PROBLEMAS DE CONTROL CON ARDUINO: MÉTODO DE ASIGNACIÓN DE ESTADOS.

Un método que puede resultar práctico, por lo sistemático, para resolver los problemas de control de sistemas técnicos es el que vamos a designar por “método de la asignación de estados”.

Los sistemas técnicos que controlaremos tendrán normalmente **receptores** que activar y desactivar (motores, señalizaciones visuales, señalizaciones acústicas, etc) y elementos **sensores** que aportan información sobre posiciones, presencia, etc (finales de carrera, detectores de luz o calor, etc).

Los pines de nuestra placa Arduino que definamos como **salidas** actuarán sobre los receptores del sistema técnico controlado (para activarlos o desactivarlos) y los pines que definamos como **entradas** recibirán información sobre la situación del sistema técnico desde los sensores que hay en él.

Groso modo, entenderemos por **estados** de un sistema a las diferentes situaciones en las que se puede encontrar dicho sistema.

El sistema formado por el sistema técnico y el sistema que lo controla (circuito electrónico o programa informático) puede estar en diferentes estados. Lo que distingue unos estados de otros puede ser que estén activados receptores diferentes o bien que los sensores del sistema estén aportando informaciones diferentes o que el sistema deba reaccionar de forma diferente ante una misma variación en los sensores, etc.

Pasos del método

1.- La primera tarea que hay que realizar en el “método de la asignación de estados” es establecer los diferentes estados en que se puede encontrar el sistema, en función de las especificaciones de funcionamiento. A cada estado se le asigna un nombre (preferentemente un número de 1 en adelante)

2.- Definir las variaciones de las entradas (sensores) que suponen la transición de un estado a otro. Se supone que dos entradas no pueden variar simultáneamente, una lo hará siempre un breve instante antes que la otra.

3.- Para clarificar gráficamente las transiciones entre estados se elabora un grafo orientado. Cada estado se representa por un círculo que encierra al número que se le ha asignado. Entre círculo y círculo se indica mediante una flecha orientada la variación de la entrada que provoca la transición de un estado a otro.

4.- Antes de empezar a programar hay que decidir en qué pines de la placa Arduino se va a conectar cada receptor (estos pines actuarán como salidas) y cada sensor del sistema técnico (estos pines actuarán como entradas).

5.- El procedimiento de control se inicia con un bloque de instrucciones para declarar las variables globales.

Aparte de las variables específicas de cada programa, definiremos una variable, que bien puede llamarse ***estado***, destinada a memorizar el estado en que se encuentra el sistema controlado en cada momento.

