G2 - Segundo Parcial - Visión

22.90 Automación Industrial

Profesores:

- Rodolfo Enrique Arias
- Federico Sofio Avogadro
- Mariano Tomás Spinelli

Integrantes

- Capparelli, Nicolás - 60066
- Lynch, John Matthew - 55011
- Mujica Buj, Juan Martín - 60050
- Olivera, Marcos - 60420
- Torres, Jorge Pedro - 60184

Introducción

Se solicita diseñar un algoritmo que permita identificar la posición de una cadena de letras, partiendo de una imagen y que pinte de verde todos los cuadrados que contengan las letras para formar esa cadena.

A modo de ejemplo, se debe obtener una solución equivalente de la imagen ‘BASE 2.jpg’ de la **Figura 1**, con la secuencia *ADYZXSWO,* dando como resultado la **Figura 2** o equivalentes**.**

El mismo se realiza con Matlab y el Toolbox de visión de Peter Corke, Machine Vision Toolbox.

El programa consiste de cinco etapas:

1. Generación de imágenes base sobre las que trabajar.
2. Procesamiento de las imágenes de entrada
    a. Extracción de templates de letras
    b. Identificación de las letras y su posición
    c. Identificación de los cuadrados negros en una matriz de 7x7.
    d. Asociación de cada cuadrado con su letra junto con su codificación.
3. Operaciones lógicas en base a los datos
    a. Generación de un camino y elección de los cuadrados a colorear.
4. Pintar dichos cuadrados.
5. Guardado de imágenes resueltas.

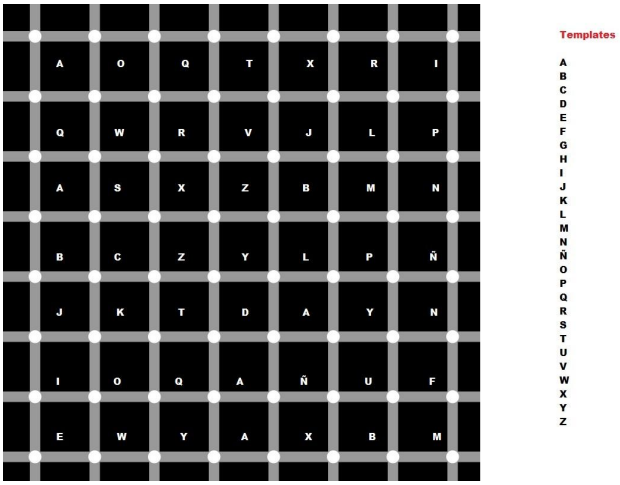


Figura 1: imagen ‘BASE 2’

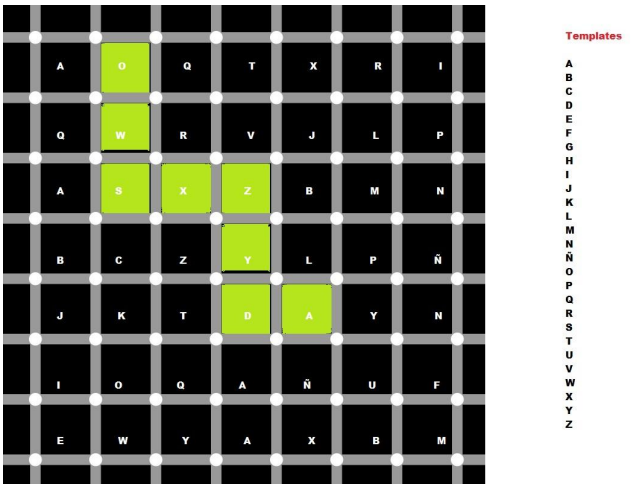


Figura 2: Solución posible esperada

Generación de imágenes

Además de trabajar con el *jpg* de entrada de la **Figura 1**, para probar el código y su robustez, se generan varias imágenes de características similares en base a los 'template fondo', de la **Figura 3**, bajo las siguientes hipótesis:

