MATERIA OPTATIVA II
PROYECTO FINAL PRIMER PARCIAL

Estévez Santiago
Lozada Jhonny
Maya Vicente

14 de diciembre de 2018

1. Introducción

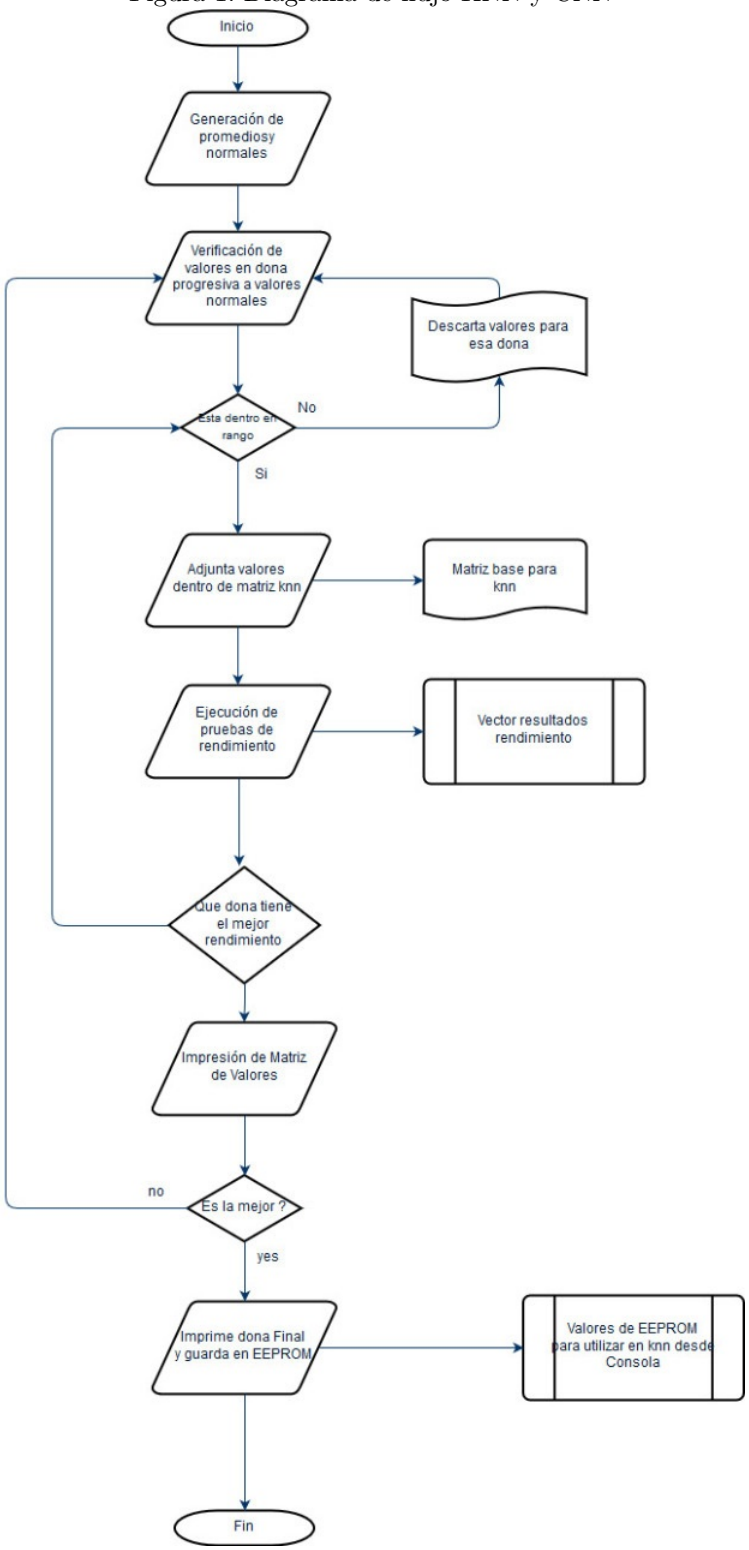
El machine learning, en español aprendizaje es un sub-campo de las ciencias computacionales con varias aplicaciones a diferentes campos de la ingeniería.
Lo que se desarrollo a continuación es un programa el cual puede clasificar diferentes objetos a través de medidas, de diferentes parámetros que se encuentran almacenados en una base de datos, ademas, selecciona y filtra datos de la base de datos la cual esta guardada una matriz, la cual hace que se los datos queden reducidos a los mejores y de esta manera ahorrar recursos y mejorar el rendimiento de nuestro microprocesador Arduino Mega.
El sistema funciona con una matriz de [120][5] la cual es nuestra base de datos mas grande, el programa selecciona los mejores datos a partir de donas, las cuales están ubicadas en medio de los valores y se mueva al rededor de toda la matriz, buscando el mejor centro para la obtención de datos a futuro de una mejor matriz sin muchos valores, de esta manera se automatiza el ingreso de datos.

