Anexo 1

Instalación del compilador AVR-GCC

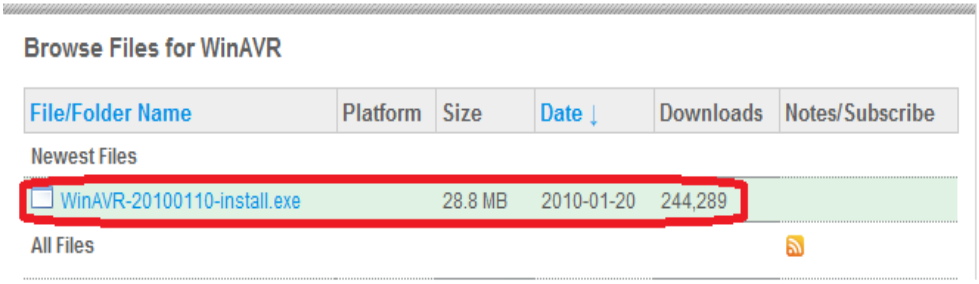
El WinAVR es un conjunto de ejecutables que permiten trabajar con los microcontroladores RISC de la serie AVR de la marca Atmel e incluye el compilador GNU GCC para lenguaje C y C++.

Es indispensable que el WinAVR se instale antes que el AVRStudio o VMLAB para que estos programas reconozcan al compilador y puedan ejecutarlo correctamente. Si no se sigue el orden establecido no se podrá compilar ningún programa. Se recomienda usar el sistema operativo Windows XP.

Los pasos para descargar el WinAVR son:

1. Entrar a la página web de WinAVR: http://winavr.sourceforge.net/
2. Hacer click en la opción Download que se encuentra en la parte superior izquierda de la página principal.

3. Seguir la indicación que aparece y hacer click en: SourceForge.net WinAVR download page.
4. En la nueva página se debe hacer click en el ejecutable más reciente que en este caso es: WinAVR-20100110-install.exe

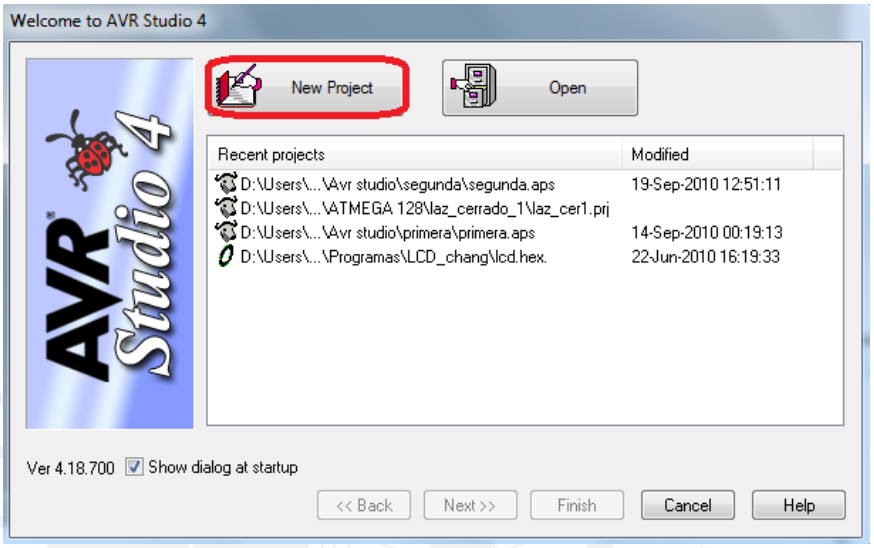


5. Finalmente, se elige la ubicación del disco duro donde se va a guardar el compilador y se hace click en Aceptar para que comience la descarga.

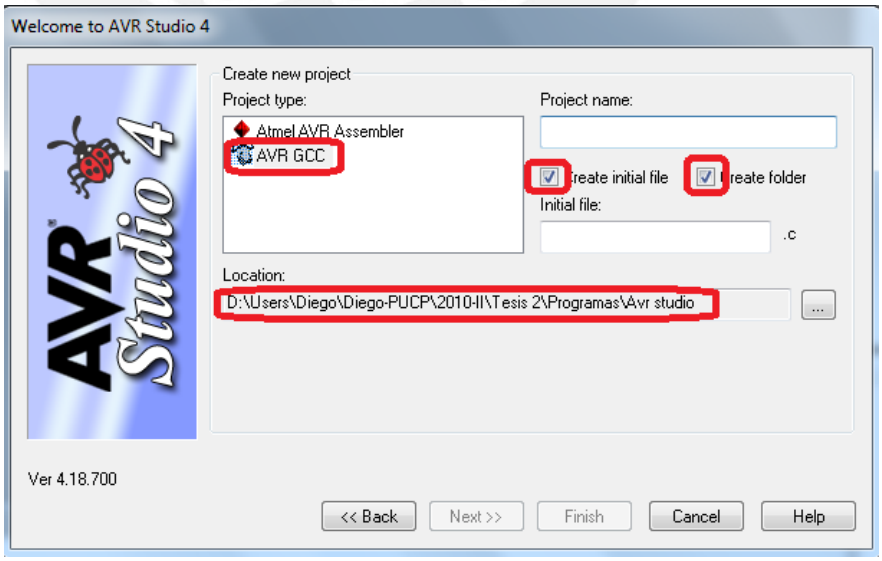
Anexo 2

Pasos para compilar un código en lenguaje C usando el AVRStudio

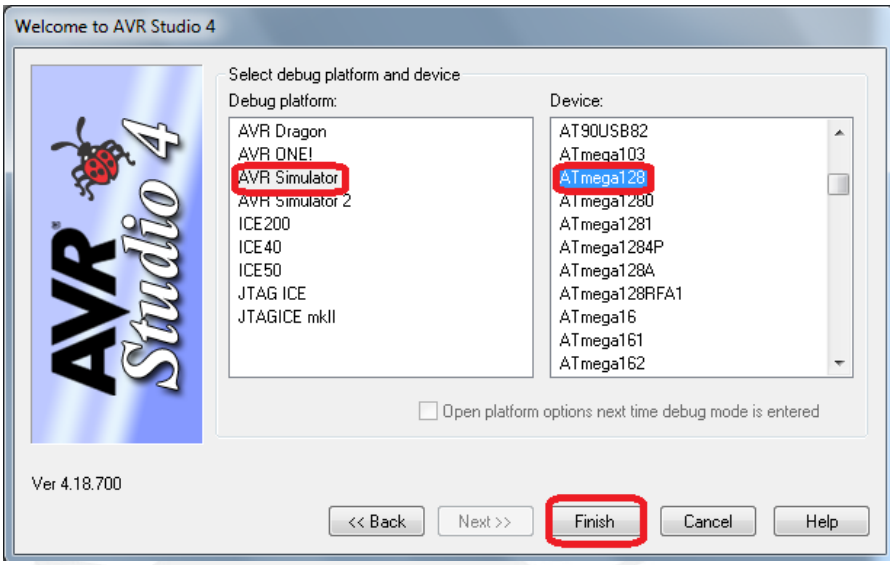
1. Ir a la barra de Inicio → Todos los programas → Atmel AVR Tools → AVRStudio 4 y aparecerá la siguiente ventana en la cual se hace click en New Project.



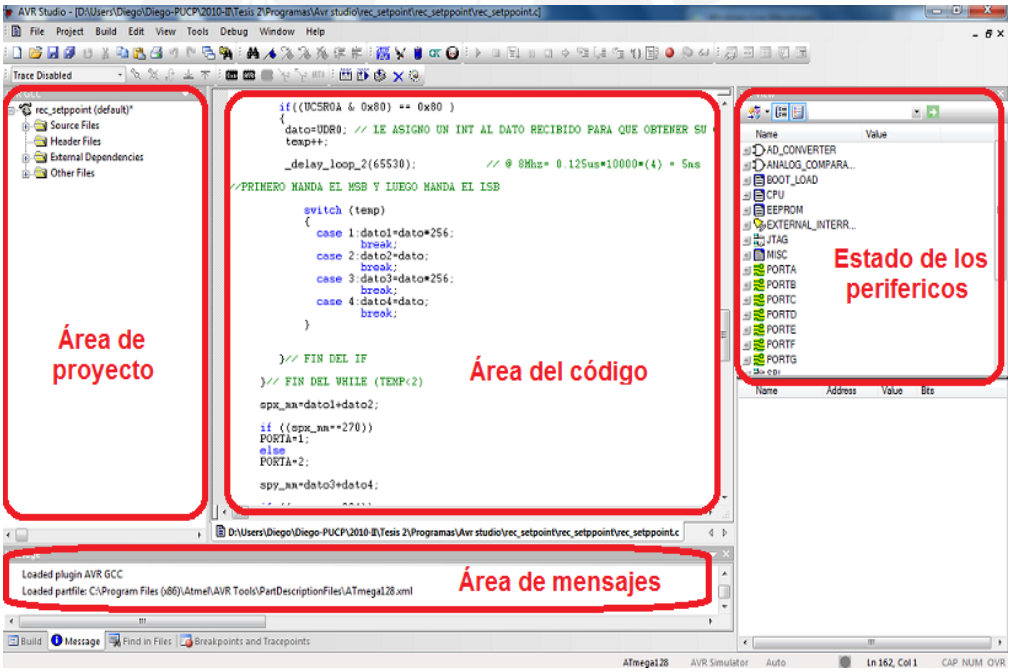
2. Luego, en la opción Project Type se elige AVR GCC y en Location se indica la ubicación del disco duro donde se quiere guardar la carpeta con los archivos generados por el programa. Después, se introduce el nombre del programa en el recuadro Project name, verificando que las opciones Create initial file y Create folder estén marcadas con un check.



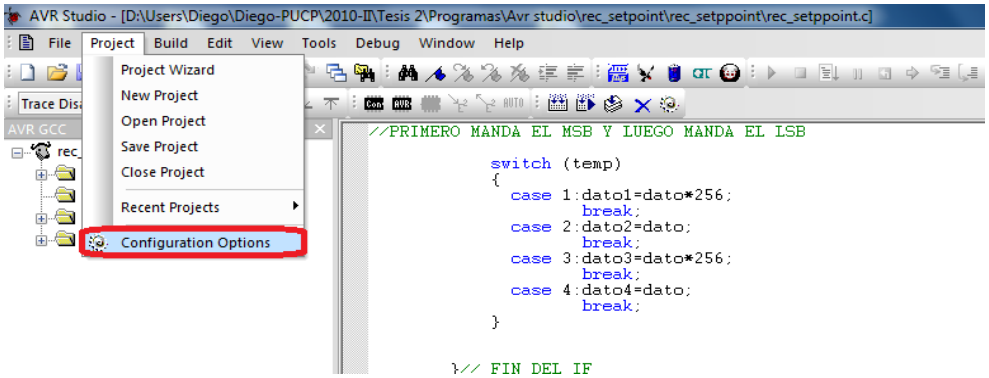
3. En la siguiente ventana se elige AVR Simulator en la opción Debug Platform y en la lista Device se selecciona el microcontrolador a usar, que en este caso es el Atmega128. Por último, se hace click en Finish



