para aumentar aún más el acierto en la segmentación, se decidió aumentar la saturación de la cámara a la hora de tomar los vídeos. Según nuestras observaciones, eso hace que los algoritmos funcionen notablemente mejor.
- A lo largo de 2 días, tabajamos en el laboratorio de robótica. Allí pudimos grabar y segmentar vídeos tomados desde la propia cámara del robot, con el objetivo de tener ya un vídeo del entorno real de la práctica, además de empezar a probar situaciones reales de seguimiento de líneas en el suelo y de seguimiento de una pelota de ténis.

Respecto a la implementación de nuestro programa en Python, al comenzar a ejecutar el programa, se llama al clasificador, y se le pide que entrene a partir de los datos del dataset, que se guardan en un fichero (con datasetGenerator se genera este fichero, a partir de las imágenes coloreadas con etiquetaImagenes. Esa generación solo se hace una vez).

Se inicia la captura de imágenes, en base a un vídeo concreto (aunque puede realizarse sobre un vídeo en directo desde una cámara). Cada imagen recibida se reduce en un factor de 4, vertical y horizontalmente (no hemos considerado el trabajar con imágenes algo pequeñas un problema para el futuro, pues el tamaño sigue siendo considerable para poder ver correctamente la línea y cada marca), y en un bucle, se van leyendo las imágenes. Se pasa cada imagen al formato HSV, y se obtiene el H y el S. Se clasifica con predict cada pixel de la imagen con el clasificador, y se muestra por pantalla la imagen segmentada. Entre medias se realizan pruebas del tiempo tardado, usadas para el apartado correspondiente de esta memoria.

2. Observaciones acerca del límite de upload de Moodle

En todos los casos, por el tamaño máximo de la entrega en Moodle, incluimos todos los vídeos convertidos en .m4v (video codec h264) y altamente comprimidos. La calidad es defínitivamente mala, pero al menos podemos incluir varios vídeos de esa manera. A su vez, incluimos el dataset utilizado para entrenar el algoritmo. De esa forma, el evaluador puede ejecutar el programa y observar su comportamiento.

3. Resultado. Algoritmo de segmentación

Tras probar el algoritmo que tenemos en el robot de la sala de robótica, podemos certificarnos de que funciona bien en un entorno real controlado. Además, podemos asegurarnos de que los tiempos de ejecución son, como mínimo, adecuados. En nuestro caso, la fase de segmentación se hace de tal forma que conseguimos concluirla en 0.01 segundos, lo que nos permite ejecutar muchos frames en cada segundo (serían 77 si el step que implementamos fuera el total del programa). Si ejecutamos el programa que se entrega en este momento, que no es el brain que desarrollamos para probar en el laboratorio de robótica, el tiempo de ejecución de un ciclo es, de media, de 0,013 segundos en un ordenador portátil similar al del robot del laboratorio de robótica (ambos tienen un procesador Intel i3, por ejemplo). La velocidad de la segmentación fue de 77 frames por segundo.

En cuanto a resultados, mostramos a continuación algunas capturas que hicimos en el laboratorio de robótica controlando manualmente el robot y capturando el vídeo, y luego procesándolo en nuestro programa. En la figura 1 se puede ver que la segmentación es correcta. Sin embargo, los bordes de la línea tienen partes rojas. Eso se debe a que no entrenamos el algoritmo con un dataset demasiado separable. Para solucionar este problema, podemos hacer uso de la función erode de openCV, que nos permite eliminar pequeñas regiones mal clasificadas. Otra solución sería hacer un dataset más variado, aunque dado que no consideramos el contexto del píxel como feature en nuestro algoritmo eso puede no ser suficiente. En un brain que hicimos para probar en la sala de robótica para que el robot pudiera seguir líneas, utilizamos la función erode y obtuvimos resultados excelentes.