2. Diseño del Sistema

2.1. Diagrama de Flujo y Diagrama de Bloques KNN

Diagrama de flujo realizado para KNN y CNN unificado

Figura 1: Diagrama de flujo KNN y CNN

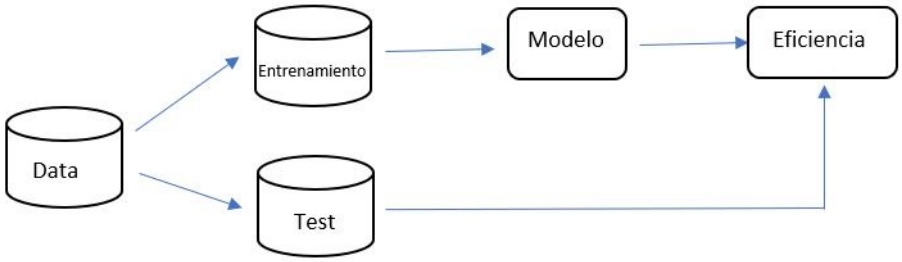


Lo principal que se debe tener es la base de datos en este caso esta compuesta de [120][5], de la cual para obtener sus centroides, debemos primero sacar los promedios de cada columna de datos con excepción de la columna de etiqueta, una vez tenemos el promedio se prosigue con las distancias con la fórmula previamente vista y sacamos el valor máximo del grupo de datos, con estos se los procede a normalizar los valores y por último se obtiene los centroides, estos se los realiza por el método de donas, para este caso se ha realizado que éstas donas empiecen en un rango de 0-0,1 y así sucesivamente ir aumentando 0,1 cada rango, de todo el grupo de datos que llegamos a obtener realizamos la mejor matriz.
Para obtener la mejor matriz se aplica KNN a cada una de las matrices generadas con las donas, para saber cuál es la mejor debemos tener en cuenta la eficiencia que aportan cada una de ellas, una vez el programa entiende los cálculos obtiene la mejor matriz y todo esto se realiza de manera autónoma.

Ahora finalmente al ingresar un valor de prueba si este es erróneo a la base S y en caso de ser correcto va a la matriz D.