- Todos los cuadrados negros (grises oscuros) tienen una letra blanca (gris clara). Esta letra se genera con los templates de la derecha. Qué letra y en qué cuadrado es aleatorio.
- La posición de los cuadrados negros no cambia de forma significativa (en realidad no cambia en absoluto, dado que todas las imágenes son creadas a partir del mismo *'template fondo'*).
- La posición de las letras *template* de la derecha no cambian de forma significativa y las mismas están separadas entre sí por un par de píxeles.
- Las letras tienen una posición aleatoria dentro de cada cuadrado negro (aunque sin tocar sus bordes).
- En todas las imágenes siempre hay un *template* de letras a su derecha en la misma posición, correspondiente a las letras usadas en la grilla y ordenadas de la A-Z (con Ñ).
- La secuencia de letras pedidas es entre celdas adyacentes (arriba, abajo, derecha, izquierda), no se considera que dos celdas puedan unirse en la secuencia diagonalmente.

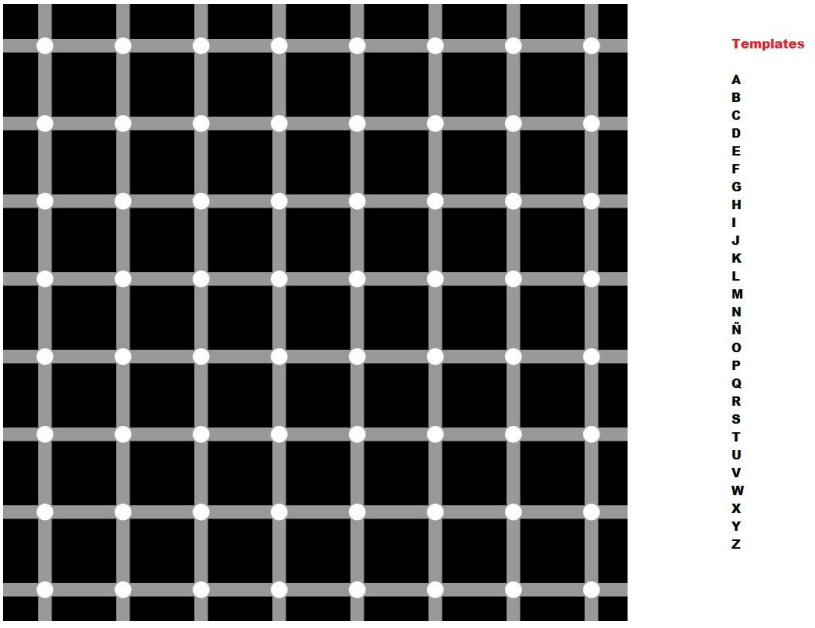


Figura 3: Template fondo

En el proceso de generar la imagen, se utilizan las mismas funciones de extracción de templates de letras y cálculo de propiedades (límites, centroides) de los cuadrados negros que serán usadas en la etapa siguiente. Ambas están basadas principalmente en la función *iblobs()* del Toolbox de Corke. Cabe aclarar que no se guarda ninguna información adicional sobre los templates de letras o cuadrados al generar estas imágenes. Sólo se obtiene un jpg similar a BASE 2.

En la práctica las imágenes generadas se obtienen con la siguiente función:

generarTest(cantidadImagenesDePrueba)

que genera *cantidadImagenesDePrueba* archivos .jpg similares la imagen 'BASE 2'. Esto se hace iterando en la función:

generar_imagen()

que toma la imagen de la **Figura 3**, la convierte a escala de grises, extrae los templates de las letras con la función:

extraer_letras(imGrey)

que devuelve imagenes de 25x25 píxeles con cada letra, identificada con blobs, en color blanco con fondo negro (invertidos del *Template fondo*).

Cabe destacar que, si bien no está implementado, el uso de esta función permitiría seguir aplicando el mismo programa aún así se cambie el tipo de fuente utilizado para las letras dentro de la cuadrilla.

y extrae de forma análoga los cuadrados negros y sus posiciones con la función:

extraer_posiciones_cuadrados(imGrey, *columnasGrillas, filasGrillas*)

que obtiene los delimitadores de cada cuadrado y su posición relativa en una grilla de *filasGrilla* x *columnasGrilla* (7x7 en este caso). Esto lo logra con el uso de blobs y filtrado por área.

Finalmente para cada cuadrado, se genera una letra aleatoria de las templates, ubicada en una posición aleatoria dentro del mismo, sin tocar los bordes.

Ejemplos de las imágenes generadas se ven en las **Figuras 4** y **5** a continuación.