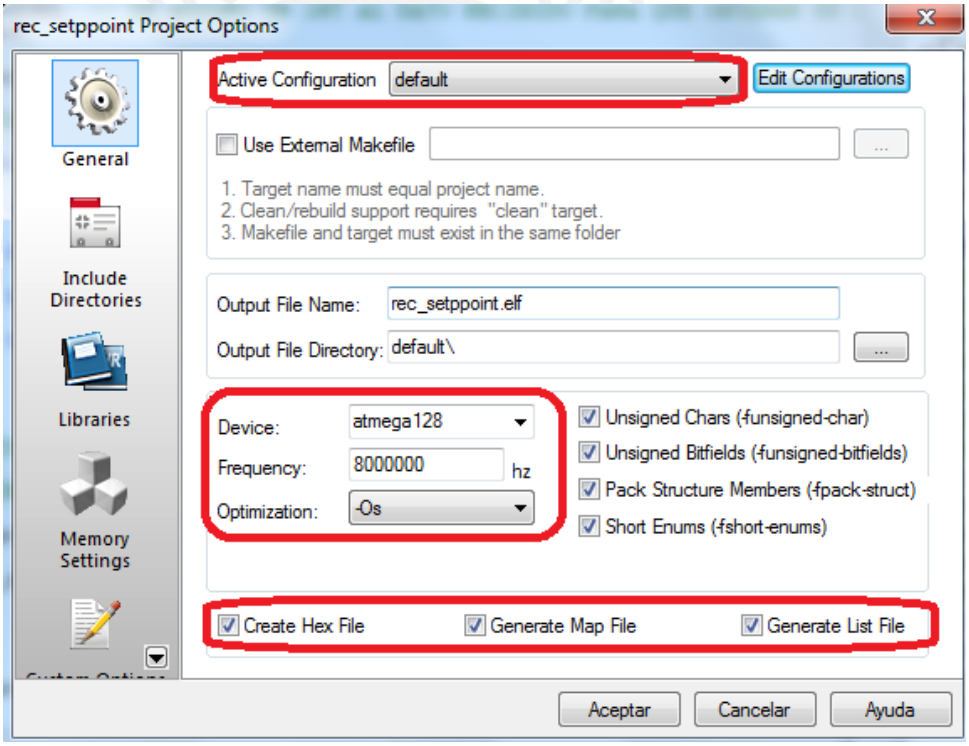
4. Entonces, aparecerá la ventana de trabajo que tiene la siguiente apariencia:



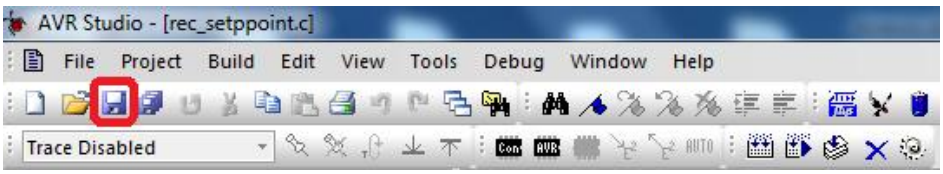
5. Antes, de compilar cualquier programa se debe entrar al Menú -> Project -> Configuration Options:



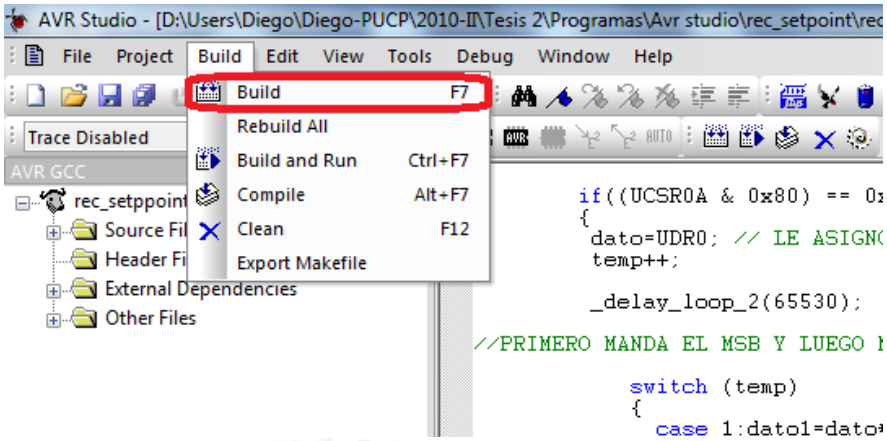
En la opción General se elige el microcontrolador, la frecuencia de trabajo y el tipo de Optimización recomendado que es –Os. Después de seleccionar las 7 opciones de la parte inferior hacer click en Aceptar.



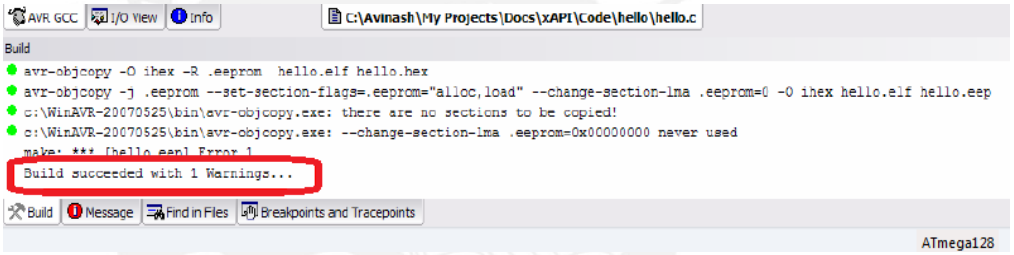
6. Luego de haber escrito el programa en lenguaje C en el Área del código, se hace click en el disquete para guardar el programa y poder compilarlo más adelante.



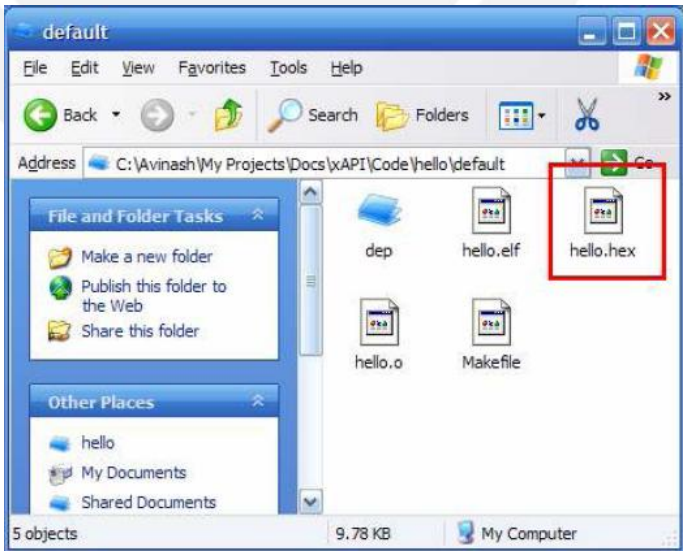
7. El último paso es ir al Menú -> Build -> Build para compilar el programa y generar el archivo .hex respectivo.



En la ventana de mensajes aparecerá que el programa fue compilado correctamente o en caso contrario, se indicará el número de línea del código donde se produjo el error.



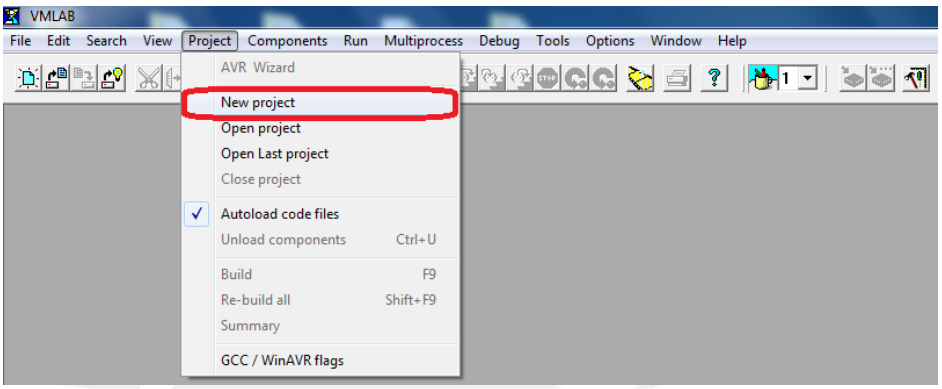
Finalmente, el archivo .hex se guarda en la carpeta default que se encuentra dentro de la carpeta creada al inicio.



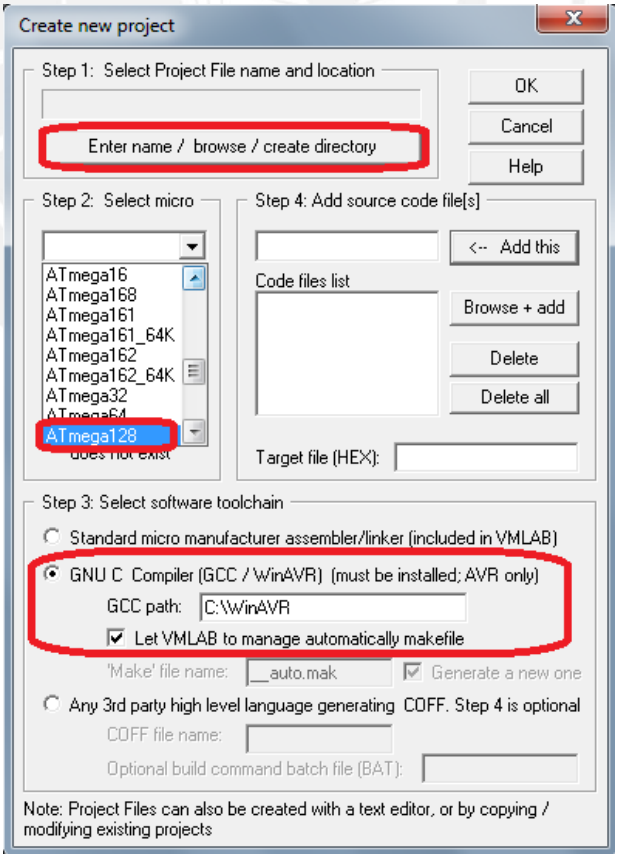
Anexo 3

Pasos para compilar y simular código en lenguaje C usando el entorno integrado de desarrollo VMLAB

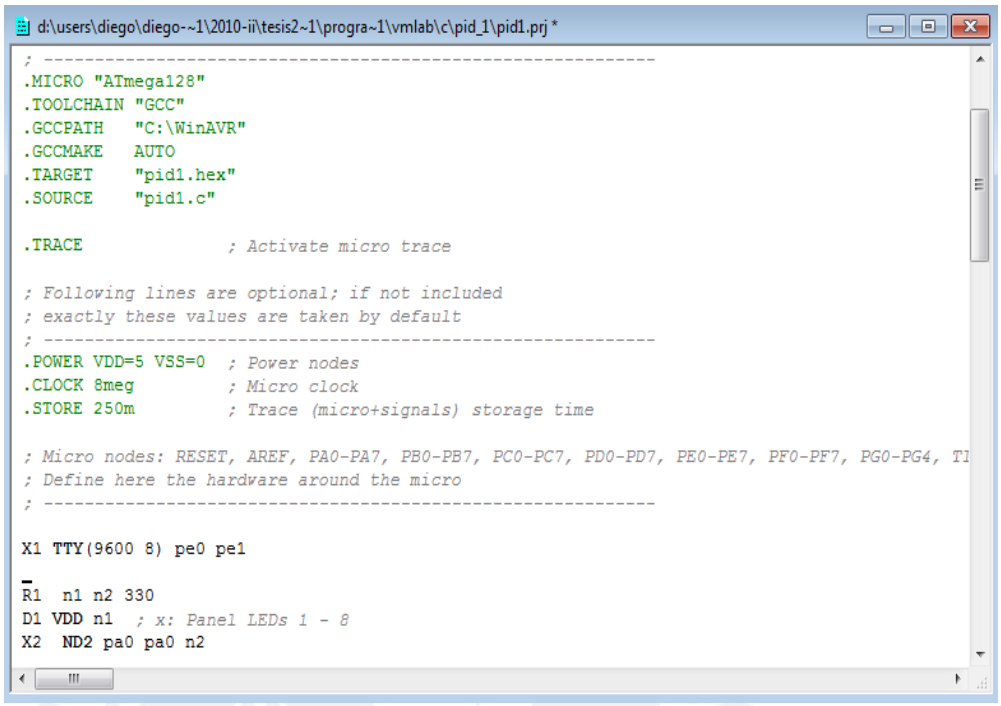
1. En la pantalla incial del VMLAB, hacer click en New project para crear un nuevo proyecto donde se va a guardar el programa.