Diagrama de Bloques del sistema KNN

Figura 2: Diagrama de bloques KNN



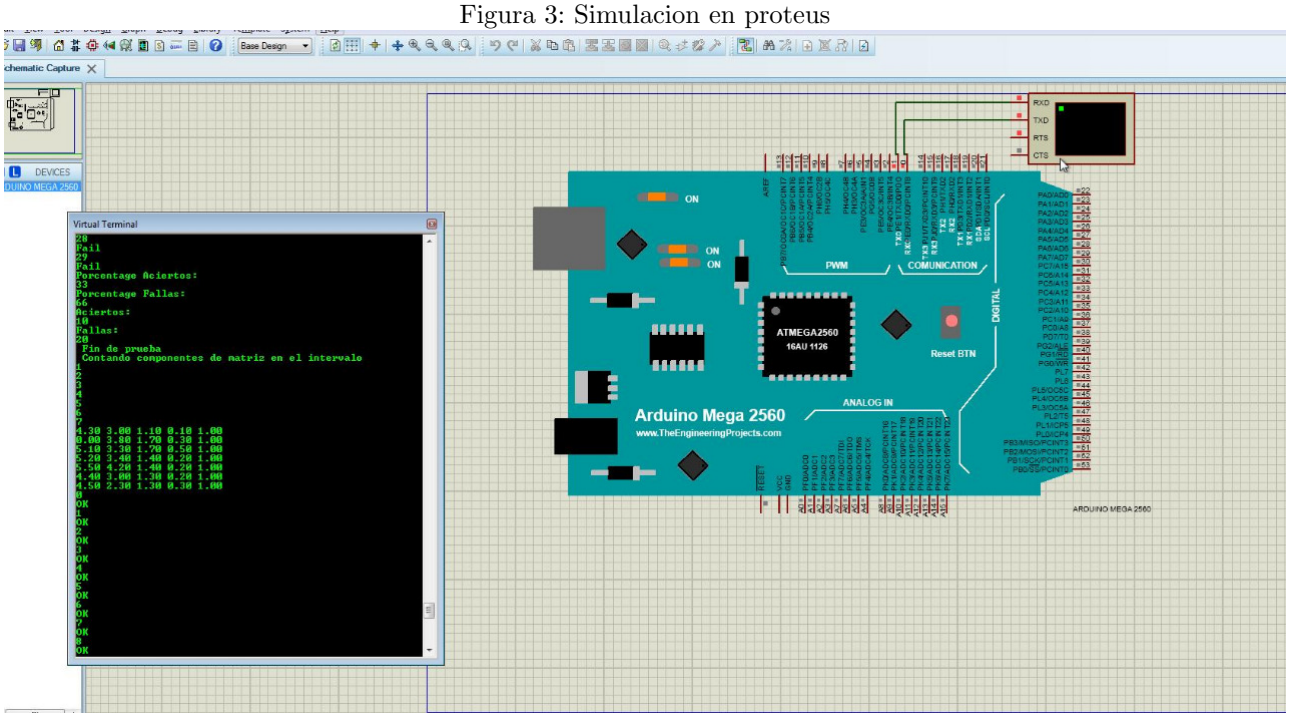
Para realizar KNN lo primero que debemos tener son nuestros datos divididos en dos base de datos, la primera será la base de entrenamiento y la segunda la base de test o de prueba, pero con la que se trabaja es con la matriz de entrenamiento en donde realizando un modelo de donas obtenemos los centroides para generar la mejor matriz de referencia, al tratarse del algoritmo del vecino más cercano nosotros podemos escoger cuántos vecinos necesitamos, en nuestro caso serán 3 vecinos.

3. Desarrollo

3.1. Simulación

Simulacion de KNN y CNN

Figura 3: Simulacion en proteus



4. Análisis de Resultados

El sistema Funciona de tal manera que al ejecutarlo recorre toda la matriz original en búsqueda de los mejores datos para generar una nueva matriz y de esta manera cree otra la cual será la adecuada con el fin de ser mas exacta y ocupe menos recursos.

A continuación se muestra todo el código de programación el cual se encuentra comentado y se explica por si solo.

Figura 4: Código Parte 1

```
#include<math.h>//libreria pow

//---------------------/Ingreso Librerias
#include "matrices.h"
#include "test.h"
#include "train.h"
#include "reducida.h"
#include "temp.h"
```

Figura 5: Código Parte 2

```
//-------------------Variables

//-----delimitaci'on tipo 1
int type1min = 0;
int type1max = 40;
//-----delimitaci'on tipo 2
int type2min = 41;
int type2max = 80;
//-----delimitaci'on tipo 3
int type3min = 81;
int type3max = 120;

float sumatoria = 0.0;
float promedio = 0;

float distancia = 0;

float max1 = 0;
float max2 = 0;
float max3 = 0;


float normal1[40];
float normal2[40];
float normal3[40];
```

Figura 6: Código Parte 3

```
//------------ DONA


// valores

int result=0;
int aciertos=0;
int fallos=0;
int cont_efl=0;

//----------------------Fin Variables
//----------------------Matrices aux

float pr_norl[5];
float pr_nor2[5];
float pr_nor3[5];


float distanciasl[40];
float distancias2[40];
float distancias3[40];
```

Figura 7: Código Parte 4

```
//---_______DONA

float donasl[40][3];
int cont_pos=0;
//-----------efectividad

int exactitud[10];

//union para an'alisis

float matrixu[80]={};

//-----------------------Fin Matrices Aux
```

Figura 8: Código Parte 5

```
void setup() {
  // -----------------Inicializo Serial
  Serial.begin(9600);
  //-------------test


  //--------------------------------- obtencion de valores promedio

  //------------------promedios tipo 1



  for (int r = 0; r < 4; r++) {
    for (int a = 0; a < 40; a++) {

      sumatoria += base[a][r];

     // Serial.println(sumatoria);

    }
    pr_nor1[r] = (sumatoria / 40);
   // Serial.println("Promedio");
   // Serial.println(pr_nor1[r]);
    sumatoria = 0;
    //   delay(500);

  }
```