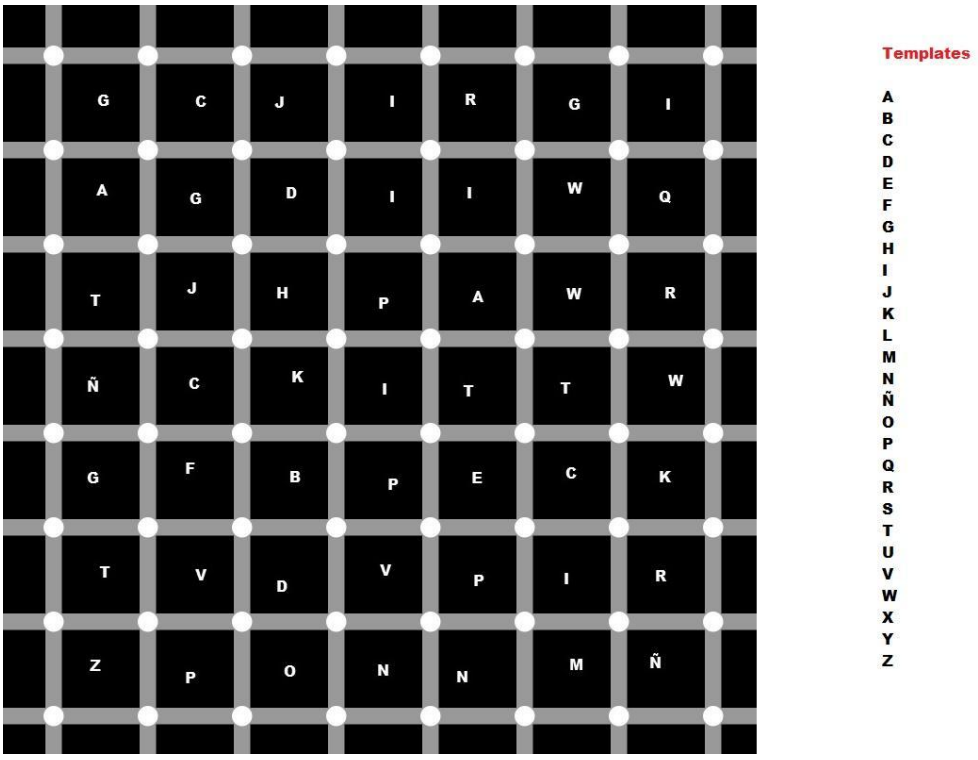
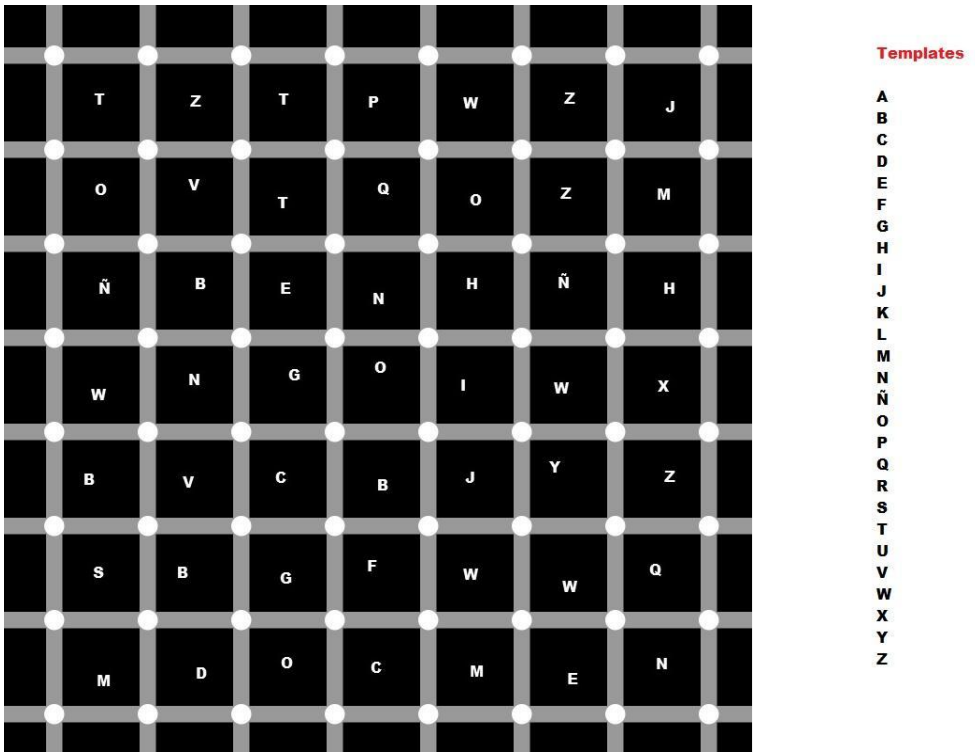