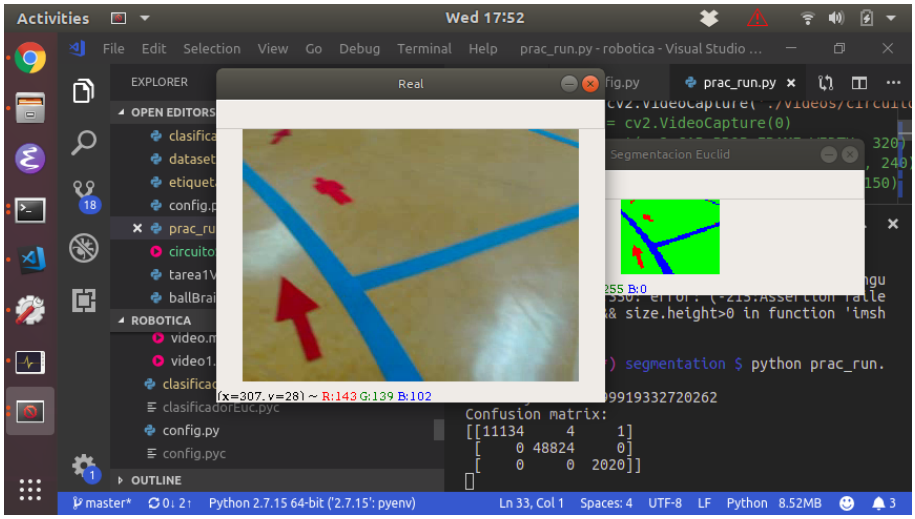

Figura 1: Ejemplo de la segmentación.

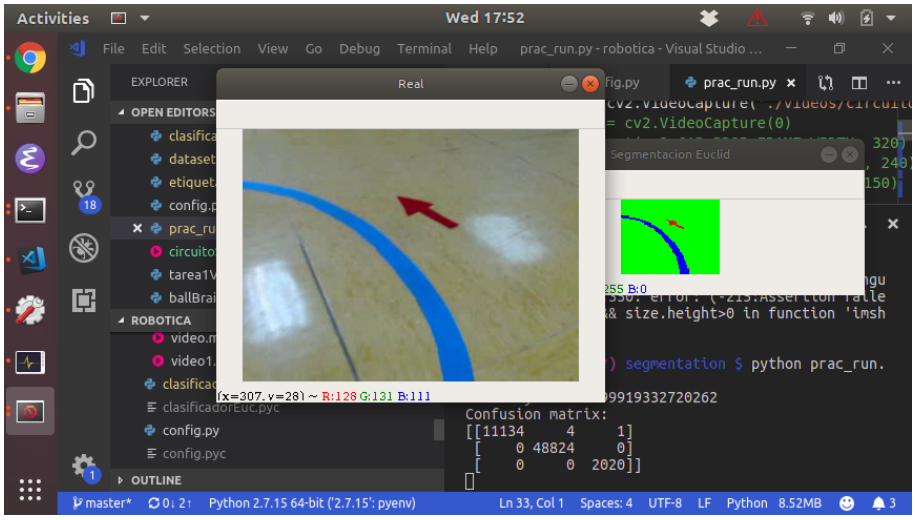

Figura 2: Ejemplo de la segmentación. Si nos detenemos bien en los bordes de la línea, podemos ver fácilmente algunos píxeles rojos. Sin embargo, hemos llegado a hacer pruebas en el caso de la parte opcional con la función erode de openCV, que nos permite eliminar esas pequeñas regiones mal segmentadas. No implementamos en nuestra solución aún porque eso exige capturar la capa de línea y la de símbolo por separado y en valores binarios para luego aplicar la función a cada una por separado y volver a mostrar los resultados.