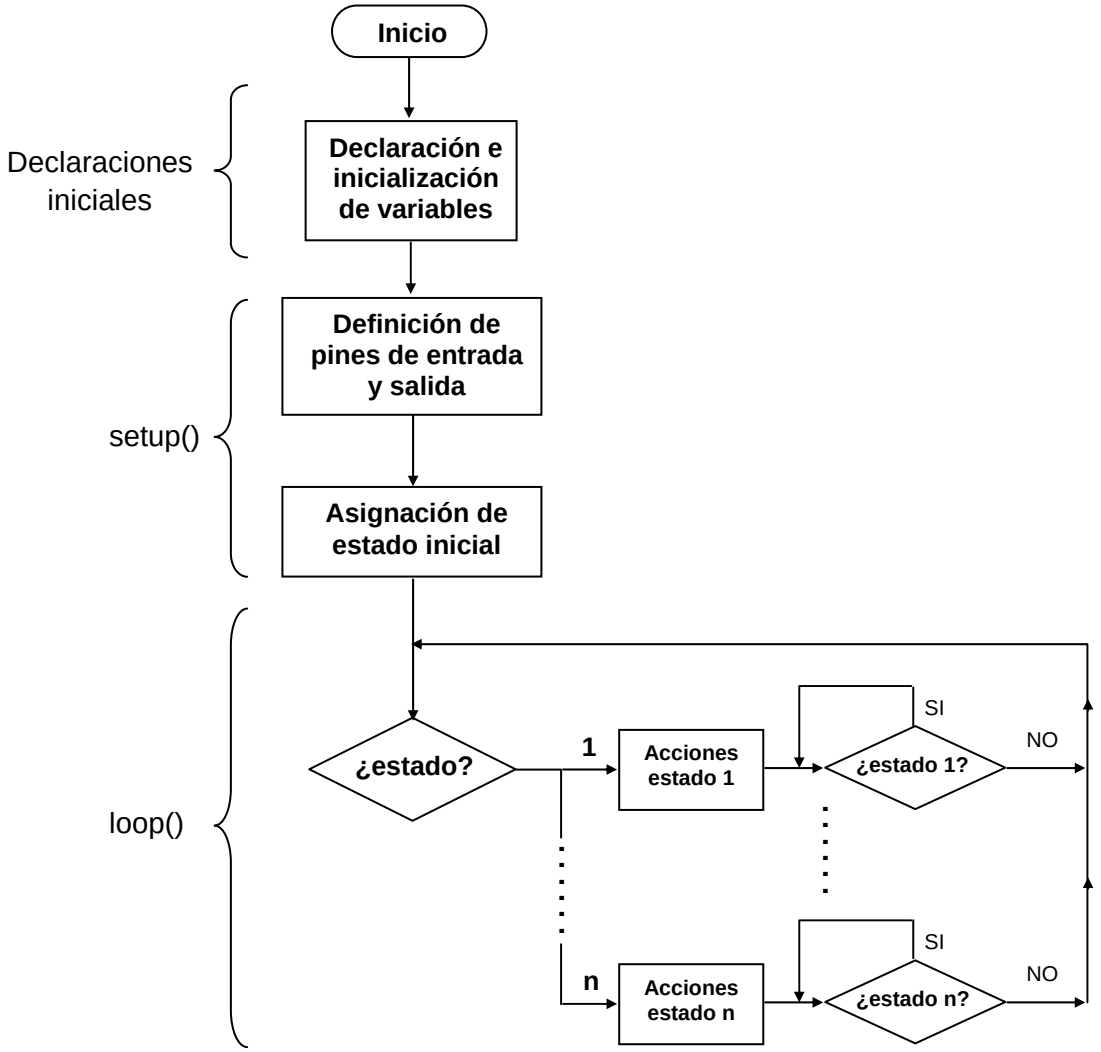
6.- Le sigue la función setup() que contendrá las instrucciones de definición de los pines como entradas o salidas y también contendrá unas instrucciones cuya función será evaluar, a través de las indicaciones de los sensores, la situación inicial del sistema y asignar un estado inicial. Puede haber otras instrucciones de inicialización que sean necesarias.

7.- Para cada estado hay que diseñar una función que, además de activar o desactivar las salidas correspondientes del sistema de control, vigile las entradas que le provocan las transiciones de este estado. Cuando se cumplen las variaciones de las entradas, la función actúa variando el valor de la variable *estado*. La transición entre estados también puede tener lugar por tiempo.

8.- Por último, se define la función recursiva loop() de forma que vigile permanentemente la variable *estado* y en función de ella ejecute la función asociada al estado correspondiente.

El **diagrama de flujo** de un programa de control de un sistema genérico, a través del método de la asignación de estados, tendría la siguiente estructura:

- Se inicia el programa declarando las variables necesarias (entre ellas la variable estado).
- Se definen los pines de entrada y salida.
- Se ejecutan las instrucciones que evalúan la situación del sistema a partir de los sensores y asigna un estado inicial.
- A continuación se entra en un procedimiento recursivo que va examinando el estado en el que se encuentra el sistema y en función del mismo se ejecutan las acciones correspondientes a dicho estado.
- Las funciones correspondientes a cada estado constan normalmente de un conjunto de acciones de control (activaciones y desactivaciones de salidas y lectura de entradas) y un bucle de espera en la salida hasta que el sistema cambia de estado, volviendo al programa recursivo, que hace que se ejecute el procedimiento correspondiente al nuevo estado.
- En el caso de que la placa esté conectada al ordenador por el puerto USB, podemos añadir en las funciones de cada estado instrucciones para que se visualice en el monitor del puerto serie información sobre el estado en que se encuentra (esto es especialmente útil en la fase de diseño para detectar errores).



EJEMPLO 1: CONTROL DE LLENADO DE UN DEPÓSITO

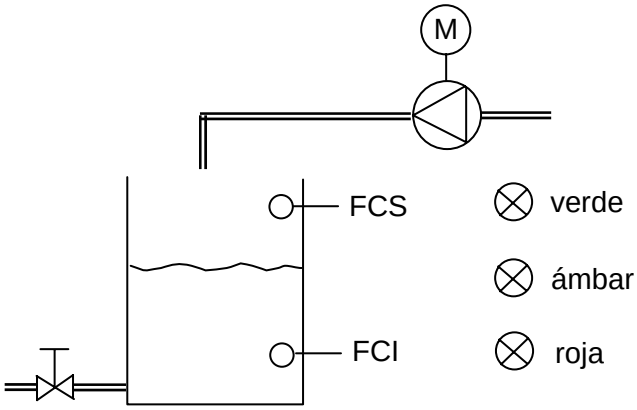
Queremos hacer el sistema de control de llenado de un depósito de agua por medio de una bomba.