Figura 4: Ejemplos de las imágenes de prueba generadas



Figuras 5: Ejemplo de las imágenes de prueba generadas

Todas estas imágenes se generan con el formato *'testImage#.jpg'* dentro del directorio *'./testImages'* que se encuentra en la carpeta del programa.

Procesamiento de imágenes

Se selecciona una cadena de caracteres deseada para evaluar. Luego, para cada imagen disponible en el directorio *'./testImages'* se repite el esquema a continuación. Cabe repetir que en caso de imágenes generadas, se toma solamente el archivo *jpg* , sin referencias de cómo fue generado.

Primero se carga la imagen al espacio de trabajo y se la convierte a escala de grises para simplificar la identificación. A continuación, se identifica y guarda el *template* de letra a utilizar con:

letrasBlancas = *extraer_letras*(imGrey)

Luego se identifican y se guardan las posiciones absolutas, delimitadores y posiciones relativas de los mismos en una matriz de *filasGrilla x columnasGrilla* (7x7) con:

cuadrados = *extraer_posiciones_cuadrados*(imGrey, *columnasGrilla, filasGrilla*)

Desde este punto en adelante se trabaja con una matriz de 7x7 para llevar registro de las letras y en qué cuadrante se encuentran. Esto se hace con:

mLetras = get_matriz_letras(letrasBlancas,cuadrados,imGrey,columnasGrilla,filasGrilla)

que itera para cada cuadrado, asociado con un elemento en una matriz de *filasGrilla x columnasGrilla* (7x7) y por similaridad identifica qué letra contiene. Luego, convierte la letra a un valor contenido entre 1-27 representando de la 'A' a la 'Z'. Se aplica *isimilarity()* y se debe refinar los resultados con umbrales para distinguir bien casos muy semejantes (ej N y Ñ).

A modo de ejemplo si se tiene en la imagen las siguientes letras, la matriz de salida sería:

A B A 1 2 1

C C C → 3 3 3

E A A 5 1 1

Nota: La **grilla de letras** de la imagen es la propia imagen de entrada, no se obtiene una matriz de caracteres todavía.

Generacion del camino a pintar y coloreado

Conocida la matriz codificada del punto anterior, se la traduce en elementos tipo *chars*, para una comparación directa con la secuencia buscada.

A B A 1 2 1 'A' 'B' 'A'

C C C → 3 3 3 → 'C' 'C' 'C'

E A A 5 1 1 'E' 'A' 'A'

Si bien al determinar las letras en cada cuadrado se puede extraer la posición en la grilla 7x7 de cada uno, se elige crear una matriz 7x7 con los números del 1 al 27 representando cada uno de los templates, convertirlos a las letras correspondientes y luego usar esa matriz de letras para buscar la secuencia de letras deseada. Si bien esto resulta ligeramente más ineficiente, se considera más sencillo de realizar y las funciones se consideran de un uso más general. El algoritmo para encontrar el camino que recorre la secuencia se explica a continuación.

El algoritmo en cuestión se encuentra en la función:

[mSecuencia, error] = PathMatrixGenerator(secuencia,mLetras);.

Se comienza por encontrar las posiciones de la primera letra en la secuencia. Cada una de estas se guarda dentro de una matriz diferente, conformada por ceros y un 1 (representando un 'índice' para la primera letra) en la posición de la primera letra. Para la segunda letra, se recorre cada una de las celdas de cada una de estas matrices y se chequean las siguientes condiciones:

1) En esa posición, en la matriz de letras original, está la siguiente letra (segunda en este caso).
2) En esa posición, en la matriz que se está recorriendo, no hay ningún índice de letras anteriores (equivalente a pedir que haya un cero), lo que asegura que el camino no se cruce.