Figura 9: Código Parte 6

```
//------------------Promedios Tipo 2


for (int r = 0; r < 4; r++) {
    for (int a = 40; a < 80; a++) {

      sumatoria += base[a][r];

    //  Serial.println(sumatoria);

    }
    pr_nor2[r] = (sumatoria / 40);
 //   Serial.println("Promedio");
//Serial.println(pr_nor2[r]);
    sumatoria = 0;
    //   delay(500);

  }
```

Figura 10: Código Parte 7

```
//---------obtencion de distancias tipo 1 -



  for (int d = 0; d < 40; d++) {


    distancia = sqrt(pow(base[d][0] - pr_nor1[0], 2) +
                     pow(base[d][1] - pr_nor1[1], 2) +
                     pow(base[d][2] - pr_nor1[2], 2) +
                     pow(base[d][3] - pr_nor1[3], 2));
  //  Serial.println(distancia);
    // delay(200);
    distancias1[d] = distancia;
  }


  //   maximo 1

  max1 = distancias1[0];

  for (int o = 1; o < 40; o++) {

    if (max1 < distancias1[o]) {

      max1 = distancias1[o];
     // Serial.print(max1);
    }
```

Figura 11: Código Parte 8

```
//---------obtencion de distancias tipo 2


  for (int d = 40; d < 80; d++) {


    distancia = sqrt(pow(base[d][0] - pr_nor2[0], 2) +
                     pow(base[d][1] - pr_nor2[1], 2) +
                     pow(base[d][2] - pr_nor2[2], 2) +
                     pow(base[d][3] - pr_nor2[3], 2));
   // Serial.println(distancia);
    // delay(200);
    distancias2[d-40] = distancia;
  }


  //   maximo 1

  max2 = distancias2[0];

  for (int o = 1; o < 40; o++) {

    if (max2 < distancias2[o]) {

      max2 = distancias2[o];
    //  Serial.print(max2);
    }

  }
```

Figura 12: Código Parte 9

```
//---------------vector normalizadas tipo 1


for (int n = 0; n < 40; n++) {

  normal1[n] = (distancias1[n] / max1);
 // Serial.println(normal1[n]);

}



//---------------vector normalizadas tipo 2


for (int n = 0; n < 40; n++) {

  normal2[n] = (distancias2[n] / max2);
  //Serial.println(normal2[n]);

}
```

Figura 13: Código Parte 10

```
//-------------------dona con valores


Serial.println("Dona Valores" );

    for(float fd=0.0;fd<1;fd+=0.1){

      for(int pos=0;pos<80;pos++){

       if(matrixu[pos]>=fd && matrixu[pos]<=(fd+0.1f)){
          matriz[cont_pos][0]=base[pos][0];
          matriz[cont_pos][1]=base[pos][1];
          matriz[cont_pos][2]=base[pos][2];
          matriz[cont_pos][3]=base[pos][3];
          matriz[cont_pos][4]=base[pos][4];
          cont_pos++;
          sizel++;

          Serial.println(sizel);
          //delay(500);

       }
```

Figura 14: Código Parte 11

```
}

for(int r=0;r<=(sizel-1);r++){

    for(int e=0;e<5;e++){

      Serial.print(matriz[r][e]);
      Serial.print(" ");
    }
  Serial.println(" ");
}


exactitud[cont_efl]=efectividad();
sizel=0;
fallos=0;
aciertos=0;
matriz[50][5]={};
//Serial.println(exactitud[cont_efl]);
cont_efl++;
cont_pos=0;
}
```

Figura 15: Código Parte 12

```
void loop() {

}


float knn(int fila, int col, int k, int label, float datos[]){//parametros de entrada , datos=nueva infromación
  int c=0;// movernos en columnas
  int f=0;//movernos en fila

  float k_vecinos_dist[k];//vector de almacenamiento de k distancias menores
  float etiquetas[2][label];// matriz y conteo de etiquetas
  float dist=0; //variable que almacena cada distancia.
  float dist_total=0;// caribel para almacenamiento distancai
  //lenar vector k_vecinos_dist con valores altos
  float eti_menor[k];//vector de eqtiquetas de distancia menor
  int k_cont=0;//contador de k
  int i=0;//contador
  float clase; //return etiqueta
  float comparacion;// comparar k-vecino mayor
  for (;c<k;c++)
  {
    k_vecinos_dist[c]=2000.0+c; // valores altos y ordenados
  }
  c=0;//reiniciar variable
```