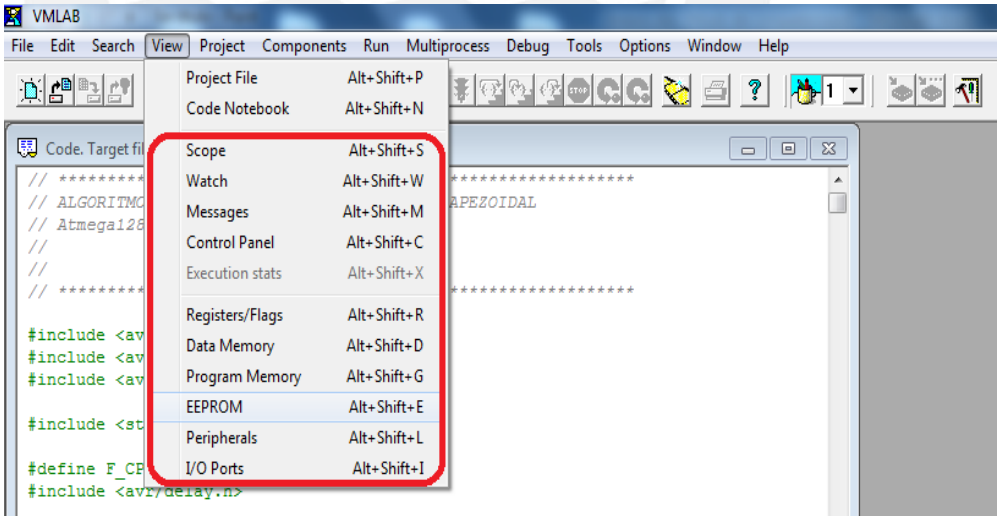
2. En la opción Step 1 se ubica la carpeta donde se va a guardar el programa y se le asigna su respectivo nombre. Después, en Step 2 se selecciona el microcontrolador que se está utilizando para luego marcar la opción GNU C COMPILER. Finalmente, se hace click en Add this y OK.



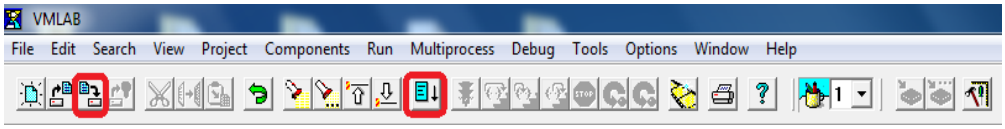
3. Para simular los puertos de entrada-salida, puerto serial, señales PWM, etc. se introduce la sintaxis correspondiente en el archivo .prj.



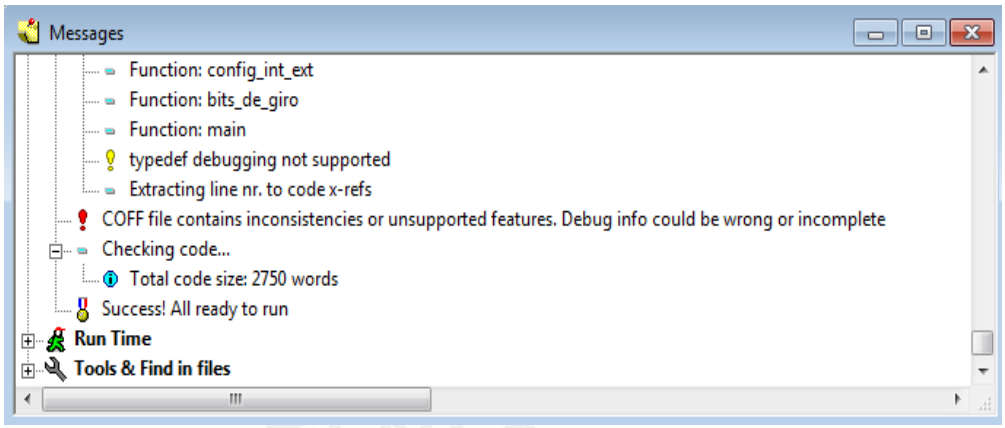
4. Además, en el menú View se encuentran todos módulos disponibles para la simulación.



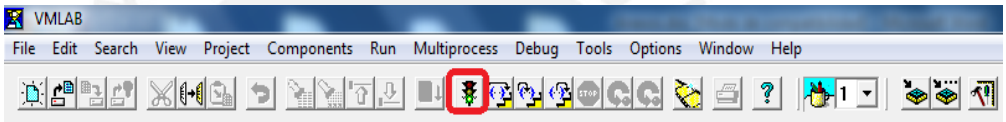
5. Luego de tener el archivo .hex y el .prj terminado, se hace click en Save y luego en Build para proceder a compilar el código.



6. Si el código no tiene errores, al final de la ventana de mensajes aparecerá *Success! All ready to run*, en caso contrario saldrá un mensaje de error que significa que existen errores de sintaxis.

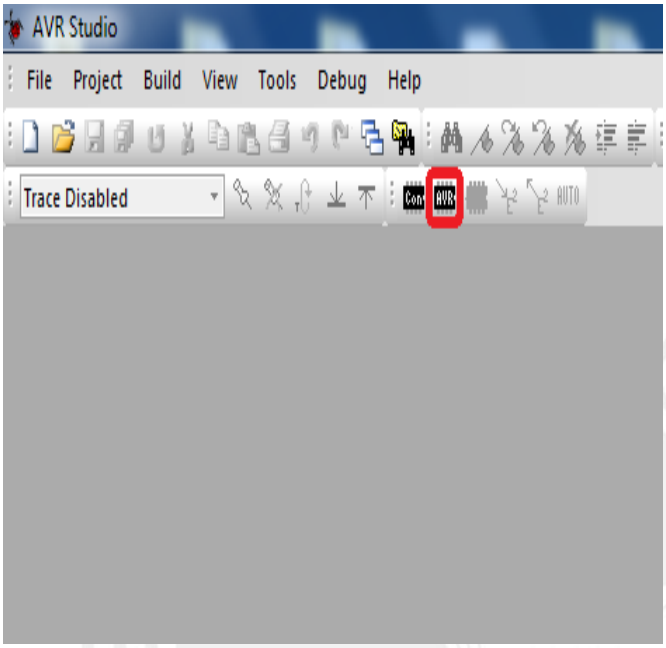


7. Por último, se hace click en el icono del semáforo (Run) para comenzar la simulación y para detenerla se hace click en la flecha azul de Restart (deep)

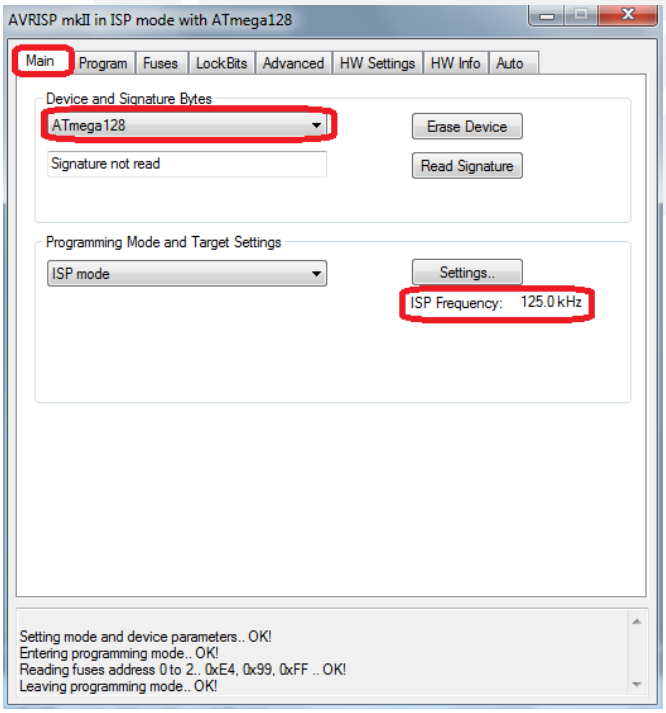


Anexo 4

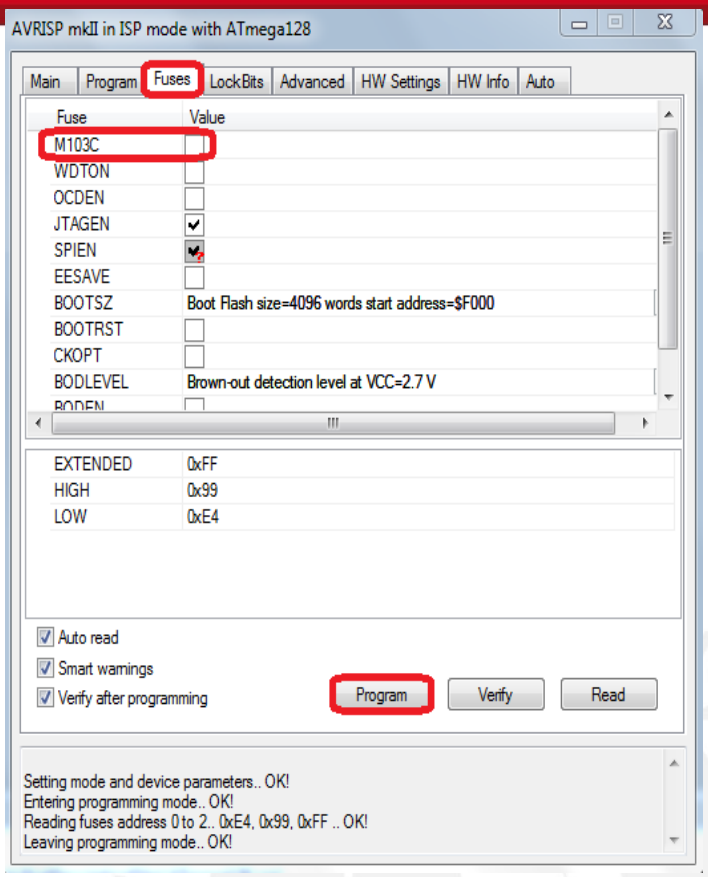
Guía para programar el Atmega128 vía ISP usando el AVRStudio y el programador AVRISPmkII



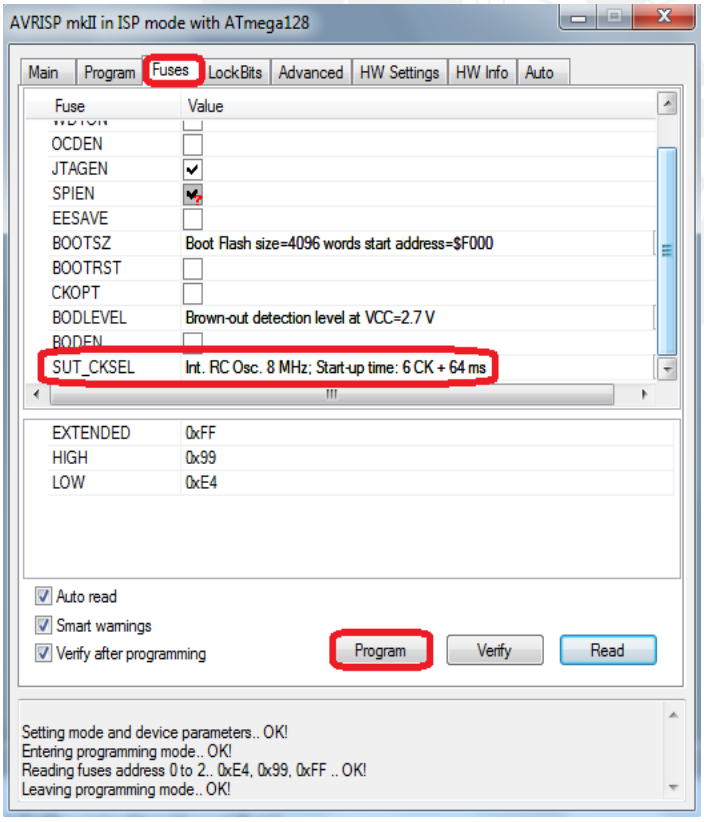
1. En primer lugar, se hace click en la opción Select the connected AVR programmer.