3) En alguna de las posiciones adyacentes (arriba, abajo, derecha, izquierda) está el número de secuencia de la letra anterior (para el caso de la segunda letra, equivale a pedir que haya un 1 en alguna de estas posiciones). Este último chequeo tiene 9 casos distintos donde hay que chequear 2, 3 o 4 lados: 4 esquinas, 4 bordes y el interior, respectivamente.

Si todas estas condiciones se cumplen, se genera una nueva matriz que incluye la matriz que se estaba chequeando y un nuevo número correspondiente a la letra (en el caso de la segunda letra, un 2) en la posición que cumplió estas condiciones.

Repitiendo esta lógica para todas las letras en la secuencia se genera una serie de matrices (denominada *Path*) con todas las secuencias válidas posibles de: la primera letra, la primera y segunda letra, la primera, segunda y tercera letra, etc. hasta todos los posibles caminos de la secuencia completa. Se elige una de estas últimas como salida de la función.

Dado que la serie de matrices guardan el índice de cada letra (lo cual permite identificar cada paso de la secuencia), es posible ver si la secuencia pedida existe o no en la matriz de letras, comparando el índice de la última letra encontrada con el largo de la secuencia. La **Figura 6** ilustra un ejemplo de aplicación del algoritmo desarrollado.

```
  7×7 char array

    'JXELSHU'
    'ZDWLNKE'
    'JDOIQFS'
    'YIWYTFF'
    'CLBSWDL'
    'DANQWWW'
    'MKDYFYD'

Ingrese la secuencia a buscar, con comillas
"MKDYFYDWWWWDLFFFKNLIODJZJXELSHU"
>> Path(:,:,end)

ans =

    25    26    27    28    29    30    31
    24     0     0    19    18    17     0
    23    22    21    20     0    16     0
     0     0     0     0     0    15    14
     0     0     0     0    11    12    13
     0     0     0     0    10     9     8
     1     2     3     4     5     6     7
```

Figura 6. Ejemplo de generación de camino a pintar.

Por consigna, se pintan de verde todos los casilleros de la grilla que contengan la secuencia. Cuando la secuencia pedida no existe en la matriz, se elige colorear la la maxima porcion de la secuencia que sí existe de amarillo y dar un mensaje de *secuencia no encontrada*. El pintado de los cuadrados se realiza con la función:

res_imColor = PintarCuadrado(imColor,cuadrados,pintar,color,columnasGrilla, filasGrilla)

que toma la imagen a color, los datos sobre ubicación de los cuadrados, una matriz con los cuadrados que debe pintar y finalmente un vector RGB con el color que debe pintar cada cuadrado. Devuelve una imagen con los cuadrados indicados pintados, del color indicado.

Se proporcionan 2 métodos de coloreado: un método donde se pinta toda la secuencia de un mismo tono de verde, y otro método donde se pinta la secuencia de verde, pero de distintas tonalidades generando un degradado de más oscuro a más claro siguiendo en orden la secuencia pedida.

Resultados

En la **Figura 7** se muestra la imagen de entrada BASE 2 con la secuencia pedida ADYZXSWO. Se omiten los templates de las letras en ambas imágenes para una mejor presentación.