Figura 16: Código Parte 13

```
c=0;//reiniciar variable
//llenado de las k distancias menores
for(;c<label; c++){
  etiquetas [0][c]=c+1;//lleno con valores d eetiqueta
  etiquetas [1][c]=0;//lleno con el conteo de etiquetas
  }
  c=0;// reinicio de variabel
  // distancia mas corta del nuevo punto hacia la amtriz
  for(;f<fila;f++)
  {
    for(;c<col-1;c++){
      dist=dist+pow(matriz[f][c]-datos[c],2);//distancia entre dos puntos
      }
      dist_total=sqrt(dist);//raíz total de la formula
      //Serial.print(dist_total);
      for(;k_cont<k;k_cont++){// determinar las k distancias enores y ordenarlas
        if(dist_total<k_vecinos_dist[k_cont]){
          k_vecinos_dist[k_cont]=dist_total;// asignar nuevo valor a vetor de dsitancias
          eti_menor[k_cont]=matriz[f][col-1];//col=5,4 //ya se ordenos el vector
          }
        }
        k_cont=0;
        dist=0;
        dist_total=0;
        c=0;
    }//for fila

    // comparar cada uno de los posiciones d ele matriz y comparar
```

Figura 17: Código Parte 14

```
    //seleccion del k vecino
    for(;c<label;c++){
      for(;k_cont<k;k_cont++){// recorro cad posicion de eti_menor y comparo con etiquetas y cuento si son igua
        if(etiquetas[0][c]==eti_menor[k_cont]){
          i++;
          etiquetas[1][c]=i;
          }
        }
        k_cont=0;
        i=0;
      }
      //c=0;
      c=1;//para que en comparacion
      comparacion=etiquetas[1][0];
      clase=etiquetas[0][0];
      for(;c<label;c++){
        if(etiquetas[1][c]>comparacion){//omparacion entre valores de suma de etiqueta
       clase=etiquetas[0][c];//ponga neuva clase
       comparacion=etiquetas[1][c];

       }
        }
        comparacion=0;
        c=1;
        return clase;

}
```