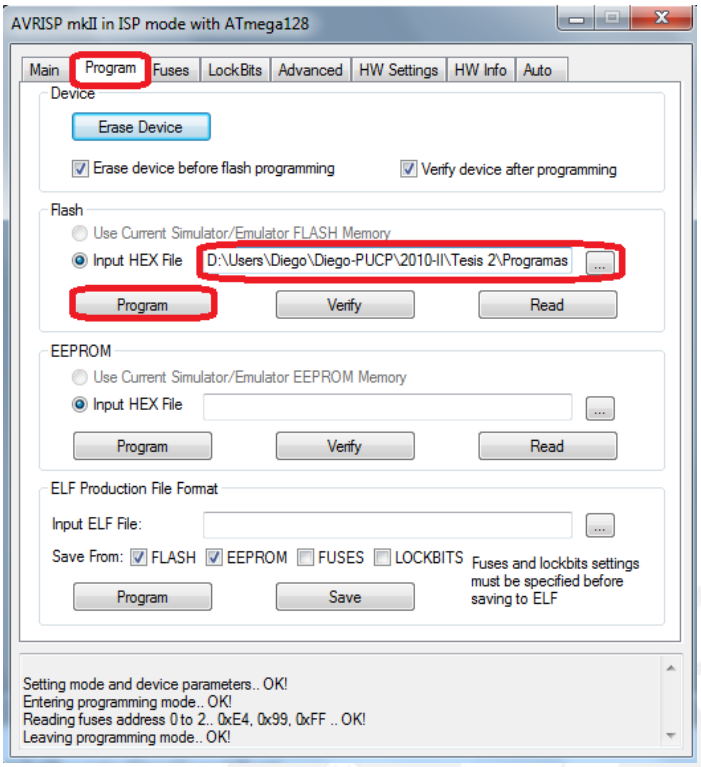
2. En la pestaña Main se elige el microcontrolador a programar en la opción Device and Signature Bytes. La opción Programming Mode and Target Settings por defecto estará en ISP mode y en Settings se elige la frecuencia de programación que debe ser como máximo la cuarta parte de la frecuencia de trabajo del microcontrolador.



3. En la pestaña Fuses se debe tener mucho cuidado con el fusible **M103C**, el cual viene activado por defecto haciendo que el Atmega128 adopte todas las características del Atmega103 y no funcione como debería hacerlo. Por lo tanto, se debe desactivar este fusible y hacer click en Program para concretar el cambio realizado.



4. En la misma pestaña, Fuses, se debe seleccionar la frecuencia de trabajo del microcontrolador entre las distintas opciones que se presentan en la lista despegable del fusible SUT_CKSEL y hacer nuevamente click en Program para que realice el cambio respectivo.



5. Por último, en la opción Flash de la pestaña Program se selecciona el archivo .hex generado por el compilador y se hace click en Program para comenzar la descarga del programa en el microcontrolador.

Anexo 5

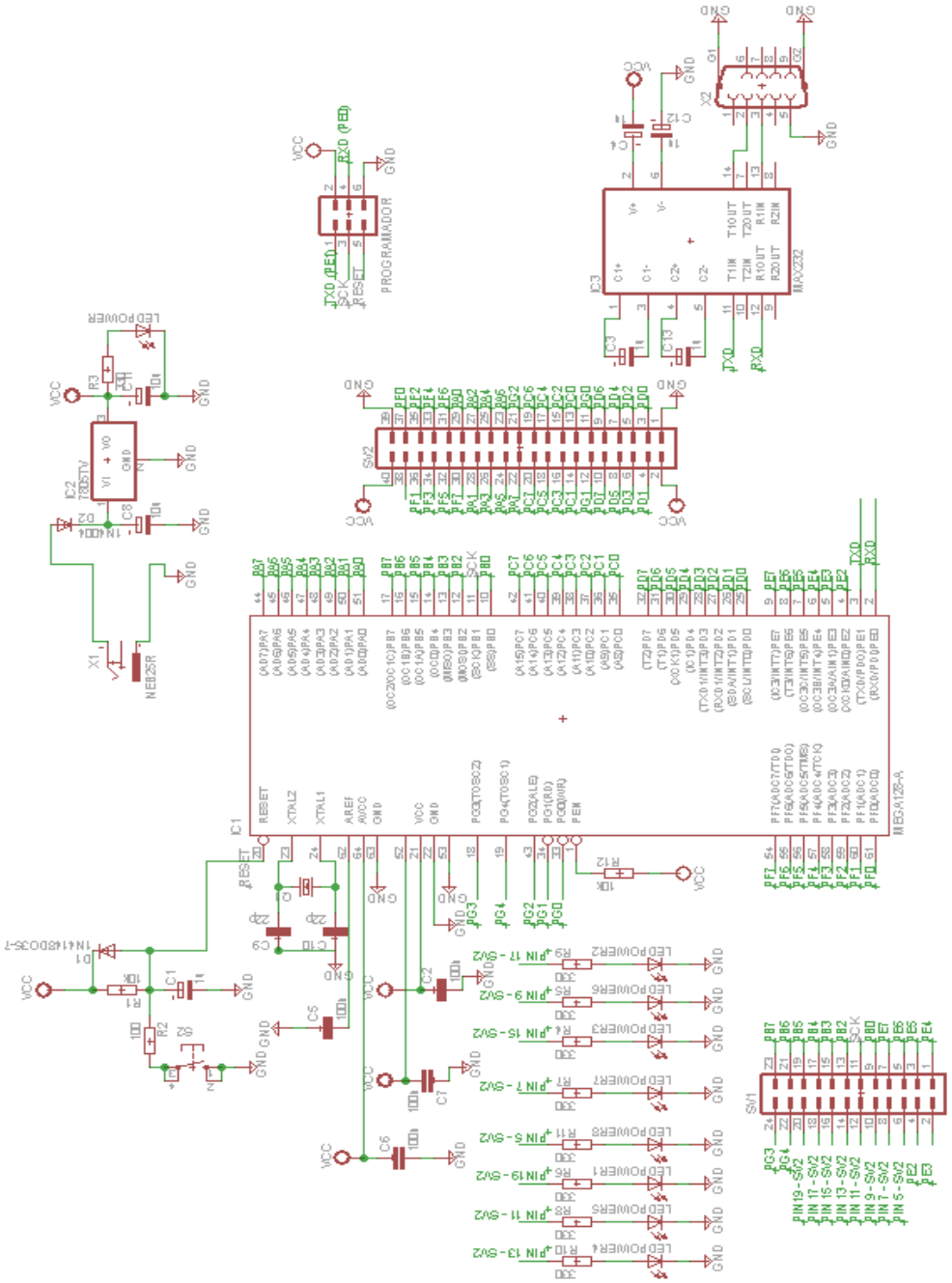
Pasos a seguir para usar la interfaz gráfica

1. Asegurarse de que los parámetros de comunicación serial sean: 9600bps, 1 bit de stop y el nombre del puerto serial correspondiente.
2. Verificar que el led de la parte superior derecha esté de color verde. En caso se encuentre de color rojo, se debe hacer click para que cambie de color.
3. Hacer click en el botón Run  de la barra de herramientas para poder ejecutar el programa.
4. Ingresar la coordenada X en el recuadro de la izquierda.
5. Presionar el botón que se encuentra a la derecha del recuadro para mandar la posición respectiva al microcontrolador.
6. Ingresar la coordenada Y en el recuadro de la derecha.
7. Presionar el botón que se encuentra a la derecha para enviar la posición correspondiente.
8. Para mandar nuevos set points se realiza el mismo procedimiento. Pero, previamente se debe aumentar en una unidad el índice del lado izquierdo de cada recuadro, el cual representa el número de ingresos realizados por el usuario.
9. Cuando se quiera detener la ejecución, se debe hacer click en el led de color verde para que cambie al color rojo indicando que el programa ha sido detenido.

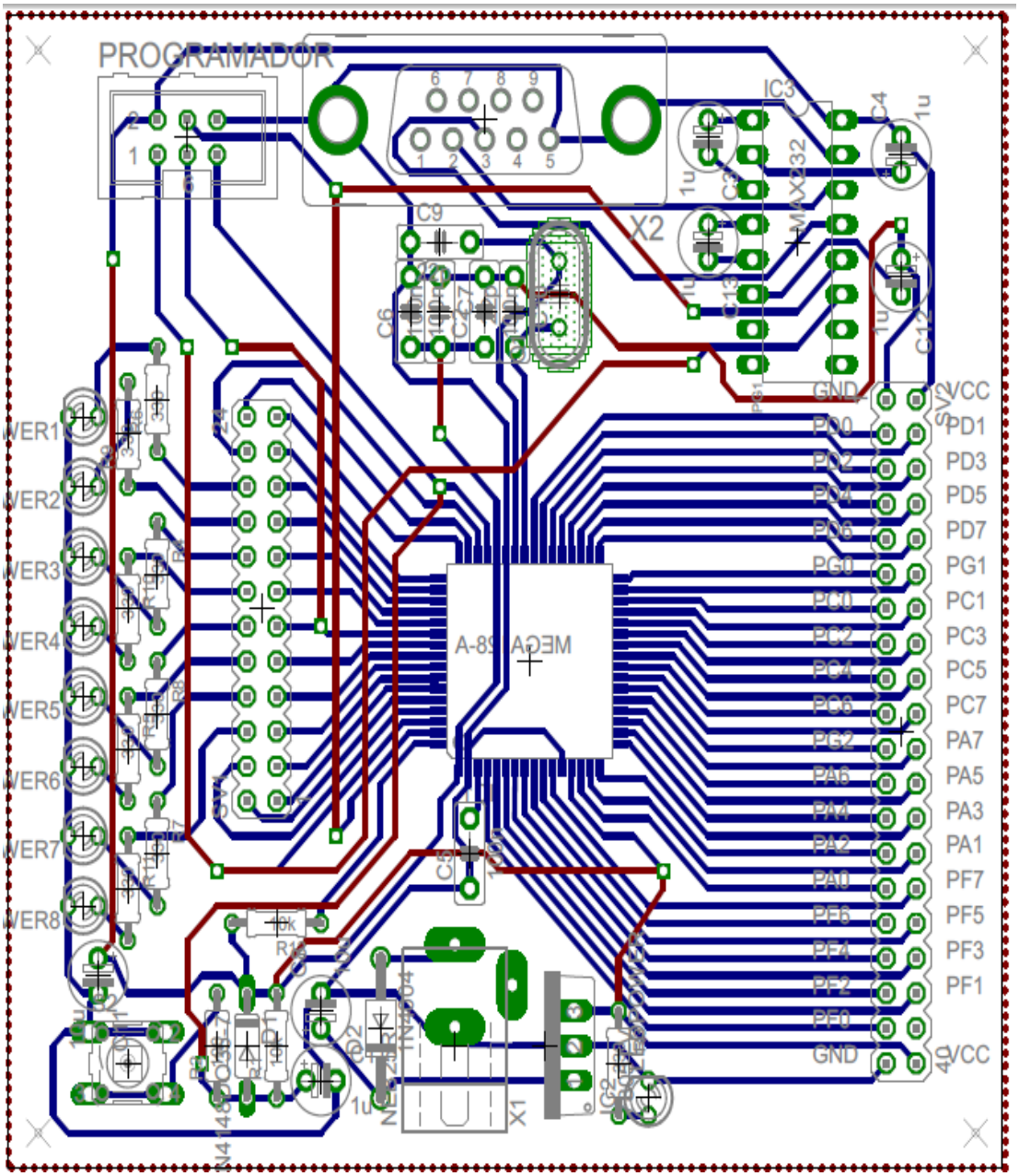
Anexo 6

Diagrama esquemático y board de la tarjeta del controlador (Atmega128)