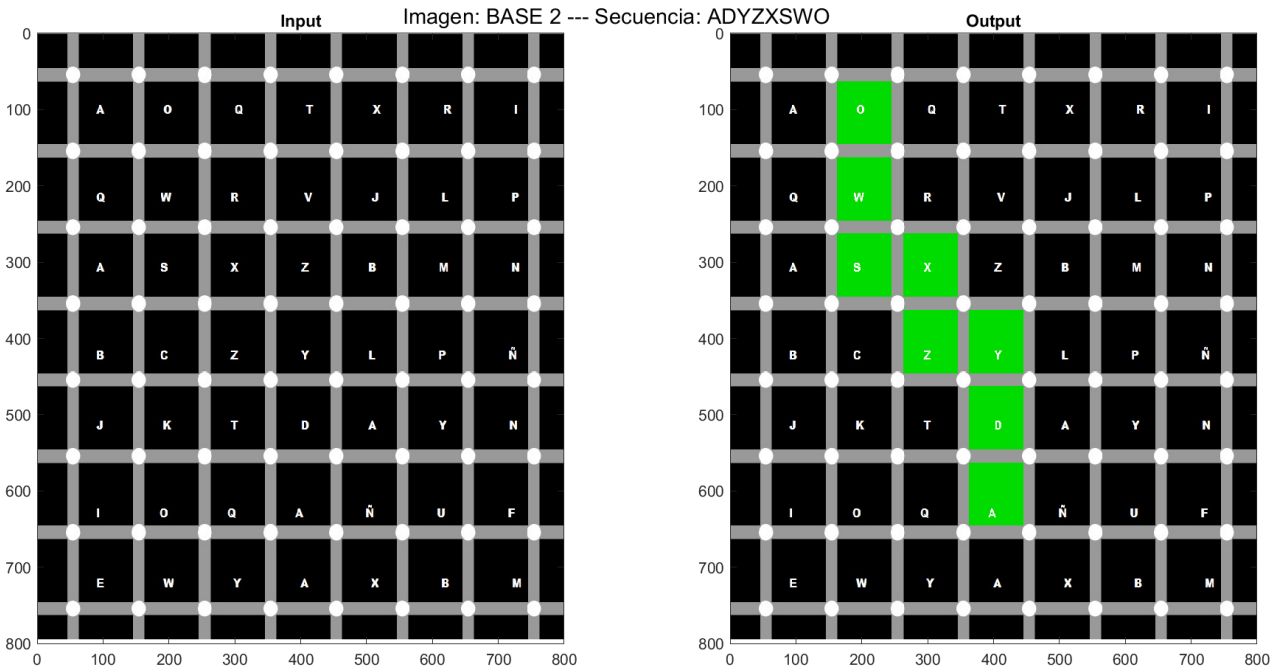


Figura 7: Cadena completa detectada y pintada de verde, según consigna.

Nótese que no es la misma solución encontrada en el ejemplo dado en la consigna, pero la secuencia es correcta (toma la A de abajo en lugar de la A de la derecha y la Z de la izquierda en lugar de la Z de arriba).

A continuación se muestran resultados para los distintos modos de visualización discutidos en la sección anterior: un degradado en el orden de secuencia (de oscuro a claro), una secuencia parcialmente dentro de la grilla (si inicio pero no final) y una con secuencia completamente fuera de la grilla.

En la **Figura 8** se demuestra que se pudo detectar de forma satisfactoria la secuencia “QWSXZYLAÑ”.