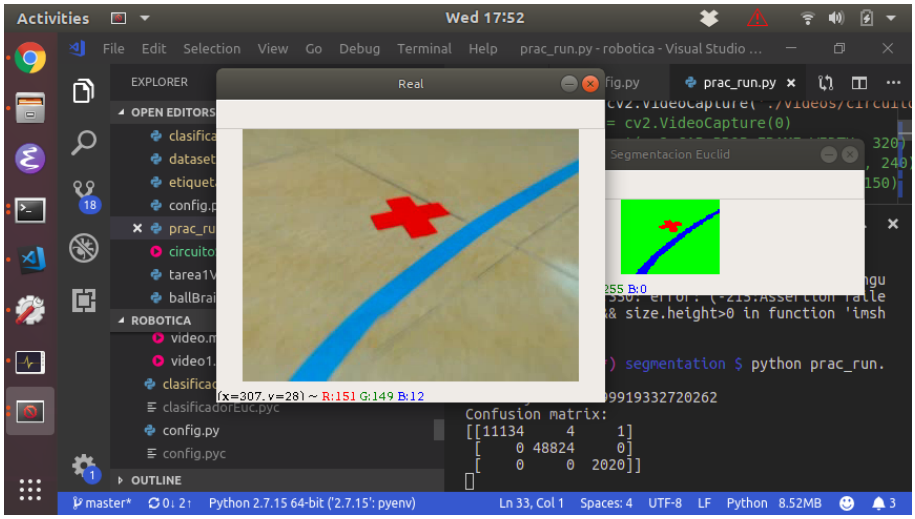

Figura 3: Ejemplo de la segmentación.

Figura 4: Ejemplo de la segmentación.

4. Parte opcional. Seguimiento de pelota

También realizamos la parte opcional de la práctica, referida al seguimiento de un objeto esférico por parte del robot, que va controlando su distancia a dicho objeto según los resultados del tamaño de la pelota relativo a la cámara del robot. Logramos adaptar el algoritmo de segmentación para poder obtener la pelota de un vídeo, y calculamos la manera de obtener la distancia relativa de la cámara a la pelota, en base a la explicación dada en clase. También hemos realizado una primera integración de esto con la parte de control, de tal manera que el robot se acerque a la pelota si la encuentra a lo lejos y se aleje si la tiene muy cerca. También implementamos la capacidad del robot de girar para intentar mantener la pelota en el centro de su campo de visión. Probamos el algoritmo en forma de "Brain" en el robot del laboratorio de robótica, logrando unos resultados interesantes. Incluimos en la carpeta de entrega un video de demostración del seguimiento junto con la segmentación hecha por el mismo programa.

Sobre nuestras pruebas en el laboratorio, incluimos en la carpeta de entrega un timelapse grabado con un móvil de uno de los integrantes del grupo guiando el robot por la sala. Además, incluimos dos videos, uno con la imagen original y otro de la segmentación, hechos en directo por el robot en una situación similar a la mostrada en el time lapse (hemos perdido la referencia de si se trata de la misma ejecución o no). Se muestra a seguir una captura de pantalla del video mencionado, tanto de la imagen original (Figura 5) como de la segmentación (Figura 6).

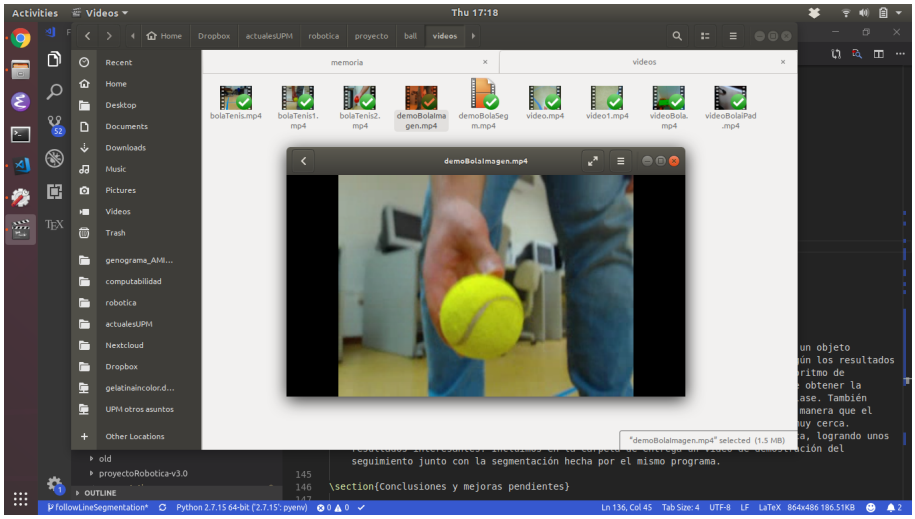

Figura 5: Captura de la imagen original de la pelota en un brain que ejecuta en el robot real.