El depósito dispone de dos detectores de nivel, uno en la parte superior que detecta cuando está lleno y otro en la parte inferior que detecta cuando está próximo a vaciarse.

Para que la bomba no esté continuamente arrancando y parando, lo cual acabaría dañando el motor, queremos que empiece a llenar cuando llegue al nivel inferior y deje de llenar cuando llegue al superior

Queremos disponer de tres LEDs indicadores, cuyo encendido tenga los siguientes significados:

– LED verde indica depósito lleno hasta el nivel superior
– LED ámbar indica depósito entre ambos niveles.
– LED rojo indica depósito casi vacío, es decir, en el nivel inferior.



Solución:

Notación: Llamaremos LV, LA y LR a los tres LEDs y M al motor de la bomba. Cuando éstos se hacen 1 lógico se activan. Llamaremos FCS y FCI a los finales de carrera detectores de nivel y supondremos que dan un 1 lógico cuando el agua los cubre.

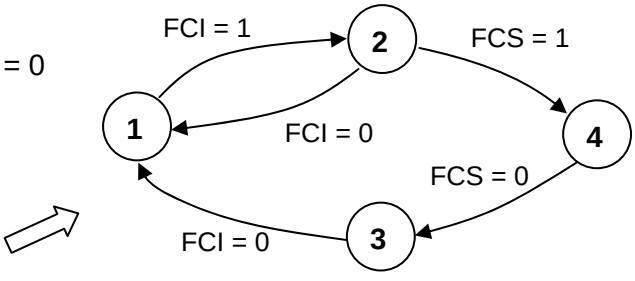
1- Primero analizamos los estados del sistema: detectamos a simple vista cuatro situaciones o estados diferentes en que puede estar el sistema:

- **Estado 1**: depósito vacío; el sistema debe reaccionar haciendo LR = 1, LA = 0, LV = 0 y M = 1.
- **Estado 2**: depósito en nivel intermedio llenándose (procede de una situación de vacío); el sistema debe reaccionar haciendo LR = 0, LA = 1, LV = 0 y M = 1
- **Estado 3**: depósito en nivel intermedio vaciándose (procede de una situación de lleno); el sistema debe reaccionar haciendo LR = 0, LA = 1, LV = 0 y M = 0.
- **Estado 4**: depósito lleno; el sistema debe reaccionar haciendo LR = 0, LA = 0, LV = 1 y M = 0.

2.- Las transiciones entre estados vendrán dadas por:

- Pasa de estado 1 a estado 2 cuando FCI = 1.
- Pasa de estado 2 de nuevo a estado 1 si FCI = 0
- Pasa de estado 2 a estado 4 si FCS = 1.
- Pasa de estado 4 a estado 3 si FCS = 0.
- Pasa de estado 3 a estado 1 si FCI = 0.

3.- Representamos esta información en un grafo:



4.- Vamos a decidir los pines digitales que usaremos como entradas y salidas en la placa:

Salida M..... en pin 3
Salida LR ..... en pin 4
Salida LA..... en pin 5
Salida LV..... en pin 6

Entrada FCI.....en pin 7
Entrada FCS .....en pin 8

5.- Incluimos un bloque de instrucciones inicial para declarar e inicializar las variables.

```
int MotorPin=3;
int LRPin=4;
int LAPin=5;
int LVPin=6;
int FCIPin=7;
int FCSPin=8;
int estado;
```

6.- En la función setup() definimos los pines que serán entradas y salidas. También incluimos las instrucciones destinadas a asignar un estado inicial.

Para la asignación de estado en función de la situación de las entradas, el programa puede hacer:

- Si FCS = 1 asignar estado 4
- Si FCI = 0 asignar estado 1
- Si no se cumple ninguna de las anteriores (o sea: FCS = 0 y FCI = 1) asignar estado 3

Obsérvese, que en la última situación también podemos asignar estado 2 y no habría problema. Queda a nuestro criterio.