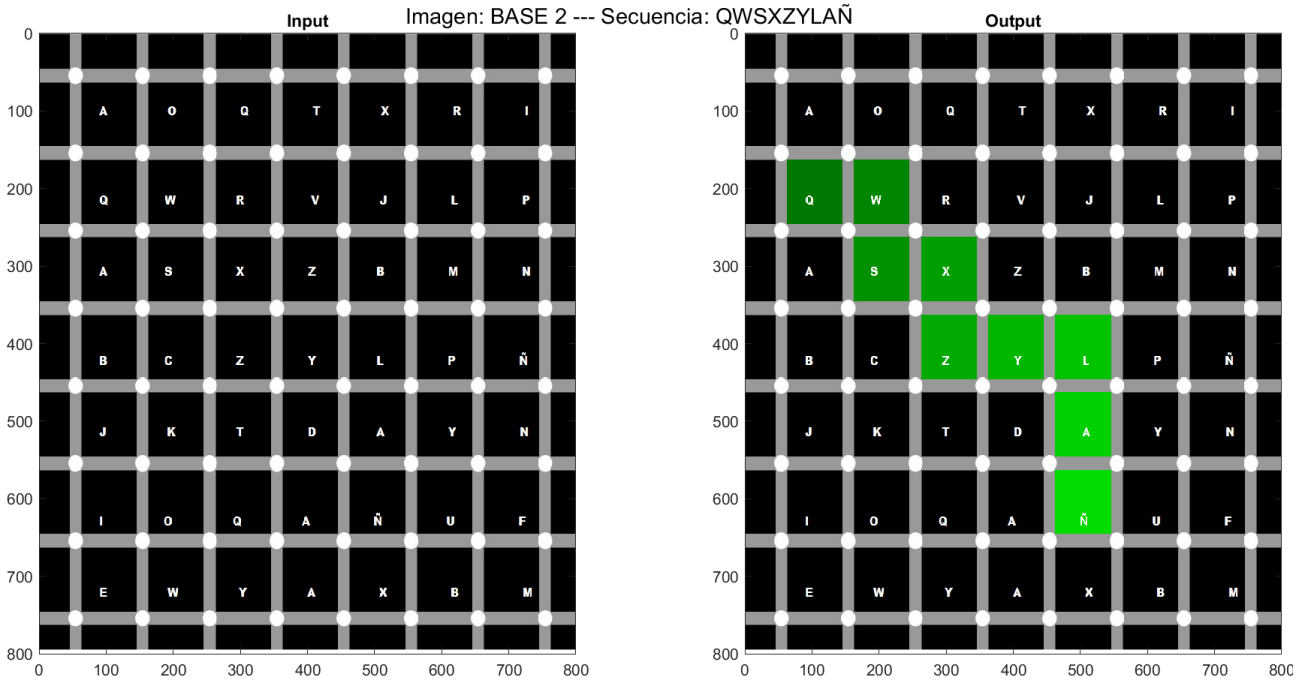


Figura 8: Cadena completa detectada y pintada de verde con *degradé* en orden.

Por otro lado, en la **Figura 9** se puede observar como la secuencia no fue detectada de forma completa, y por lo tanto se coloreo de amarillo solo el largo de la secuencia que se pudo completar.

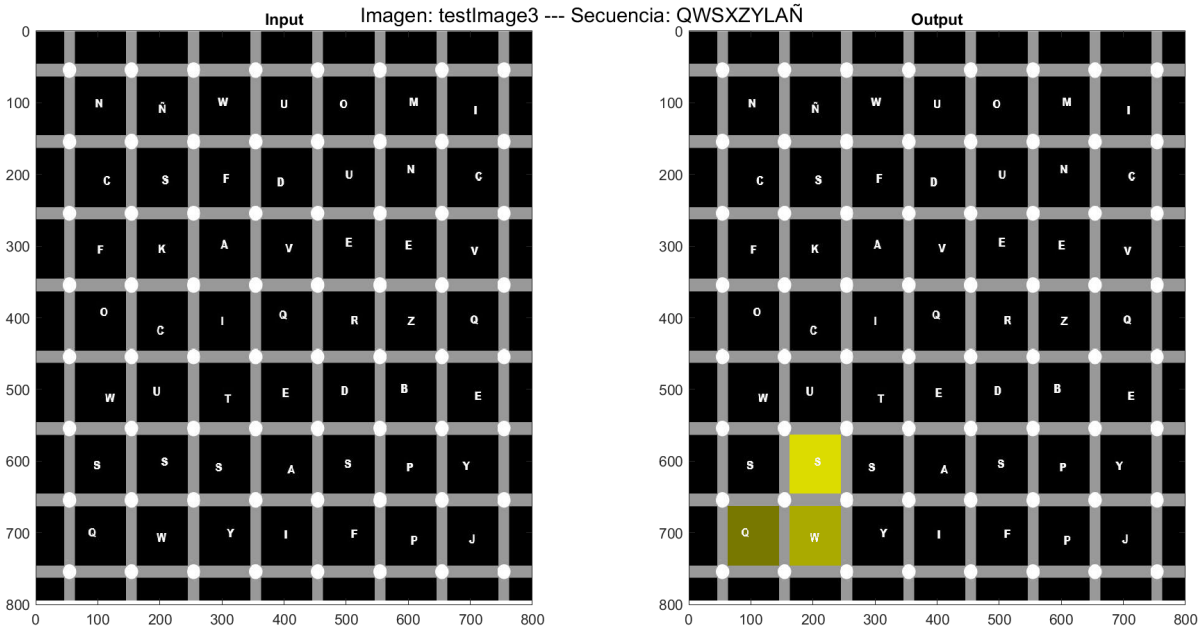


Figura 9: Cadena parcialmente en la grilla y pintada de amarillo con *degradé* en orden.

Finalmente, en la **Figura 10** se observa que, en el caso de no detectar la secuencia en absoluto, no se colorea ningún casillero.