Diagrama esquemático



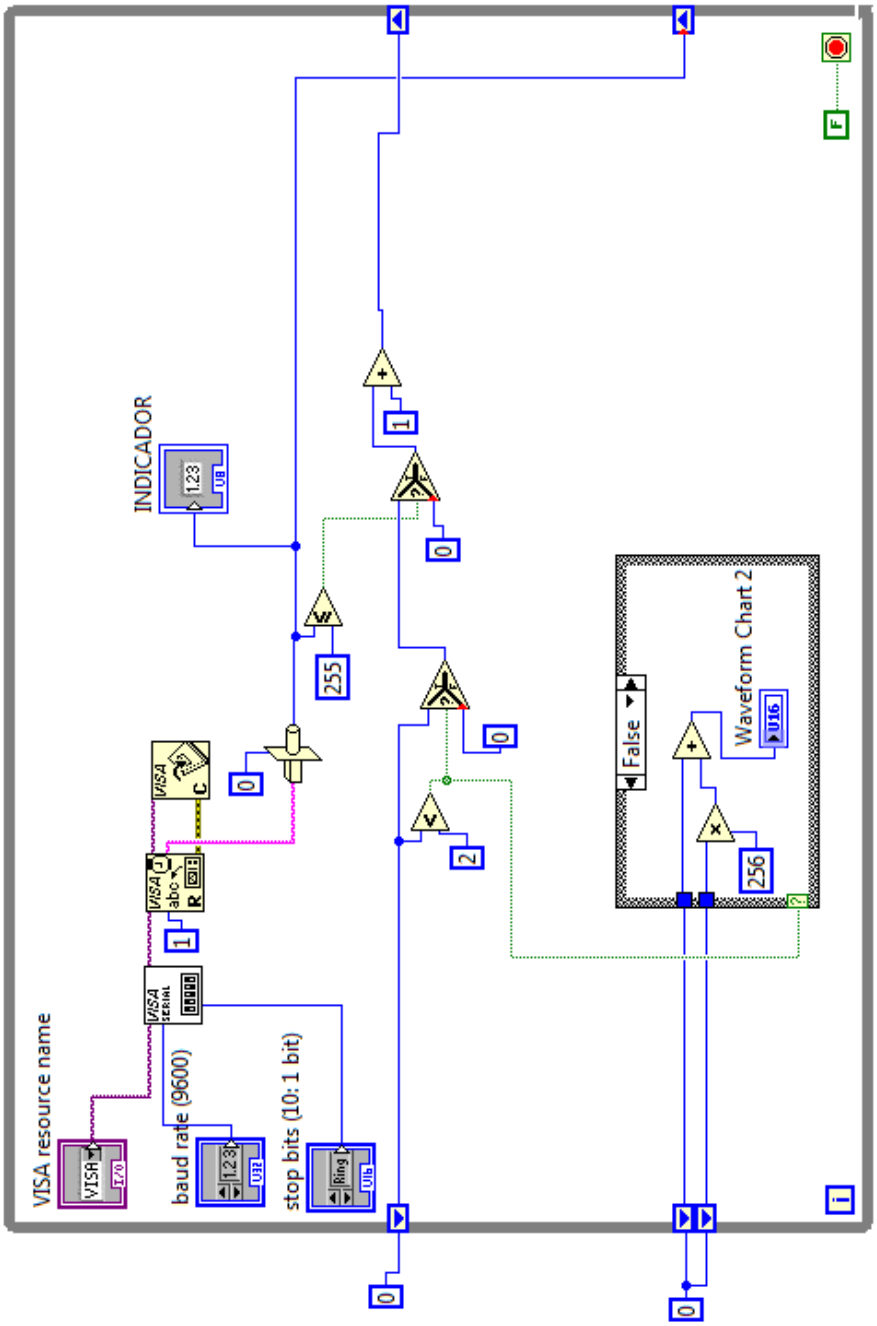
Board



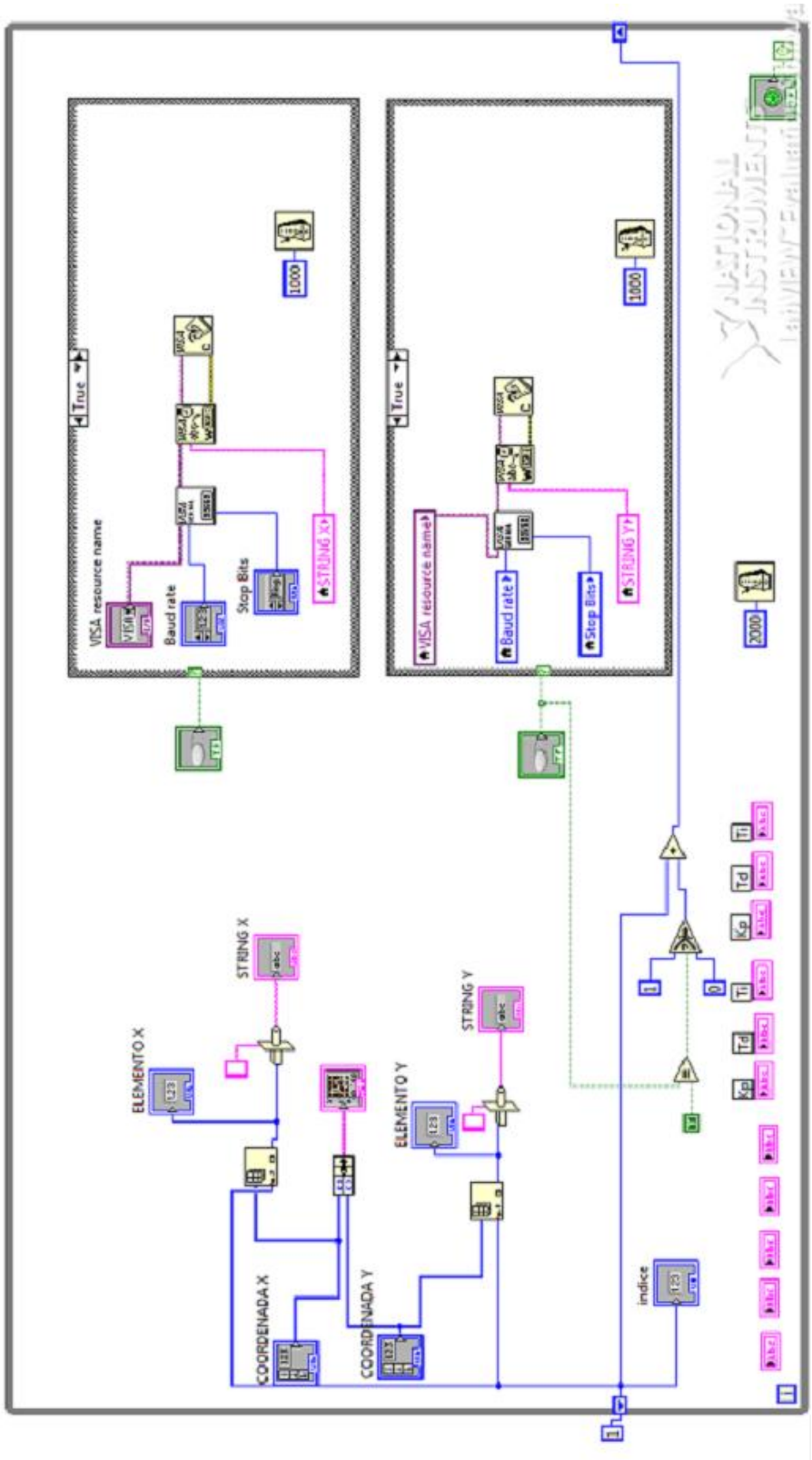
Anexo 7

Programas

Programas de la interfaz para pruebas de lazo abierto (Labview 8.5)



Programa de la interfaz grafica de la mesa XY (Labview 8.5)



Generador de trayectoria

En este caso, se debe adaptar el código de la trayectoria escrito en lenguaje C al lenguaje M que es el reconocido por el software científico Matlab.