En nuestro caso sería:

```
void setup(){
  pinMode(MotorPin,OUTPUT);
  pinMode(LRPin,OUTPUT);
  pinMode(LAPin,OUTPUT);
  pinMode(LVPin,OUTPUT);
  pinMode(FCIPin,INPUT);
  pinMode(FCSPin,INPUT);
  //Leemos entradas para asignar estado inicial:
  if(digitalRead(FCSPin)==HIGH){estado=4;}
  else if(digitalRead(FCIPin)==LOW){estado=1;}
  else{estado=3;}
}
```

7.- Cada estado tendrá una función asociada que active y desactive las salidas adecuadas y vigile las entradas para ver cuando tiene que producirse la transición de estado.

Para la función correspondiente al estado 1, por ejemplo, el código es:

```
void estado1(){
   digitalWrite(MotorPin,HIGH);
   digitalWrite(LRPin,HIGH);
   digitalWrite(LAPin,LOW);
   digitalWrite(LVPin,LOW);
   do{
       if(digitalRead(FCIPin)==HIGH){estado=2;}
     }while(estado==1)
   return;}
```

Las cuatro primeras instrucciones activan y desactivan salidas, según corresponde al estado. La quinta instrucción es un bucle repetitivo do...while que se ejecuta una y otra vez hasta que la variable *estado* deje de valer 1, lo cual ocurre cuando una de las veces que lea la entrada FCI ésta esté en nivel HIGH, con lo que el valor de la variable *estado* pasará a ser 2.

Una vez que salimos del bucle, la instrucción *return* hace que se salga de la función estado1 (aunque esta instrucción no es necesaria en este caso)

8.- Definimos la función recursiva loop(). Esta función debe hacer que se ejecute una u otra de las funciones asociadas a los estados, según el valor de la variable estado.

El código de la función loop() sería:

```
void loop(){
   if(estado==1){estado1();}
   if(estado==2){estado2();}
   if(estado==3){estado3();}
   if(estado==4){estado4();}
}
```

El código completo quedaría:

```
// Programa para control de llenado de depósito con Arduino
// Declaramos las variables que guardan los números de pines
int MotorPin=3;
int LRPin=4;
int LAPin=5;
int LVPin=6;
int FCIPin=7;
int FCSPin=8;
int estado;

void setup(){

//Definimos los pines de salida y de entrada
  pinMode(MotorPin,OUTPUT);
  pinMode(LRPin,OUTPUT);
  pinMode(LAPin,OUTPUT);
  pinMode(LVPin,OUTPUT);
  pinMode(FCIPin,INPUT);
  pinMode(FCSPin,INPUT);
//Leemos entradas para asignar estado inicial:
  if(digitalRead(FCSPin)==HIGH){estado=4;}
  else if(digitalRead(FCIPin)==LOW){estado=1;}
  else{estado=3;}
}

void loop(){
   if(estado==1){estado1();}
   if(estado==2){estado2();}
   if(estado==3){estado3();}
   if(estado==4){estado4();}
}

void estado1(){
   //Este estado corresponde a nivel por debajo de mínimo, llenamos.
   digitalWrite(MotorPin,HIGH);
   digitalWrite(LRPin,HIGH);
   digitalWrite(LAPin,LOW);
   digitalWrite(LVPin,LOW);
   do{
       if(digitalRead(FCIPin)==HIGH){estado=2;}
     }while(estado==1)
   return;}

void estado2(){
   //Este estado corresponde a nivel intermedio llenando
```

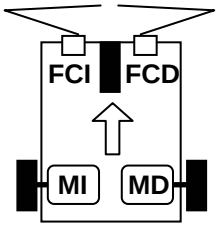
```
    digitalWrite(MotorPin,HIGH);
    digitalWrite(LRPin,LOW);
    digitalWrite(LAPin,HIGH);
    digitalWrite(LVPin,LOW);
    do{
        if(digitalRead(FCSPin)==HIGH){estado=4;}
        if(digitalRead(FCIPin)==LOW){estado=1;}
       }while(estado==2)
    return;}
void estado3(){
    //Este estado corresponde a nivel intermedio vaciando
    digitalWrite(MotorPin,LOW);
    digitalWrite(LRPin,LOW);
    digitalWrite(LAPin,HIGH);
    digitalWrite(LVPin,LOW);
    do{
        if(digitalRead(FCIPin)==LOW){estado=1;}
       }while(estado==3)
    return;}

void estado4(){
    //Este estado corresponde a depósito lleno
    digitalWrite(MotorPin,LOW);
    digitalWrite(LRPin,LOW);
    digitalWrite(LAPin,LOW);
    digitalWrite(LVPin,HIGH);
    do{
        if(digitalRead(FCSPin)==LOW){estado=3;}
       }while(estado==4)
    return;}
```

EJEMPLO 2: VEHÍCULO EXPLORADOR

Queremos diseñar el control de un vehículo explorador que avance hacia delante hasta que tope con algún obstáculo. Tras topar, retrocederá un breve tiempo haciendo un giro y volverá a avanzar.