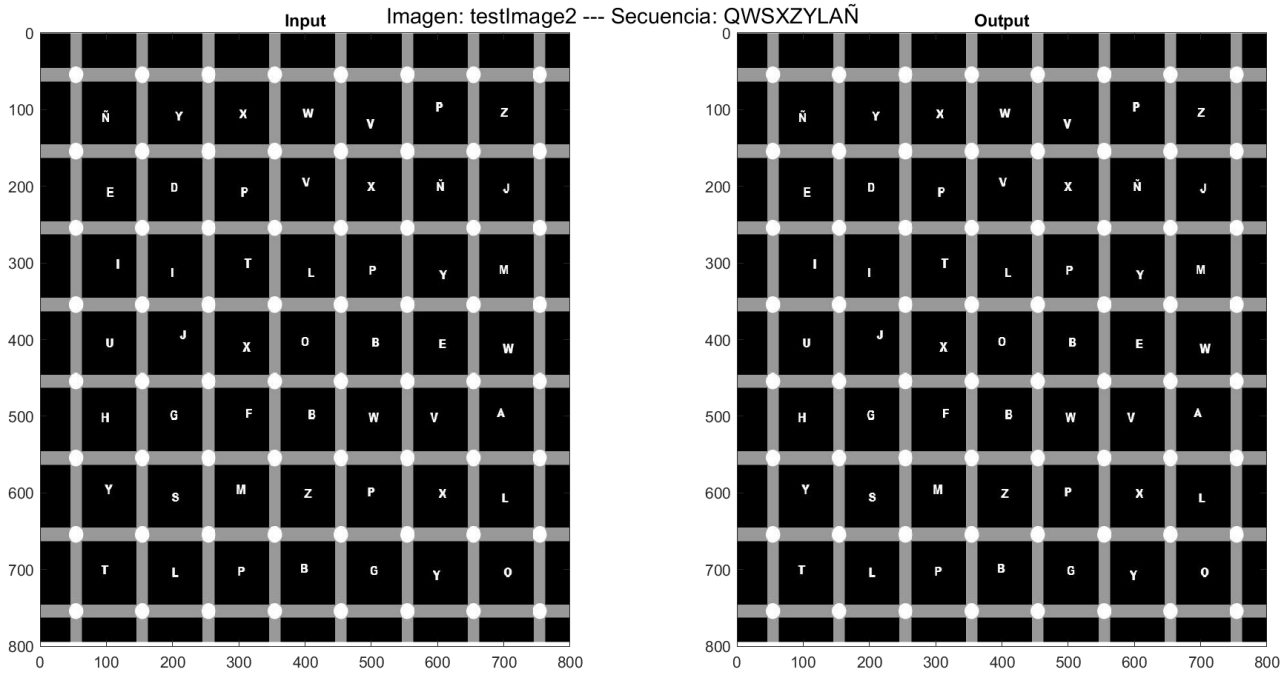


Figura 10: Cadena completamente fuera de la grilla

Finalmente se encuentra en el directorio entregado un compilado de más resultados con imágenes generadas con posiciones aleatorias y distintas secuencias.

Conclusión

Se lograron los objetivos planteados. El algoritmo logra satisfactoriamente reconocer la imagen, encontrar la secuencia pedida si es posible y pintar la imagen correctamente. También es lo suficientemente robusto como para detectar sí la secuencia no está en su totalidad, encontrar la máxima posible y pintarlo en un color diferenciado. Además puede reconocer las letras aunque estas no estén en el centro de su respectivo recuadro. El algoritmo reconoce exitosamente que puede haber múltiples secuencias, que las mismas pueden tener cualquier largo deseado, y puede manejar situaciones extremas en las cuales la secuencia no exista, o no esté en su totalidad.

Se puede agregar más funcionalidad al sistema si se adecúa una plantilla para un cambio de fuente, color, tamaño, de letra ya que para cada imagen se puede adoptar y reconocer un *template* diferente. Las funciones utilizadas permiten la generación de grillas de distinto tamaño, aunque esto no fue probado debido a utilizar siempre el mismo fondo. Dado que se pueden obtener las grillas en estados intermedios en el proceso de obtención de camino a pintar, se podría generar también imágenes/videos donde se muestre el proceso de generación de dicho camino.