El programa que se ejecutó en Matlab es el que se muestra a continuación:

```
%PERFIL DE VELOCIDAD TRAPEZOIDAL

%FUNCION QUE RECIBE LOS VALORES DE ENTRADA:
% spx_ini: Punto de partida del eje X
% spy_ini: Punto de partida del eje Y
% spx_fin: Punto de llegada del eje X
% spy_fin: Punto de llegada del eje Y
% acel: aceleración definida por el usuario para el movimiento.

% SALIDAS:
% arr_sp1: Vector con la trayectoria del eje que tiene mayor recorrido
% arr_sp2: Vector con la trayectoria del eje que tiene menor recorrido

% COORDENADAS DEL EJE "X"
spx_ini=0     %en milimetros
spx_fin=180   %en milimetros

% COORDENADAS DEL EJE "Y"
spy_ini=0     %en milimetros
spy_fin=120    %en milímetros
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
vel_max=150      %en mm/s

acel_max=1200 %en mm/s^2

acel=250          %aceleración definida por el usuario

%EL OBJETIVO ES QUE AMBOS EJES TERMINEN SU MOVIMIENTO EN EL
MISMO INSTANTE DE TIEMPO. POR ESA RAZON, SE MANTIENE EL TIEMPO
DEL EJE QUE DEMORA MAYOR TIEMPO Y SE RALENTIZA EL MOVIMIENTO
DEL EJE QUE TARDA MENOS TIEMPO EN LLEGAR A SU POSICION FINAL

%ASI, SE CONSIGUE QUE AMBOS EJES COMIENCEN Y TERMINEN SU
MOVIMIENTO AL MISMO TIEMPO


%SE CALCULA EL TIEMPO QUE TARDA CADA EJE EN LLEGAR A SU PUNTO
FINAL.

%ENTONCES, SE SABE CUAL EJE TARDA MAYOR TIEMPO EN LLEGAR A SU
PUNTO FINAL

tfx=((vel_max*vel_max) - acel*(spx_ini - spx_fin))/(acel*vel_max)

tfy=((vel_max*vel_max) - acel*(spy_ini - spy_fin))/(acel*vel_max)


if (tfx>=tfy)       % SI EL EJE “X” DEMORA MAYOR TIEMPO QUE EL EJE “Y” EN
  sp_ini=spx_ini % LLEGAR AL FINAL DE SU RECORRIDO, SE CONSIDERAN
  sp_fin=spx_fin %LOS PARÁMETROS DEL EJE “X” PARA CALCULAR EL
  tf=tfx             %TIEMPO CRITICO. ESTO QUIERE DECIR PUNTO  INICIAL,
else               % PUNTO FINAL Y TIEMPO FINAL.
  sp_ini=spy_ini    % EN CASO CONTRARIO, SE CONSIDERAN LOS
  sp_fin=spy_fin  % PARAMETROS DEL EJE “Y” PARA CALCULAR EL TIEMPO
  tf=tfy             % CRITICO
end
```

```
%TIEMPO CRÍTICO CALCULADO A PARTIR DEL EJE CON MAYOR TIEMPO FINAL

tc= (sp_ini - sp_fin + vel_max*tf) / (vel_max)

N_total = tf/0.005 % CANTIDAD DE PUNTOS EN TODA LA TRAYECTORIA


if (tc<tf/2)  % SE VERIFICA QUE SE CUMPLA LA CONDICION DE
              % ACELERACION MINIMA


    N=N_total/3 % CANTIDAD DE PUNTOS POR TRAMOS (3 tramos)
    N=floor(N)


     % TRAMO DE ACELERACION
     t1=0: tc/(N-1) : tc
     x1= sp_ini + (0.5*acel*t1.^2)
     v1= acel*t1


    % TRAMO DE VELOCIDAD CONSTANTE
     t2=tc: (tf-2*tc)/(N-1) : tf-tc
     x2= [x1 (sp_ini)+(acel*tc*(t2-tc/2))]
     v2= [v1 acel*tc*ones(1,N)]


     % TRAMO DE DESACELERACION
     t3=tf-tc: tc/(N-1) : tf
     x3=[x2 (sp_fin)-(0.5*acel*((tf*ones(1,N)-t3).^2))]
     v3=[v2 acel*(tf*ones(1,N)-t3)]
```

```
else
    v3=0
end

%SE ALMACENA EL VECTOR CON LA TRAYECTORIA DEL EJE DE MAYOR TIEMPO FINAL EN EL VECTOR arr_sp1
arr_sp1=x3

%SE CALCULA LA PENDIENTE
pendiente=(spy_fin-spy_ini)/(spx_fin-spx_ini)

%SE CALCULAN LAS POSICIONES DEL EJE DE MENOR TIEMPO FINAL (arr_sp2)
if (tfx>=tfy)
    arr_sp2=spy_ini*(ones(1,3*floor(N)))+ pendiente*(arr_sp1-spx_ini)
else
    arr_sp2=spx_ini*(ones(1,3*floor(N)))+ (1/pendiente)*(arr_sp1-spy_ini)
end

tiempo= [t1 t2 t3] % SE CONCATENA EL EJE DEL TIEMPO

%SE CALCULA EL PERFIL DE VELOCIDAD DEL EJE RESTANTE

%TRAMO DE ACELERACION
for i=1:(N-1)
v5(i)=(arr_sp2(i+1)-arr_sp2(i))/(tiempo(i+1)-tiempo(i));
end
```

```
%TRAMO DE VELOCIDAD CONSTANTE

for i=1:(N-1)

v6(i)=(arr_sp2(N+i+1)-arr_sp2(N+i))/(tiempo(i+1)-tiempo(i));

end


%TRAMO DE DESACELERACION

for i=1:(N-1)

v7(i)=(arr_sp2(2*N+i+1)-arr_sp2(2*N+i))/(tiempo(i+1)-tiempo(i));

end


%SE CONCATENA EL VECTOR DE VELOCIDAD DEL EJE DE MENOR TIEMPO FINAL

v8= [ 0 v5 v6(1) v6 v7(1) v7 0 ]


% SE GRAFICAN LAS POSICIONES DE REFERENCIA Y EL PERFIL DE VELOCIDAD DEL EJE "X"

figure

plot(tiempo,arr_sp1,tiempo,v3)

title('POSICION Y PERFIL DE VELOCIDAD DEL EJE X')

xlabel ('Tiempo (seg)')

ylabel ('Posicion (mm), Velocidad (mm/seg)')

% SE GRAFICAN LAS POSICIONES DE REFERENCIA Y EL PERFIL DE VELOCIDAD DEL EJE "Y"

figure

plot(tiempo,arr_sp2,tiempo,v8)

title('POSICION Y PERFIL DE VELOCIDAD DEL EJE Y')

xlabel ('Tiempo (seg)')

ylabel ('Posicion (mm), Velocidad (mm/seg)')
```

```
% SE GRAFICAN LOS PERFILES DE VELOCIDAD DEL EJE "X" e "Y"
figure
plot(tiempo,v3,tiempo,v8)
title('PERFIL DE VELOCIDAD DEL EJE X e Y')
xlabel ('Tiempo (seg)')
ylabel ('Velocidad (mm/seg)')
%%%%%%%%%%%%%%%%          FIN          %%%%%%%%%%%%%%%%
```

Programa para pruebas a lazo abierto

```
// ********************************************************
// Este programa genera una señal PWM de 32KHz (Timer 1 - OC1A – Fast PWM)
// con una resolución de 0.4% (ICR1 = 247). Este se va a emplear para hacer la
// prueba a lazo abierto y determinar la relación entre la cantidad de pulsos de
// encoder y el desplazamiento lineal de cada eje.

// SALIDAS:
// PB5: Señal PWM
// PB0: Bit de giro

// ENTRADAS:
// PE4: INT4  (interrupción por pulsos de encoder)
// PE5: INT5  (interrupción por detección de sobrecorriente en el motor)

#include <avr\io.h>   // Declaración de librerías
#include <avr\interrupt.h>
#include <avr\signal.h>
#include <stdint.h>

#include <avr/delay.h>
#define F_CPU 8000000UL // Frecuencia del oscilador interno: 8 MHz

uint16_t palabra, pulsos=0;  // Declaración de variables globales
uint8_t pulso1, pulso2, cont=0, pwm;
```

```
// Sub rutina de interrupción por pulsos de encoder
signal(sig_interrupt4) {
pulsos++;
}

// Sub rutina de interrupción por sobrecorriente
signal(sig_interrupt5) {
TCCR1A=0x00; // Se detiene la generación de la señal PWM
TCCR1B=0x00;
}

// Sub rutina de configuración de puertos de entrada/salida
void config_puertos(void) {
 DDRC=255;
 PORTC=0b00000000;
 DDRB=0b01100001; // PB5=OC1A, PB6=OC1B
 PORTB=PORTB | 0b00000000;  // bit de giro (PB0)
}
// Sub rutina de configuración del puerto serial
void config_serial(void) {
 UBRR0H = 0x00;
 UBRR0L = 0x19;  // velocidad de transmisión: 38400bps
 UCSR0A = 0x02;  //inicialización del registro de control
 UCSR0C = 0x86;  //habitación de transmisión
 UCSR0B = 0x18;  //comunicación asíncrona, sin paridad, 1 bit de parada, 8 bits de
                 // datos
}
```

```
// Sub rutina de configuración de las interrupciones externas
void config_int4_int5(void) {
EICRB = EICRB | 0b00001111; // INT4 e INT5 configuradas por detección de flanco
                            // de subida
EIMSK = EIMSK | 0b00110000;  // Habilitación de INT4 e INT5
}

// Sub rutina de configuración del canal PWM
void config_pwm(void) {
TCCR1A=0b10000010; // Sin pre escalamiento, COM1A1 1 y COM1A0 = 0
TCCR1B=0b00011001; // Modo 14: Fast PWM
OCR1A =1;
ICR1=247;
}
// Sub rutina de envío de datos por el puerto serial del microcontrolador hacia el
// Labview
void envio_serial(void) {
        pulso1 = pulsos & 0x00FF; // LSB
        palabra = pulsos >> 8;
        pulso2 = palabra & 0x00FF; // MSB
        if((UCSR0A & 0x20) == 0x20 )
        { UDR0=pulso1;  //Primero se envía el LSB
        }
        _delay_loop_2(20000); // Retardo
        if((UCSR0A & 0x20) == 0x20 )
        { UDR0=pulso2;  //Por último se envía el MSB
        }
}
```

```
// Programa principal
int main(void){

 cli(); // Desabilitación de interrupciones

// Se invocan a las sub rutinas de configuración
 config_puertos();
 config_serial();
 config_int4_int5();
 config_pwm();

 sei();// Habilitación de interrupciones

 while(1) {
    while(pulsos==0) {  //Mientras no se reciba pulsos provenientes del encoder o no
                        // se desplace el eje, se va aumentando en 0.4% el ciclo de
                        // de trabajo de la señal PWM
      _delay_loop_2(65530); // Retardo de 32ms
      _delay_loop_2(65530);  // Retardo de 32ms
      OCR1A++;
        if (OCR1A>=247)
          OCR1A=0;
        else
          ;
     } //Fin del While pulsos== 0
```

```
while(pulsos<1300) { // Condición de parada. Esta sirve para detener el motor
                     // y calcular la relación que existe entre el desplazamiento
                     // lineal y la cantidad de pulsos enviados por el encoder.

   envio_serial(); // Se invoca a la sub rutina de envío serial de datos al Labview
   _delay_loop_2(65530);   // Retardo de 32ms

 } // Fin de la condición de parada

 TCCR1A=0x00; // Se detiene la generación de la señal PWM en el puerto OC1A
 TCCR1B=0x00;

}// Fin del While (1)

}// Fin del Programa principal
```

