Máster de Visión Artificial

Asignatura: Aplicaciones Industriales

Lectura de códigos de barras - Práctica obligatoria

Orión García Gallardo

INTRODUCCIÓN

El código de barras consiste en un sistema de codificación creado a través de series de líneas y espacios paralelos de distinto grosor. Generalmente se utiliza como sistema de control ya que facilita la actividad comercial del fabricante y del distribuidor, por lo que no ofrece información al consumidor, si no datos de operaciones aplicados a identificar productos, llevar control de inventarios, carga y descarga de mercancías, disminuir tiempos de atención en ventas, etc.

La tecnología óptica del código de barras fue desarrollada por primera vez por Bernard Silver y Norman Joseph Woodland en la universidad de Drexel Institute of Technology y su primer uso comercial fue en 1966 [1].

Se denomina simbología al mapeo entro los mensajes y los códigos de barras [2]. Hay numerosas simbologías, la más extendida es el código UPC que se inventó en 1973. Otros códigos comúnmente usados son el Code 128, Code 25, Code 93 [3] y el DataMatrix 2D.

En la actualidad se ha generalizado el uso de código de barras para la identificación de productos de venta. Sin embargo, en numerosas ocasiones, el empleado se ve obligado a teclear manualmente el código correspondiente al producto por no ser este detectado por el scanner. Esto repercute en la eficiencia del empleado al realizar su trabajo, siendo crítico en momentos de carga abundante de trabajo. Con esta práctica se pretende presentar una aplicación que, gracias al uso de cámaras, permita mejorar el rendimiento en la detección de códigos de barras de los productos.

DESCRIPCIÓN DEL MÉTODO DESARROLLADO

Para la implementación de esta aplicación se hace uso de las funciones que aporta la librería ImageJ [4]. ImageJ es un programa de dominio público de procesamiento de imágenes inspirado en NIHImage [5] para los Macintosh. Se ejecuta en cualquier ordenador que tenga instalada máquina virtual de Java 1.4 o posterior y permite aplicar todo tipo de operaciones tales como mostrar, editar, analizar, procesar, guardar e imprimir sobre imágenes de 8-bits, 16-bits y 32-bits.

El diseño de está practica tiene en cuenta una serie de acciones. Para describir el proceso se va a proceder de las acciones más generales a las más específicas y concretas. Primeramente hay que aclarar que los códigos a detectar van a venir determinados por la simbología Code 25. Esta simbología se caracteriza por tener dos tipos de barras (anchas y finas), ser continuo, numérico, y por tener bandas de inicio y final prefijadas (Fig 1).

Fig 1. Ejemplo de código de barras Code 25.

En el proceso general se intenta obtener el código de barras de una imagen. Si no se ha detectado ninguno o la eficiencia del código detectado es menor a un mínimo (que establecemos en un 5%) entonces se procede a aplicar una máscara o kernel a la imagen para mejorarla y se procesa de nuevo. La eficiencia de un código se define como el porcentaje líneas en la que ese código es reconocido con respecto al número total de líneas de la imagen. Por otro lado, el kernel con el que se convoluciona la imagen es aquel que permite propagar los puntos verticalmente para paliar las barras difuminadas. En este caso se ha aplicado:

$$\ker nel = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

La acción de obtener el código de barras de una imagen se describe a continuación. El primer paso corresponde a abrir la imagen y procesarla línea a línea. Como el procesamiento de cada línea de la imagen es independiente de las demás y con el objetivo de hacer más eficiente este proceso, se hace uso de programación concurrente. De esta manera se divide la altura de la imagen en 10 secciones y se crea un hilo de ejecución por cada sección. Cada hilo de ejecución nos devuelve el código detectado en esa sección y en el número de líneas que ha aparecido. Por lo tanto en este primer paso se crean los 10 hilos de ejecución, se mandan a ejecutar y se espera a que todos y cada uno de estos hilos haya finalizado. Una vez hayan acabado se combinan los resultados, es decir, se hace el sumatorio por código del número de apariciones en cada uno de los hilos ejecutados. A continuación se selecciona el código que más número de apariciones tenga.

Para el procesado de cada línea de la imagen se ha tenido en cuenta un autómata de detección de códigos (Fig 2). En la figura se puede ver un autómata sencillo de cuatro estados de los cuales 2 de ellos son finales. Uno de los estados finales es el estado 3, al que se llega cuando se ha conseguido detectar un código válido en la línea procesada, devolviéndose dicho código como resultado. Al otro estado final, el estado 4, se llega cuando la línea procesada no nos devuelve ningún código, devolviéndose la cadena vacía como resultado.