El vehículo dispone de dos antenas detectoras que accionan finales de carrera; están situadas en la parte delantera del vehículo, una a cada lado.



Si el vehículo topa por su lado izquierdo, el vehículo retrocederá manteniendo parado el motor del lado izquierdo, de forma que cuando vuelva a avanzar lo haga hacia la derecha, evitando el obstáculo. Cuando topa por el lado derecho, retrocederá manteniendo parado el motor derecho. Es casi imposible que un choque de frente sea tan simétrico que accione ambos finales de carrera al mismo tiempo, en tal caso nos da igual que retroceda hacia la derecha o hacia la izquierda.

Solución:

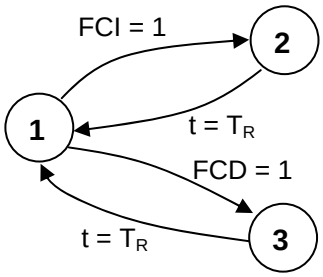
Notación: Llamaremos MI y MD a los motores izquierdo y derecho respectivamente y FCI y FCD a los finales de carrera izquierdo y derecho respectivamente. Supondremos que FCI y FCD se ponen a 1 cuando son accionados. El tiempo de retroceso lo designamos por T_R. Los motores no se conectarán a la placa Arduino directamente, sino a través del chip L293 que amplifica las señales que le llegan desde la placa y las aplica al motor. Por ello, cuando digamos que la salida tal o cual de Arduino se conecta al motor, debe entenderse siempre que es a través del chip L293.

1- Primero analizamos los estados del sistema: detectamos a simple vista tres situaciones o estados diferentes en que puede estar el sistema:

- **Estado 1**: el vehículo avanza hacia delante. El sistema de control debe hacer girar MI en sentido antihorario (mirando al motor desde su acoplamiento a la rueda) y MD en sentido horario.
- **Estado 2**: el vehículo retrocede metiendo la trasera hacia la izquierda. El sistema de control debe mantener parado MI y hacer girar MD en sentido antihorario.
- **Estado 3**: el vehículo retrocede metiendo la trasera hacia la derecha. El sistema de control debe mantener parado MD y hacer girar MI en sentido horario.

2.- Las transiciones entre estados vendrán dadas por:

- Pasa de estado 1 a estado 2 cuando FCI = 1.
- Pasa de estado 2 de nuevo a estado 1 tras $t=T_R$.
- Pasa de estado 1 a estado 3 si FCD = 1.
- Pasa de estado 3 de nuevo a estado 1 tras $t=T_R$.



3.- Representamos esta información en un grafo:

4.- Vamos a decidir los pines digitales que usaremos como entradas y salidas en la placa:

Salida MI-1..... en pin 3	Entrada FCI.....en pin 4
Salida MI-2 ..... en pin 9	Entrada FCS .....en pin 2
Salida MD-1 ..... en pin 5	
Salida MD-2 ..... en pin 6	

5.- Incluimos un bloque de instrucciones inicial para declarar e inicializar las variables.

```
int MI1Pin=3;
int MI2Pin=9;
int MD1Pin=5;
```

```
int MD2Pin=6;
int FCIPin=4;
int FCDPin=2;
int TR=2000;//tiempo de retroceso 2000 ms
int estado;
```

6.- En la función setup() definimos los pines que serán entradas y salidas. También incluimos las instrucciones destinadas a asignar un estado inicial.