Programa principal

```
#include <avr\io.h>
#include <avr\interrupt.h>
#include <avr\signal.h>
#include <stdint.h>

#define F_CPU 8000000UL

#include <avr/delay.h>
#include <math.h>
#include <stdlib.h>

#define pi 3.14159


//***************************************
// DECLARACION DE ARREGLOS
//***************************************

float arr_sp1[480]; // ARREGLO DECLARADO PARA QUE ALMACENE LOS SET
                    //  POINTS DE UN EJE

float arr_sp2[480]; // ARREGLO DECLARADO PARA QUE ALMACENE LOS SET
                    //  POINTS DEL EJE RESTANTE

//*************************************


uint8_t dato,temp=0;              // VARIABLES UTILIZADAS PARA LA RECEPCION

uint16_t dato1,dato2,dato3,dato4; // SERIAL DE LOS SET POINTS

uint8_t  min=0,max=50;   // LIMITES DEL ANCHO DE PULSO EXPRESADO EN
                         // PORCENTAJE

uint16_t palabra,indice; // VARIABLES UTILIZADAS PARA LA COMUNICACION
                         // SERIAL

uint16_t pulsos_x=0,pulsos_y=0; // VARIABLES QUE ALMACENAN LOS PULSOS
                                // DE ENCODER RECIBIDOS

uint16_t contador=0,cont=0;

uint16_t long1,long2,long3; // VARIABLES QUE ALMACENAN LA CANTIDAD DE
                            // MUESTRAS POR CADA TRAMO
```

```
//*****************************************************
// PARAMETROS DEL PEFIL DE VELOCIDAD
//*****************************************************

float   spx_ini=0,spy_ini=0;  // SE DEFINE AL ORIGEN (0,0) COMO LA
                              //          COORDENADA INICIAL
float   spx_fin,spy_fin;  // VARIABLES QUE ALMACENAN LOS SET POINTS
                          //  RECIBIDOS DESDE LABVIEW

float sp1_ini,sp1_fin;
float sp2_ini,sp2_fin;

float tf,tc,tfx,tfy,pendiente,N,N_total;
float vel_max=150;
float acel=300;

//*************************************************************************************************

//*********************************************
// PARAMETROS DEL ALGORITMO PID
//*********************************************


float posx_rad=0,posy_rad=0;// VARIABLES QUE GUARDAN LA POSICION
                           // ACTUAL DE  CADA MOTOR (radianes)

float      error_act_x,error_act_y,error_ant_x,error_ant_y;//     VARIABLES     QUE
                          // ALMACENAN LOS ERRORES DEL EJE X e Y PARA
                          // EL ALGORITMO PID

float salida_x,salida_y; // SALIDA DEL ALGORITMO PID QUE REPRESENTA EL
                         // VALOR DE DUTY CYCLE EN %

//*************************************************************************************************


//***************************************
// INTERRUPCIONES EXTERNAS
//***************************************


//*************************************
// INT4: ENCODER EJE X  (PE4)
//*************************************

  SIGNAL(SIG_INTERRUPT4)
```

```
{
  if (spx_fin>spx_ini)     // SI EL PUNTO DE LLEGADA DEL EJE "X" ES MAYOR
                           // AL PUNTO DE PARTIDA,
  pulsos_x++;              // SE INCREMENTA EN UNA UNIDAD LA CANTIDAD DE
                           // PULSOS
  else                     // LUEGO DE CADA FLANCO DE SUBIDA RECIBIDO
                           // DEL ENCODER
  pulsos_x--;              // EN CASO CONTRARIO, SE DECREMENTA EN UNO.
}


//*****************************************
// INT5: SOBRECORRIENTE X (PE5)
//*****************************************
SIGNAL(SIG_INTERRUPT5)
{
 TCCR1A= 0x00;   // SI SE DETECTA SOBRECORRIENTE EN EL MOTOR DEL
 TCCR1B= 0x00;   // EJE X, SE DESACTIVA EL CANAL PWM
}


//*************************************
// INT6: ENCODER EJE Y (PE6)
//*************************************
SIGNAL(SIG_INTERRUPT6)
{
  if (spy_fin>spy_ini)    // SI EL PUNTO DE LLEGADA DEL EJE "Y" ES MAYOR
                          // AL PUNTO DE PARTIDA,
  pulsos_y++;             // SE INCREMENTA EN UNA UNIDAD LA CANTIDAD DE
                          // PULSOS
  else                    // LUEGO DE CADA CADA FLANCO DE SUBIDA
                          // RECIBIDO DEL ENCODER
  pulsos_y--;             // EN CASO CONTRARIO, SE DECREMENTA EN UNO.
}
```

```
  //****************************************
  // INT7: SOBRECORRIENTE Y (PE7)
  //****************************************

  SIGNAL(SIG_INTERRUPT7)

  {

   TCCR1A= 0x00;  // SI SE DETECTA SOBRECORRIENTE EN EL MOTOR DEL
   TCCR1B= 0x00;  // EJE Y SE DESACTIVA EL CANAL PWM

  }


void config_puertos(void)

{

 DDRB=255;  // PB5: PWM X
            // PB6: PWM Y

 PORTB=PORTB | 0b00000000;


 DDRC=255; // PC0: BIT DE GIRO EJE
           // PC1: BIT DE GIRO EJE Y


 PORTC=0;

}



void config_serial(void)

{

 UBRR0H = 0x00;

 UBRR0L = 0x67;  // @8Mhz: 0x67 --> 9600bps

 UCSR0A = 0x02;  // DOBLE VELOCIDAD: U2X = 1

 UCSR0C = 0x86;  // 8 BIT DE DATOS

 UCSR0B = 0x18;  // SIN PARIDDAD, 1 BIT DE STOP, ASINCRONO

}
```

```
void config_pwm(void)
{
  TCCR1A= 0b10100010; // MODO 14: FAST PWM, COM1A1:COM1A0 = 1:0 =
                      // COM1B1:COM1B0
  TCCR1B= 0b00011001;     // SIN PREESCALAMIENTO
  OCR1A=1;
  OCR1B=1;
  ICR1=247;
}

void config_int_ext(void)
{
   EICRB = EICRB | 0xFF;    // SE CONFIGURA LAS 4 INTERRUPCIONES
                            // EXTERNAS POR FLANCO DE SUBIDA
   EIMSK = EIMSK | 0xF0;  // SE HABILITAN LAS ULTIMAS 4 INTERRUPCIONES
                            // EXTERNAS
}

void bits_de_giro(float spx_fin,float spx_ini,float spy_fin,float spy_ini)
{
    // BIT DE GIRO X
    if (spx_fin>spx_ini)    // SI LA POSICION FINAL DEL EJE "X" ES MAYOR A LA
                            // POSICION ACTUAL
      PORTC=PORTC | 0b00000000; // EL EJE DEBE AVANZAR, EN CASO
                                // CONTRARIO  DEBERÁ RETROCEDER.
    else                                // AVANCE: BIT = 0 y RETROCESO: BIT = 1
      PORTC=PORTC | 0b00000001;

    // BIT DE GIRO Y
    if (spy_fin>spy_ini)       // LO MISMO SE CUMPLE PARA EL EJE "Y"
```

```
      PORTC=PORTC | 0b00000000;
    else
      PORTC=PORTC | 0b00000010;
}

// PROGRAMA PRINCIPAL

int main(void)  // INICIO DEL PROGRAMA PRINCIPAL
{
 float i,j,k;

 cli();   // DESABILITACION DE INTERRUPCIONES PARA INVOCAR A
          // SUBRUTINAS DE  CONFIGURACIÓN

 config_puertos();
 config_pwm();
 config_int_ext();
 config_serial();

 sei();   // HABILITACIÓN DE INTERRUPCIONES EN EL PROGRAMA

 while(1)
  {
//*****************************************
// RECEPCION DE LOS SET POINTS
//*****************************************
   while(temp<4)
    {
      If ((UCSR0A & 0x80) == 0x80 )
```

```
{
 dato=UDR0;// SE LE ASIGNA UNA VARIABLE ENTERA AL DATO
                // RECIBIDO PARA OBTENER SU CODIGO ASCII

          temp++;

     _delay_loop_2(65530);  // @8Mhz: Delay = (1/8000000)*65530*(4)
                            // = 32.8ms

   switch (temp)

   {
    case 1:dato1=dato*256;  // PRIMERO RECEPCIONA EL MSB DEL EJE X

         break;

    case 2:dato2=dato;          // LUEGO EL LSB DEL EJE X

         break;

    case 3:dato3=dato*256;   // DESPUES RECEPCIONA EL MSB DEL EJE Y

         break;

    case 4:dato4=dato;          // FINALMENTE EL LSB DEL EJE Y

          break;

   }  // FIN DEL SWITCH

  } // FIN DEL IF

 } // FIN DEL WHILE (TEMP<4)

//*************************************************************************************************


 spx_fin=dato1+dato2; // SE FORMA EL SET POINT DEL EJE X CON EL LSB Y
                      // MSB RESPECTIVO

 spy_fin=dato3+dato4; // SE FORMA EL SET POINT DEL EJE Y CON EL LSB Y
                      // MSB RESPECTIVO

 temp=0; // SE REINICIA EL CONTADOR PARA PODER RECIBIR UN SET POINT
```

```
//*****************************************************
// PERFIL DE VELOCIDAD TRAPEZOIDAL
//*****************************************************

//*************************************************************************************************
// CALCULO DEL TIEMPO FINAL, TIEMPO CRITICO Y NRO. TOTAL DE
MUESTRAS
//*************************************************************************************************


// TIEMPO FINAL:

// SE CALCULA EL TIEMPO QUE TARDA CADA EJE EN LLEGAR A SU PUNTO
// FINAL. ENTONCES, SE SABE CUAL EJE TARDA MAYOR TIEMPO EN LLEGAR
//  A SU PUNTO FINAL


   tfx=((vel_max*vel_max)-acel*(spx_ini-spx_fin))/(acel*vel_max);

   tfy=((vel_max*vel_max)-acel*(spy_ini-spy_fin))/(acel*vel_max);


  // CALCULO DE LA PENDIENTE

   pendiente=(spy_fin-spy_ini)/(spx_fin-spx_ini);// SE CALCULA LA PENDIENTE


   if (tfx>=tfy)

    {

      sp1_ini=spx_ini;  // SI EL EJE “X” DEMORA MAYOR TIEMPO QUE EL EJE “Y”

      sp1_fin=spx_fin; // EN LLEGAR AL FINAL DE SU RECORRIDO, SE

      sp2_ini=spy_ini;  // CONSIDERAN LOS PARÁMETROS DEL EJE “X” PARA
                        // CALCULAR EL TIEMPO CRITICO

      sp2_fin=spy_fin; // ESTO PARÁMETROS SON: PUNTO INICIAL, PUNTO

                        // FINAL Y TIEMPO FINAL.

      tf=tfx;

    }
```

```
else
   {
     sp1_ini=spy_ini;  //  EN CASO CONTRARIO, SE CONSIDERAN LOS
     sp1_fin=spy_fin; //  PARAMETROS DEL EJE "Y" PARA CALCULAR EL
     sp2_ini=spx_ini;  // TIEMPO CRITICO
     sp2_fin=spx_fin;
     tf=tfy;
   }

   // TIEMPO CRITICO (CALCULADO CON EL EJE DE MAYOR TIEMPO FINAL)
   tc= (sp1_ini-sp1_fin+vel_max*tf)/(vel_max);

   // NUMERO TOTAL DE MUESTRAS
   N_total = tf/0.005; // SE CALCULA LA CANTIDAD TOTAL DE MUESTRAS DEL
                       // RECORRIDO
   N=(N_total/3.00); // SE CALCULA LA CANTIDAD DE MUESTRAS POR CADA
                     //UNO DE LOS TRES TRAMOS DEL PERFIL DE VELOCIDAD
   long1=N_total; // SE ASIGNA A UNA NUEVA VARIABLE PARA UNIFORMIZAR
                  // LA NOMENCLATURA
   long2=N;      // SE ASIGNA A UNA NUEVA VARIABLE PARA UNIFORMIZAR LA
                 // NOMENCLATURA
   long3=3*long2; // SE ASIGNA A UNA NUEVA VARIABLE PARA UNIFORMIZAR
                  // LA  NOMENCLATURA
//*************************************************************************************************

   // SE VERIFICA QUE SE CUMPLA LA CONDICION DE ACELERACION MINIMA
   if (tc<(tf/2))
   {
    // TRAMO DE ACELERACION
    contador=0;
```

```
for (i=0;i<tc;i=i+(tc/(N-1)))
{
 arr_sp1[contador]=sp1_ini + 0.5*acel*i*i;
 contador++;
}

// TRAMO DE VELOCIDAD CONSTANTE
for (j=tc;j<tf-tc;j=j+((tf-2*tc)/(N-1)))