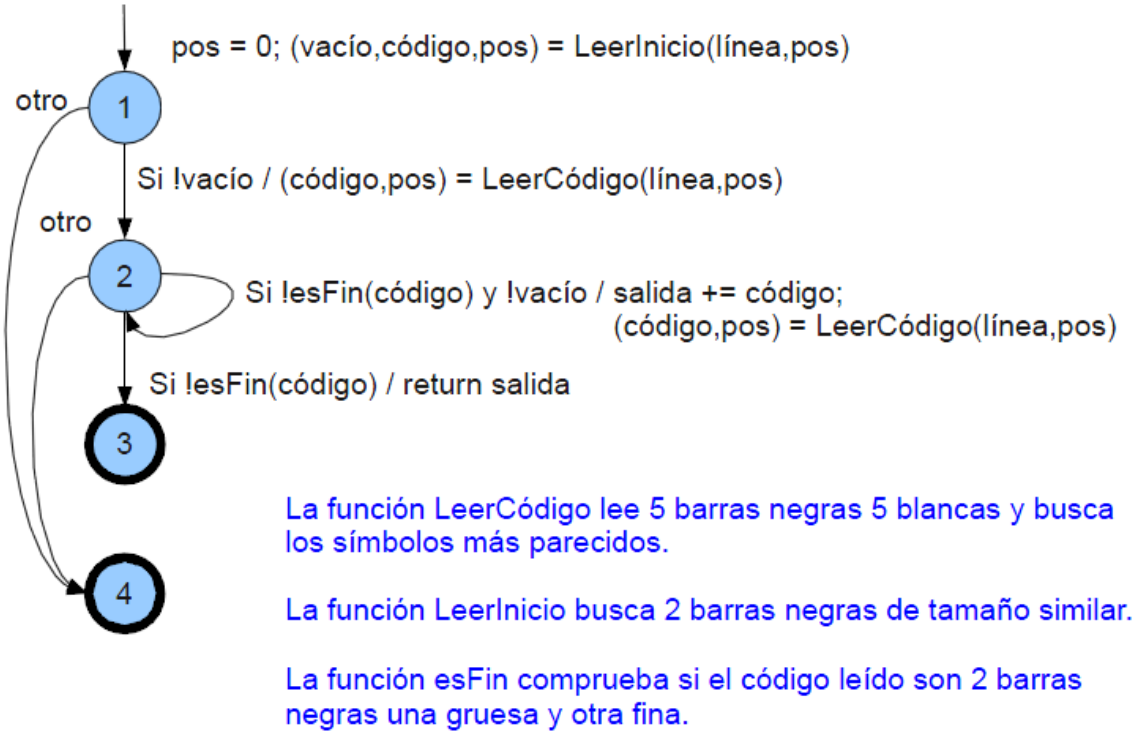

Fig2. Diagrama de estados al leer una línea.

Como paso previo a que el sistema entre en este flujo de estados el proceso agrupa los pixeles de cada línea. Este agrupamiento devuelve un vector con los tamaños de las transiciones encontradas en la línea especificada, comenzando por blanco. Por ejemplo, si devuelve: 0, 2 , 5, 2, 5, significaría: 0 píxeles blancos, 2 negros, 5 blancos, 2 negros, 5 blancos.

A continuación se procede a determinar el tamaño medio de cada barra. Para ello se descartan tanto el primer elento del vector como el último por corresponder estos a los margenes d ela imagen. Este tamaño medio de cada barra nos servirá para determinar si una barra es fina o gruesa dependiendo de si su grosor es menor o mayor que el de la media respectivamente.

Dentro del autómata merece especial mención la implementación de la función LeerCódigo. Esta fundión devuelve dos carácteres correspondientes a haber leído 5 barras negras y 5 barras blancas entrelazadas, a partir de una determinada posición en el vector de entrada. Para decodificar el vector se ha tenido en cuenta una tabla (Tabla 1). En ella, por cada dígito, se traduce cada patrón de código de barras a código binario y este a código decimal. Lo mismo hace la aplicación, coge esas 5 barras, a partir de una posición, traduce el patrón a código binario y este a codigo decimal. Una vez obtenido el código decimal se decodifica en el dígito o número que representa y se devuelve.