Figura 18: Código Parte 15

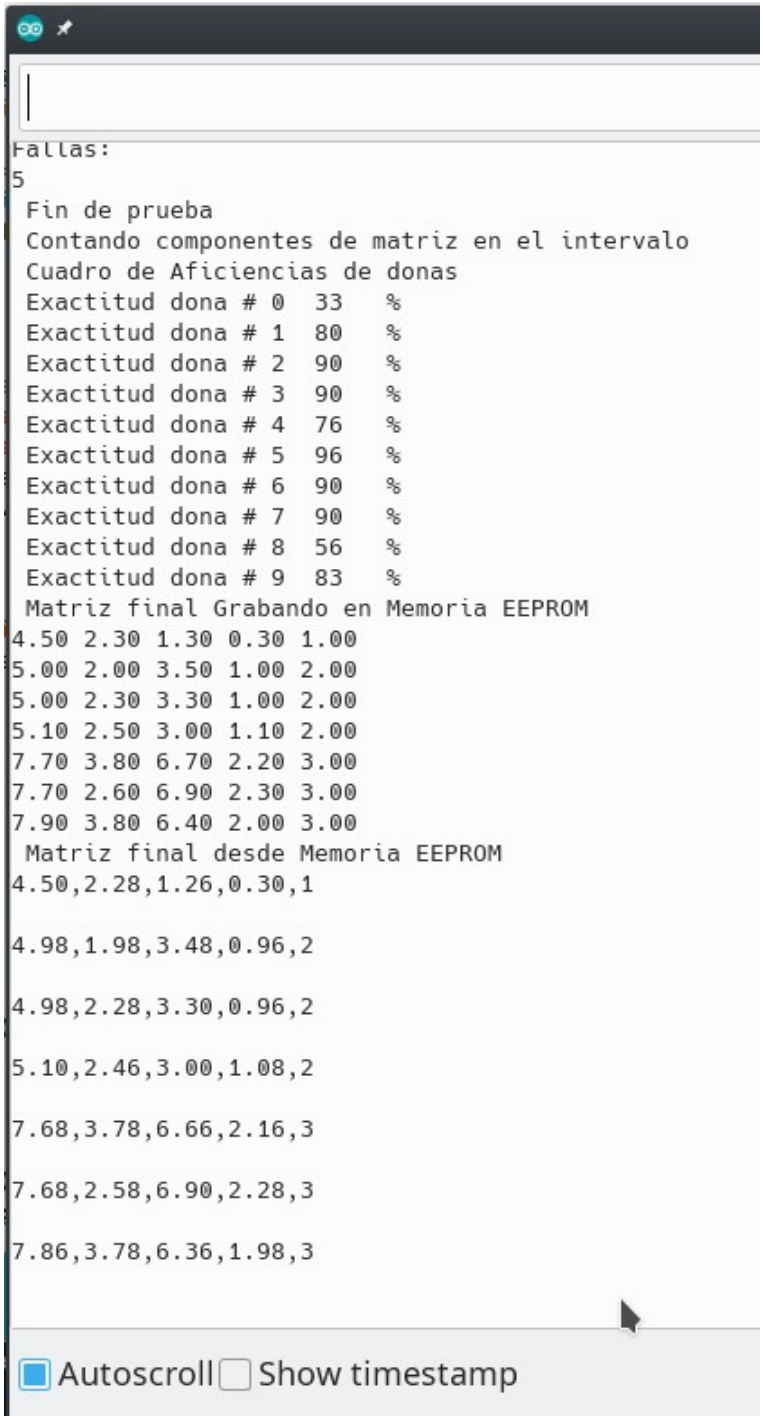
```
int efectividad(){

      for(int i=0;i<=29;i++){
            result=knn(80,5,3,3,test[i]);
             //Serial.println(result);
             Serial.println(i);
               if(result==test[i][4]){
                aciertos++;Serial.println("OK");}else{fallos++;Serial.println("Fail");}
            }
        Serial.println("Porcentage Aciertos:");
      Serial.println((aciertos*100)/30);
        Serial.println("Porcentage Fallas:");
    Serial.println((fallos*100)/30);
Serial.println("Aciertos:");
Serial.println(aciertos);
Serial.println("Fallas:");
Serial.println(fallos);
Serial.println(" Fin de prueba" );
Serial.println(" Contando componentes de matriz en el intervalo");
  return (fallos*100)/30;
}
```

Al final se obtiene la siguiente matriz la cual en este caso va a ser la de mejor resultado en las donas, la cual en este caso esta compuesta de [9][5].

Figura 19: Resultado de Matriz Final

Finalmente tenemos nuestra tabla de eficiencia la cual muestra el porcentaje de eficiencia de cada dona y selecciona la mejor para guardarla mostrarla y generar de esta forma la mejor matriz de comparacion.

Figura 20: Tabla de Resultados en Eficiencia



5. Conclusiones y Recomendaciones

5.1. Conclusiones

- El método de donas nos ayuda a poder excluir datos que no son necesarios eso sí se debe escoger la que mejor eficiencia presente al momento de comparar valores de prueba. El algoritmo de KNN es de mucha importancia para la realización de este proyecto ya que de esto depende el algoritmo de CNN para poder clasificar qué métodos van a la matriz S o la matriz D.
- El algoritmo que se ha desarrollado con cnn y knn es modular ya que puede ser ampliado a cuantas variables o tipos de datos a clasificar.
- Mientras más variables el algoritmo tiene que analizar muchos más recursos utilizara para ju ejecución lo que podría limitar el desarrollo y normal operación del mismo en grandes cantidades de información.

5.2. Recomendaciones

- Comprobar antes de realizar demasiado código que lo que estamos realizando funcione de manera correcta para que en caso de que haya algún error sea fácil de solucionar.
- Escoger los mejores datos para que al momento de comprobar los datos de prueba se obtenga una buena eficiencia es decir muchos más aciertos que errores.
- Limitar el numero de variables a utilizar debido a la poca capacidad de procesamiento que nos brinda nuestra plataforma arduino.
- Comentar el código para de esta manera sea mucho mas fácil su modificación y adaptación.