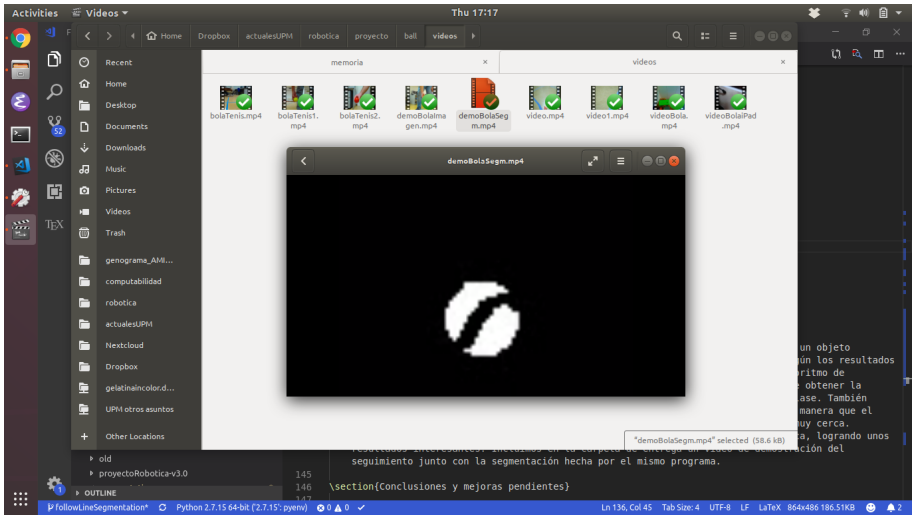

Figura 6: Captura de la segmentación de la pelota en un brain que ejecuta en el robot real. Como bien se puede ver, la curva blanca de la pelota de tennis no se captura como parte de la pelota. Además, esa línea crece en la segmentación. Eso es un efecto de la función erode de OpenCV, que usamos príncipalmente para que no se consideren pequeños defectos de la segmentación.

5. Conclusiones y mejoras pendientes

Pensamos, en conclusión, que nuestro algoritmo de segmentación obtiene buenos resultados con las pruebas llevadas a cabo. Es cierto que podría ser más rápido (especialmente en nuestra primera versión, de la primera entrega de esta parte), pero hemos tratado de encontrar un equilibrio entre resultados y tiempos de ejecución que nos permita emplear este algoritmo una vez usemos el robot real.

Cuánto a mejoras pendientes, pensamos que diversos puntos requieren nuestra atención. El principal de ellos es que la segmentación todavía está lejos de ser perfecta. Eso se debe principalmente a que un píxel, en su individualidad, no es capaz de expresar del todo bien la entidad a que pertenece. Pensamos utilizar esos datos de segmentación para entrenar un clasificador basado en contexto, que como mínimo pueda fijarse en los píxeles alrededor del que intenta clasificar. Una posibilidad un poco lejos dada la potencia de cálculo que tiene el robot sería utilizar un segmentador por convoluciones. Según nuestro análisis, sería una red de neuronas artificiales con una capa convolucional al principio y probablemente algunas de pooling o convolución más. Estimamos un total de 3 o 4 capas, lo que ya sería demasiado para el robot que disponemos, además que exigiría operar con bibliotecas

muy recientes que quizás no presenten demasiada compatibilidad con Python 2.7. Intentaremos, sin embargo, hacer esta mejora en el futuro.