Number	Pattern	Code	Decoded
0	F F A A F	00110	6
1	A F F F A	10001	17
2	F A F F A	01001	9
3	A A F F F	11000	24
4	F F A F A	00101	5
5	A F A F F	10100	20
6	F A A F F	01100	12
7	F F F A A	00011	3
8	A F F A F	10010	18
9	F A F A F	01010	10

Tabla 1. Tabla de codigicación de dígitos.

PRUEBAS REALIZADAS SOBRE LA MUESTRA

En un principio se realizó la aplicación son las imágenes tal y como se encontraban en el directorio de entrada. Sin embargo, se detectó que en un número elevado de casos la eficiencia del código devuelto estaba por debajo del 5 % (Tabla 2). Como se aprecia en la tabla para cada imagen original el sistema devolvía códigos detectados en más de 5 líneas de cada 100 sólo en el 26.67% de los casos. Este porcentaje hace poco fiable los resultados obtenidos por lo que, en la mejora del sistema, a las imágenes que devolvían un porcentaje de eficiencia menor a 5 se les aplica el kernel anteriormente descrito y se procesan de nuevo. Los resultados obtenidos en este caso mejoran notablemente llegando a tener un 64.29% de los códigos devueltos una eficiencia mayor al 5%.

Máscara	Códigos reconocidos (%)	Más de 5 % eficiencia (%)
Sin kernel	66.67	26.67
Con ke rnel	66.67	64.29

Tabla 2. Resultados con el porcentaje casos exitosos.

Otro de los aspectos que se ha querido mejorar es el tiempo de ejecución del algoritmo. Para pocas imágenes el tiempo de ejecución es ínfimo y las reducciones que se puedan aplicar casi inapreciables. Sin embargo, este factor es clave cuando hay gran número de imágenes y cuando las imágenes identifican números de barras de más de 20 dígitos. Para mejorar el rendimiento en este sentido de la aplicación se ha introducido la concurrencia en el sistema. Como se mencionó anteriormente, este problema se presta muy bien a concurrencia ya que lo procesamientos de las líneas son independientes unos de otros. Los resultados obtenidos se

pueden apreciar en la tabla proporcionada (Tabla 3). De estos datos podemos deducir que a partir de los 10 hilos la mejora introducida por el procesamiento paralelo se ve contrarrestada por la gestión de estos. Otro aspecto importante a observable es que con 10 hilos se obtiene una mejora temporal de casi el 40% con respecto a la ejecución secuencial.

Number of Threads	Elapsed Time (ms)
1	3585
2	2405
4	2350
8	2341
10	2174
12	2708

Tabla 3. Tiempo ejecución por hilos de la aplicación.

CONCLUSIONES

- Resumen de resultados obtenidos.
  Según los resultados obtenidos, con este algoritmo se tarda de media unos 145 milisegundos para procesar cada imagen. Se ha obtenido un resultado fiable en al menos el 64% de los códigos de barras devueltos.

- Mejoras que podrían realizarse.
  El algoritmo presentado tiene algunas carencias que podrían solventarse en futuras implementaciones. Se ha observado que algunos de los códigos estaban torcidos o desviados en la imagen. Por lo tanto el porcentaje de códigos detectados con éxito mejoraría considerablemente con la introducción de un algoritmo que enderezara los códigos. En [2] se presenta un algoritmo bastante eficiente basado en la transformada de Hough centrado en esta tarea.

- Problemas que no se sabe cómo solucionar.
  En la mayoría de imágenes en las que no se ha podido devolver ningún código algunas barras aparecen superpuestas. Esto provoca que no se pueda distinguir entre barras gruesas o varias finas superpuestas.

REFERRENCIAS

[1] Adams, R. (2002). Barcode 1: Barcode History page. Retrieved from http://www.adams1.com/history.html

[2] S. M. Youssef and R. M. Salem (2007). Automated barcode recognition for smart identification and inspection automation, Expert Systems with Applications, 33(4): 968–977.

[3] Groover, M.P. (1980). Automation, production, systems and computer--integrated manufacturing. New Jersey: PrenticeHall.

[4] http://rsb.info.nih.gov/nih-image/

[5] T.W. Ridler, S. Calvard, Picture thresholding using an iterative selection method, IEEE Trans. System, Man and Cybernetics, SMC-8 (1978) 630-632.