|  |
| --- |
| MISE: Inteligencia Automática.  Implementación de K-NN sobre el PIC24HJ256GP610A |

|  |
| --- |
|  |
| Iñigo Olaso  José Ángel Castán |
| Fecha: 11/06/2018 |



**ÍNDICE**

[1 Objetivo 3](#_Toc516604722)

[2 Introduccion 4](#_Toc516604723)

[2.1 Desarrollo 5](#_Toc516604724)

[2.2 Reduccion de patrones: 5](#_Toc516604725)

[2.3 Funcionamiento: 6](#_Toc516604726)

[3 Analisis de los resultados 7](#_Toc516604727)

[4 Problemas encontrados 8](#_Toc516604728)

[5 Conclusiones 9](#_Toc516604729)

# Objetivo

El objetivo de este documento es definir los pasos seguidos a la hora implementar un algoritmo K-NN con distancia euclídea sobre un PIC. Para ello en primer lugar analizaremos un pequeño diagrama de flujo del programa completo. Y en segundo lugar analizaremos los datos obtenidos en test sobre el PIC con los datos obtenidos mediante un programa escrito en JAVA en el PC.

# Introduccion

Se ha elaborado un diagrama de flujo donde se analiza el flujo secuencial de la máquina de estados. Es decir, se han definido de una forma comprensible las diferentes acciones que se realizan durante la ejecución del programa. Desde la recepción de los datos hasta el tratamiento de los resultados.

Con el fin de simplificar la comprensión del flujo se ha omitido la configuración de los parámetros para el cambio de K-NN y zonig por UART.

La implementación de este flujo se encuentra en la carpeta *knn*, donde se encuentra un proyecto para abrirse en MPLAB.



Figura 1 Diagrama de flujo.

## Desarrollo

Durante el desarrollo de la implementación se han seguido los siguientes pasos:

1. Desarrollo en Java de un pequeño programa que nos reduce los patrones de referencia del fichero de training.

2. Implementación de la bufferizacion por UART de cada matriz.

3. Adaptación de las funciones de preprocesamiento: Eliminar Basura, Giro, y Ajustar.

4. Adaptación de la función zoning.

5. Implementación del cálculo de la distancia euclídea.

6. Comprobación del sistema con el fichero de training y solución de problemas hasta conseguir el 100% sobre el propio fichero de training.

7. Implementación del algoritmo K-NN y comprobación del funcionamiento sobre test.

En los pasos 3 y 4 se han adaptado las funciones heredadas, a un sistema sin gestión de ficheros, y se ha reducido la creación de matrices innecesarias, debido a limitación de memoria.

## Reduccion de patrones:

Al estar trabajando en un sistema con limitación de memoria, hemos desarrollado un algoritmo que reduce un conjunto de patrones a un subconjunto reducido del mismo. Este subconjunto cumple la condición de asignar correctamente a cada uno de los patrones originales su clase correctamente. Se incluye a continuación un esquema del funcionamiento del algoritmo.

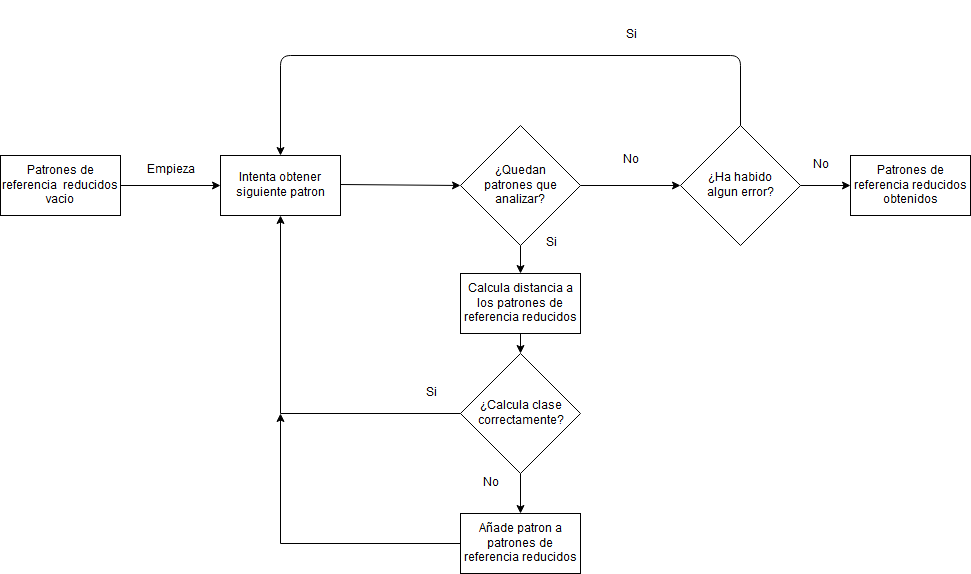


Figura 2 Diagrama de flujo.