Para la asignación de estado en función de la situación de las entradas, el programa puede hacer:

- Si FCI = 1 asignar estado 2
- Si FCI = 0 y FCD = 1 asignar estado 3
- Si FCI = 0 y FCD = 0) asignar estado 1

En nuestro caso sería:

```
void setup(){
  pinMode(MI1Pin,OUTPUT);
  pinMode(MI2Pin,OUTPUT);
  pinMode(MD1Pin,OUTPUT);
  pinMode(MD2Pin,OUTPUT);
  pinMode(FCIPin,INPUT);
  pinMode(FCDPin,INPUT);
  //Leemos entradas para asignar estado inicial:
  if(digitalRead(FCIPin)==HIGH){estado=2;}
  else if(digitalRead(FCDPin)==HIGH){estado=3;}
  else{estado=1;}
}
```

7.- Cada estado tendrá una función asociada que active y desactive las salidas adecuadas y vigile las entradas para ver cuando tiene que producirse la transición de estado.

Para la función correspondiente al estado 1, el código es:

```
void estado1(){
   digitalWrite(MI1Pin,LOW);
   digitalWrite(MI2Pin,HIGH);
   digitalWrite(MD1Pin,LOW);
   digitalWrite(MD2Pin,HIGH);
   do{
       if(digitalRead(FCDPin)==HIGH){estado=3;}
       if(digitalRead(FCIPin)==HIGH){estado=2;}
     }while(estado==1)
   return;}
```

Las cuatro primeras instrucciones determinan que los motores giren ambos de forma que el vehículo avance hacia adelante. La quinta instrucción es un bucle repetitivo do...while que se ejecuta una y otra vez hasta que la variable *estado* deje de valer 1, lo cual ocurre cuando una de las veces que lea las entradas FCD y FCI alguna esté en nivel HIGH.

Una vez que salimos del bucle, la instrucción *return* hace que se salga de la función estado1 (aunque esta instrucción no es necesaria en este caso)

Para la función correspondiente al estado 2, el código es:

```
void estado2(){
   digitalWrite(MI1Pin,LOW);
   digitalWrite(MI2Pin,LOW);
```

```
    digitalWrite(MD1Pin,HIGH);
    digitalWrite(MD2Pin,LOW);
    delay(TR);
    estado=1;
    return;
 }
```

Para el estado 3, sería:

```
void estado3(){
    digitalWrite(MI1Pin,HIGH);
    digitalWrite(MI2Pin,LOW);
    digitalWrite(MD1Pin,LOW);
    digitalWrite(MD2Pin,LOW);
    delay(TR);
    estado=1;
    return;

 }
```

8.- Definimos la función recursiva loop(). Esta función debe hacer que se ejecute una u otra de las funciones asociadas a los estados, según el valor de la variable estado.

El código de la función loop() sería:

```
void loop(){
    if(estado==1){estado1();}
    if(estado==2){estado2();}
    if(estado==3){estado3();}
}
```

Código completo del programa vehículo explorador

```
int MI1Pin=3;
int MI2Pin=9;
int MD1Pin=5;
int MD2Pin=6;
int FCIPin=4;
int FCDPin=2;
int TR=2000;//tiempo de retroceso 2000 ms
int estado;

void setup(){
  pinMode(MI1Pin,OUTPUT);
  pinMode(MI2Pin,OUTPUT);
  pinMode(MD1Pin,OUTPUT);
  pinMode(MD2Pin,OUTPUT);
  pinMode(FCIPin,INPUT);
  pinMode(FCDPin,INPUT);
  //Leemos entradas para asignar estado inicial:
  if(digitalRead(FCIPin)==HIGH){estado=2;}
  else if(digitalRead(FCDPin)==HIGH){estado=3;}
  else{estado=1;}
}

void loop(){
    if(estado==1){estado1();}
    if(estado==2){estado2();}
    if(estado==3){estado3();}
```

```
}
void estado1(){
   digitalWrite(MI1Pin,LOW);
   digitalWrite(MI2Pin,HIGH);
   digitalWrite(MD1Pin,LOW);
   digitalWrite(MD2Pin,HIGH);
   do{
       if(digitalRead(FCDPin)==HIGH){estado=3;}
       if(digitalRead(FCIPin)==HIGH){estado=2;}
     }while(estado==1)
   return;}
void estado2(){
   digitalWrite(MI1Pin,LOW);
   digitalWrite(MI2Pin,LOW);
   digitalWrite(MD1Pin,HIGH);
   digitalWrite(MD2Pin,LOW);
   delay(TR);
   estado=1;
   return;}
void estado3(){
   digitalWrite(MI1Pin,HIGH);
   digitalWrite(MI2Pin,LOW);
   digitalWrite(MD1Pin,LOW);
   digitalWrite(MD2Pin,LOW);
   delay(TR);
   estado=1;
   return;}
```