{
 arr_sp1[contador]=sp1_ini+ acel*tc*(j-(tc/2));
 contador++;
}
// TRAMO DE DESACELERACION
for (k=(tf-tc);k<tf;k=k+(tc/(N-1)))
{
 if(contador>=long3)
 { break;
 }
 arr_sp1[contador]=sp1_fin - (0.5*acel*(tf-k)*(tf-k));
 contador++;
}
contador=0;
// CONOCIENDO EL VALOR DE LA PENDIENTE SE CALCULAN LAS
// POSICIONES DEL EJE DE MENOR TIEMPO FINAL (arr_sp2)
for(contador=0;contador<long3;contador++)
{
 arr_sp2[contador]=sp2_ini+ pendiente*(arr_sp1[contador]-sp1_ini);
}
```

```
//***********************************
// CONVERSION DE mm A rad
//***********************************

// HASTA ESTA ETAPA, SE HAN CALCULADO LOS DOS ARREGLOS QUE
// CONTIENEN LOS SET POINTS DEL RECORRIDO DE CADA EJE PERO
//ESTOS SE ENCUENTRAN EN milímetros. PARA EJECUTAR EL
//CORRECTAMENTE ALGORITMO PID, LOS SET POINTS DEBEN
// ENCONTRARSE EN radianes.


if (tfx>=tfy)


  {     // SI EL TF DEL EJE "X" ES MAYOR QUE EL TF DEL EJE "Y", EL
        // "arr_sp1" CONTIENE LOS SET POINTS DEL EJE "X"

   contador=0; // POR ENDE SE LE APLICA EL FACTOR DE CONVERSIÓN
               // (2.12) DEL EJE "X" A CADA UNO DE LOS ELEMENTOS

               // DEL ARREGLO.

   for(contador=0;contador<long3;contador++)

   {

    arr_sp1[contador]=(arr_sp1[contador]*2*pi)/(2.12);

   }


   contador=0; // EN ESE CASO, EL "arr_sp2" CONTIENE LOS SET POINTS
               // DEL EJE "Y". ENTONCES, SE LE APLICA EL FACTOR DE
               // CONVERSIÓN (2.00) DEL EJE "Y" A CADA UNO DE LOS
              // ELEMENTOS DEL ARREGLO.

   for(contador=0;contador<long3;contador++)

   {

   arr_sp2[contador]=(arr_sp2[contador]*2*pi)/(2.00);

   }


   }
```

```
else
{     // SI EL TF DEL EJE "Y" ES MAYOR QUE EL TF DEL EJE "X", EL
      // "arr_sp1" CONTIENE LOS SET POINTS DEL EJE "Y"

contador=0;// POR ENDE SE LE APLICA EL FACTOR DE CONVERSIÓN
             // (2.00) DEL EJE "Y" A CADA UNO DE LOS ELEMENTOS
           // DEL ARREGLO.

for(contador=0;contador<long3;contador++)

{

  arr_sp1[contador]=(arr_sp1[contador]*2*pi)/(2.00);

}

contador=0;    // EN ESE CASO, EL "arr_sp2" CONTIENE LOS SET POINTS
               // DEL EJE "X". ENTONCES, SE LE APLICA EL FACTOR DE


for(contador=0;contador<long3;contador++)

{ // CONVERSIÓN (2.12) DEL EJE "X" A CADA UNO DE LOS ELEMENTOS
  // DEL ARREGLO

  arr_sp2[contador]=(arr_sp2[contador]*2*pi)/(2.12);

 }


}   // FIN DE LA CONVERSION DE mm A rad

} // FIN DE LA CONDICION DE ACELERACION MINIMA

//*************************************************************************************************


  // SE INVOCA A LA SUBRUTINA RESPECTIVA PARA DETERMINAR LOS
  // BITS DE GIRO DEL EJE "X" e "Y"


   bits_de_giro(spx_fin,spx_ini,spy_fin,spy_ini);


   error_ant_x=0;

   error_ant_y=0;
```

```
//**********************
//ALGORTIMO PID
//**********************


//EXISTEN DOS POSIBILIDADES DE EJECUCIÓN:

// 1) SI EL TF DEL EJE "X" ES MAYOR AL TF DEL EJE "Y", EL ARREGLO "arr
// sp1" CONTIENE LOS SET POINTS DEL EJE "X" Y POR LO TANTO SE LE
// APLICA LA ECUACION DE DIFERENCIAS DEL EJE RESPECTIVO.
// MIENTRAS QUE EL ARREGLO "arr_sp2" CONTIENE LOS SET POINTS DEL
// EJE "Y", AL CUAL SE LE APLICA LA ECUACION DE DIFERENCIAS
// CORRESPONDIENTE A ESE EJE.


// 2) SI EL TF DEL EJE "Y" ES MAYOR AL TF DEL EJE "X", EL ARREGLO "arr
// sp2" CONTIENE LOS SET POINTS DEL EJE "X" Y POR LO TANTO SE LE
// APLICA LA ECUACION DE DIFERENCIAS DEL EJE RESPECTIVO.
// MIENTRAS QUE EL ARREGLO "arr_sp1" CONTIENE LOS SET POINTS DEL
// EJE "Y", AL CUAL SE LE APLICA LA ECUACION DE DIFERENCIAS
// CORRESPONDIENTE A ESE EJE.

    if (tfx>=tfy)

    {  // PRIMERA POSIBILIDAD DE EJECUCION


     //*********
     // EJE X
     //*********

      contador=0;

      for(contador=0;contador<long3;contador++)

      {

       posx_rad=0.314*pulsos_x;// CONVERSION DE PULSOS DE ENCODER
                                // DEL  EJE "X" A RADIANES

       error_act_x=arr_sp1[contador]  -  posx_rad;// CALCULO DEL ERROR
                                                  // ACTUAL  DEL EJE "X"

       error_act_x=fabs(error_act_x);//  VALOR  ABSOLUTO  DEL  ERROR
                                      // ACTUAL DEL EJE "X"

       salida_x=(0.198*error_act_x) - (0.179982*error_ant_x); // SE APLICA LA
                                    // ECUACION DE DIFERENCIAS DEL EJE "X"
```

```
salida_x=(salida_x)*100;// SE HACE LA CONVERSION A
                        // PORCENTAJE DE ANCHO DE PULSO


        if(salida_x<min)  // SE LIMITA LA SEÑAL PWM OBTENIDA
                          // COMO SALIDA DEL ALGORITMO PID
                          // PARA NO SATURAR EL MOTOR DEL EJE X
          salida_x=min;
        if(salida_x>max)
          salida_x=max;


 // HACEMOS LA SIGUIENTE RELACION: COMO 247 (OCR1A)
 // EQUIVALE  AL 100% DE PWM, ENTONCES 100/247=0.405. POR LO
 // TANTO, CUALQUIER VALOR DE PWM RESULTANTE DEL  AL
 // ALGORITMO DIVIDIRLO ENTRE 0.405 NOS VA A DAR COMO
 // RESULTADO, LUEGO DE REDONDEARLO, UN ENTERO QUE
// CORRESPONDE A SU VALOR DE OCR1A (16 bits)


 OCR1A=floor(salida_x/0.405); // REDONDEA AL ENTERO MENOR MAS
                              //CERCANO
 error_ant_x=error_act_x;  // EL ERROR ACTUAL SE CONVIERTE EN
                           // EL ERROR ANTERIOR PARA LA
                           // SIGUIENTE EJECUCION DEL ALGORITMO PID
 _delay_loop_2(10000);   // @ 8 Mhz ---------->0.125us*10000*(4)
                         // = 5 ms (Tiempo de muestreo)


 } // FIN DEL LAZO FOR DEL EJE X



//**********
// EJE Y
//**********


  contador=0;
```

```
for(contador=0;contador<long3;contador++)

{

  posy_rad=0.314*pulsos_y;    // CONVERSION DE PULSOS DE
                              // ENCODER  DEL EJE "Y" A RADIANES

  error_act_y=arr_sp2[contador] - posy_rad; // CALCULO DEL ERROR
                                            // ACTUAL DEL EJE "Y"

   error_act_y=fabs(error_act_y);  // VALOR ABSOLUTO DEL ERROR
                                   // ACTUAL DEL EJE "Y"

   salida_y=(0.042*error_act_y) - (0.035994*error_ant_y); // SE APLICA
                                   // LA ECUACION DE DIFERENCIAS
                                   // DEL EJE "Y"

  salida_y=(salida_y)*100;         // SE HACE LA CONVERSION A
                                   // PORCENTAJE DE  ANCHO DE PULSO


        if(salida_y<min)  // SE LIMITA LA SEÑAL PWM OBTENIDA
                          // COMO  SALIDA DEL ALGORITMO PID
                          // PARA NO SATURAR EL MOTOR DEL EJE Y

            salida_y=min;

        if(salida_y>max)

            salida_y=max;


 OCR1B=floor(salida_y/0.405);


   error_ant_y=error_act_y; // EL ERROR ACTUAL SE CONVIERTE EN
                            // EL ERROR ANTERIOR PARA LA
                            // SIGUIENTE EJECUCION DEL ALGORITMO PID


  _delay_loop_2(10000); // @ 8 Mhz -->(1/8Mhz)*10000*(4)= 5 ms (Tiempo
                        // de muestreo)

 } // FIN DEL LAZO FOR DEL EJE Y

} // FIN DE LA PRIMERA POSIBILIDAD DE EJECUCION
```

```
Else

{ // SEGUNDA POSIBILIDAD DE EJECUCION

//**********
// EJE X
//**********

 contador=0;

  for(contador=0;contador<long3;contador++)

  {

   posx_rad=0.314*pulsos_x; // CONVERSION DE PULSOS DE ENCODER
                            / /DEL EJE "X" A RADIANES

   error_act_x=arr_sp2[contador]  -  posx_rad;  // CALCULO DEL ERROR
                                                // ACTUAL DEL EJE "X"

   error_act_y=fabs(error_act_y);  // VALOR ABSOLUTO DEL ERROR
                                   // ACTUAL DEL EJE "X"

   salida_x=(0.198*error_act_x) - (0.179982*error_ant_x); // SE APLICA LA
                                   // ECUACION DE DIFERENCIAS DEL
                                   // EJE "X"

   salida_x=(salida_x)*100;    // SE HACE LA CONVERSION A
                               // PORCENTAJE DE ANCHO DE PULSO


     if(salida_x<min)   // SE LIMITA LA SEÑAL PWM OBTENIDA COMO
                        // SALIDA DEL ALGORITMO PID PARA NO
                        // SATURAR EL MOTOR DEL EJE X

         salida_x=min;

   if(salida_x>max)

         salida_x=max;


   OCR1A=floor(salida_x/0.405);
```

```
    error_ant_x=error_act_x; // EL ERROR ACTUAL SE CONVIERTE EN
                             // EL ERROR ANTERIOR PARA LA
                             // SIGUIENTE EJECUCION DEL ALGORITMO PID


    _delay_loop_2(10000);// @ 8 Mhz -->(1/8Mhz)*10000*(4)= 5 ms
                         // (Tiempo de muestreo)


  }  // FIN DEL LAZO FOR DEL EJE X


//*********
// EJE Y
//*********


 contador=0;

  for(contador=0;contador<long3;contador++)

  {

   posy_rad=0.314*pulsos_y;   // CONVERSION DE PULSOS DE
                              // ENCODER DEL EJE "Y" A RADIANES

    error_act_y=arr_sp1[contador] - posy_rad; // CALCULO DEL ERROR
                                              // ACTUAL DEL EJE "Y"

    error_act_x=fabs(error_act_x); //  VALOR ABSOLUTO DEL ERROR
                                   // ACTUAL DEL EJE "Y"

    salida_y=(0.042*error_act_y) - (0.035994*error_ant_y); // SE APLICA LA
                                   // ECUACION DE DIFERENCIAS DEL EJE "Y"

    salida_y=(salida_y)*100;   // SE HACE LA CONVERSION A
                               // PORCENTAJE DE ANCHO DE PULSO


     if(salida_y<min)  // SE LIMITA LA SEÑAL PWM OBTENIDA COMO
                       // SALIDA DEL ALGORITMO PID PARA NO SATURAR
                       // EL MOTOR DEL EJE X

          salida_y=min;

     if(salida_y>max)

          salida_y=max;
```

```
                OCR1B=floor(salida_y/0.405);

                error_ant_y=error_act_y; // EL ERROR ACTUAL SE CONVIERTE EN
                                         // EL ERROR ANTERIOR PARA LA
                                         // SIGUIENTE EJECUCION DEL ALGORITMO PID


                _delay_loop_2(10000);  // @ 8 Mhz -->(1/8Mhz)*10000*(4)= 5 ms
                                       // (Tiempo de muestreo)

              } // FIN DEL LAZO FOR DEL EJE Y

            } // FIN DE LA SEGUNDA POSIBILIDAD DE EJECUCION

  contador=0;


  // EL SET POINT ACTUAL SE GUARDA COMO EL SET POINT ANTERIOR
  // PARA EL SIGUIENTE MOVIMIENTO

  spx_ini=spx_fin;

  spy_ini=spy_fin;


  } // FIN DEL WHILE (1)


} // FIN DEL PROGRAMA PRINCIPAL
```