El conjunto inicial de patrones tenía 2000 patrones. Aplicando este algoritmo, el subconjunto final obtenido según los zoning realizados es el siguiente

|  |  |  |
| --- | --- | --- |
| **4x3** | **8x5** | **13x8** |
| 256 | 145 | 166 |

La implementación de este algoritmo se ha realizado en Java, cuyos archivos fuentes se encuentran en la carpeta *ReducePatrones.* Estos patrones reducidos en el código se encuentran en el fichero *patrones\_reducidos.h*

## Funcionamiento:

Se ha dotado al programa de una serie de comandos para poder configurar los parámetros de cálculo a partir del puerto UART.

A continuación, explicaremos la interfaz de usuario:

|  |  |
| --- | --- |
| **Comandos** | **Descripción:** |
| z(zonig) | Incrementa el valor de zoning y visualiza la selección. |
| c(clear) | Resetea los resultados. |
| k(k-nn) | Visualiza la configuración de k-nn seleccionada. |
| +/- | Con + se incrementa el valor de k y con – se decrementa. |
| r(resultados) | Visualiza los resultados obtenidos. |

# Analisis de los resultados

Para la realización de las pruebas, se ha usado un conjunto de test de 500 patrones. A la hora de transmitir estos patrones se ha diseñado un pequeño script en PowerShell. Dicho script se encuentra adjunto bajo el nombre *escribirFichero.ps1*. Se ha comprobado el correcto funcionamiento de los filtros y resultados con la equivalencia en Java.

Zonig 4x3:

|  |  |  |  |
| --- | --- | --- | --- |
| K-NN | Aciertos | Total | Porcentaje |
| 1-NN | 425 | 500 | 85 |
| 3-NN | 431 | 500 | 86 |
| 9-NN | 425 | 500 | 85 |

Zonig 8x5:

|  |  |  |  |
| --- | --- | --- | --- |
| K-NN | Aciertos | Total | Porcentaje |
| 1-NN | 462 | 500 | 92 |
| 3-NN | 443 | 500 | 88 |
| 9-NN | 425 | 500 | 85 |

Zonig 13x8:

|  |  |  |  |
| --- | --- | --- | --- |
| K-NN | Aciertos | Total | Porcentaje |
| 1-NN | 462 | 500 | 92 |
| 3-NN | 448 | 500 | 89 |
| 9-NN | 428 | 500 | 85 |

# Problemas encontrados

A lo largo del desarrollo del programa nos hemos encontrado con diversos problemas, la mayoría relacionados con la reducida memoria del microcontrolador escogido. A continuación, hemos elaborado una lista con los problemas encontrados enumerados cronológicamente.

1. Adaptación librerías PROCAR: Adaptar las funciones con uso de un sistema de fichero a otro más sencillo sin dicho sistema.

2. Uso de matrices dinámicas para la optimización de memoria usada: Implementación de matrices dinámicas y gestión de la liberación de las mismas para reducir el espacio usado en función de las exigencias.

3. Adaptación de las librerías anteriores a las matrices dinámicas: La finalidad de las matrices dinámicas es usar un solo canal para la bufferizacion de las operaciones en lugar del sistema de ficheros. Esto nos ha obligado a comprender en ciertas situaciones el funcionamiento de las librerías PROCAR heredadas.

4. Introducción de patrones referencia: El hecho de querer introducir el cálculo de diversos zonig nos ha obligado a meter diferentes matrices de patrones referencia. Esta cantidad de datos ocupa un tamaño considerable por lo que una vez más hemos tenido que optimizar algunas variables para poder introducir estos datos.

5. Implementación y carga de las matrices: La bufferizacion de los datos se realiza por UART en un matriz dinámica. Además, también se han desarrollado algunos comandos de configuración explicados en el apartado 2. La bufferizacion de cada matriz requiere de la identificación de la misma mediante un identificador #0000, seguido de la obtención del tamaño de la matriz XX XX, acompañado de la clase XX, finalizando con la matriz completa. La verificación de la bufferizacion de estos datos sobre una matriz dinámica nos ha llevado a tener que comprobar en los registros de memoria la carga de cada parámetro. Este método es algo engorroso y abstracto ya que tienes que buscar la dirección de carga a través de los punteros y demás, aun así, es la mejor opción de cara a obtener y garantizar buenos resultados.

Problemas al adaptar las funciones, aunque no fuese necesario comprender completamente el código proporcionado por PROCAR. Al modificar a tan nivel el programa había que entender el funcionamiento para poder respetar las condiciones previas a los cambios.

# Conclusiones

Con los resultados obtenidos se puede suponer que los patrones de referencia son muy significativos, ya que obtienen un gran índice de aciertos, con un mínimo de un 85%. Se puede concluir también que son representativos por si mismos, ya que al aumentan el knn, no solo mejoran muy poco los resultados, sino que empeoran bastante. También se observa que se con un zoning de 4x3 se obtienen casi los mismos resultados que 8x5 y 13x8, por lo que es un buen conjunto para un sistema con limitaciones en procesamiento y memoria. Esto se debe a que, aunque el conjunto se reduce un 43% y un 35%, lo cual podría parecer que reduce el espacio, pero aumentan las dimensiones un 233% y un 766%, sin contar el tiempo de computo extra de calcular distancia euclídea de tantas dimensiones.

Por lo tanto, el conjunto ideal sería el 4x3, a no ser que fuese necesario obtener un porcentaje de acierto del 90%, que se debería escoger 8x5.