RESOLUCIÓN DE PROBLEMAS DE CONTROL CON ARDUINO POR EL MÉTODO DE LA ASIGNACIÓN DE ESTADOS.

Es un método muy sistemático para resolver los problemas de control programado de sistemas técnicos. Requiere más líneas de código, lo que lo hace parecer a priori poco práctico. Si bien esto es así para problemas simples, conforme va aumentando la complejidad del problema se ve la utilidad de esa metodología.

Los sistemas técnicos que controlaremos tendrán normalmente **actuadores** que activar y desactivar (motores, señalizaciones visuales, señalizaciones acústicas, etc) y elementos **sensores** que aportan información sobre la situación del sistema y de la interacción con los usuarios, como pueden ser pulsaciones, posiciones de objetos, presencia, etc (pulsadores, finales de carrera, detectores de luz o calor, etc).

Los pines de nuestra placa Arduino que definamos como **salidas** gobernarán los actuadores 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 técnico a las diferentes situaciones en las que se puede encontrar dicho sistema. Habitualmente, lo que distingue unos estados de otros suele venir dada por los actuadores que están activados en cada estado y por la información que aportan los sensores del sistema. La diferencia entre dos estados también puede deberse a que el sistema deba reaccionar de forma diferente ante una misma variación en los sensores.

> Nota: En algunos casos nos puede convenir definir estados diferentes para una situación de activación de los actuadores y de información aportada por los sensores idénticas; suele ser cuando a una misma situación del sistema se llega por causas diferentes y que dan lugar a que el sistema evolucione hacia diferentes estados en función de dichas causas. Ya se entenderá mejor esto cuando surja algún ejemplo.

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 causas que provocan las transiciones de unos estados a otros. Suelen ser variaciones de las entradas (sensores) o temporizaciones. Dada la gran velocidad a la que se ejecutan los programas, se supone que dos entradas no pueden cambiar justo en el mismo instante, una lo hará siempre un breve instante antes que la otra.

3.- Para clarificar gráficamente las transiciones entre estados se elabora un grafo orientado que llamaremos **"diagrama de estados"**. Cada estado se representa por un círculo que encierra al número que se le ha asignado. Entre unos círculos y otros se indican mediante flechas las causas que provocan las transiciones entre estados.

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 código del programa de control se inicia con un bloque de instrucciones para declarar las variables globales y asignarles valores iniciales cuando nos convenga. También se definen las constantes.

Aparte de las variables específicas que requiera cada programa, definiremos una variable de tipo entero, que bien podemos llamar ***estado***, destinada a guardar el número del 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

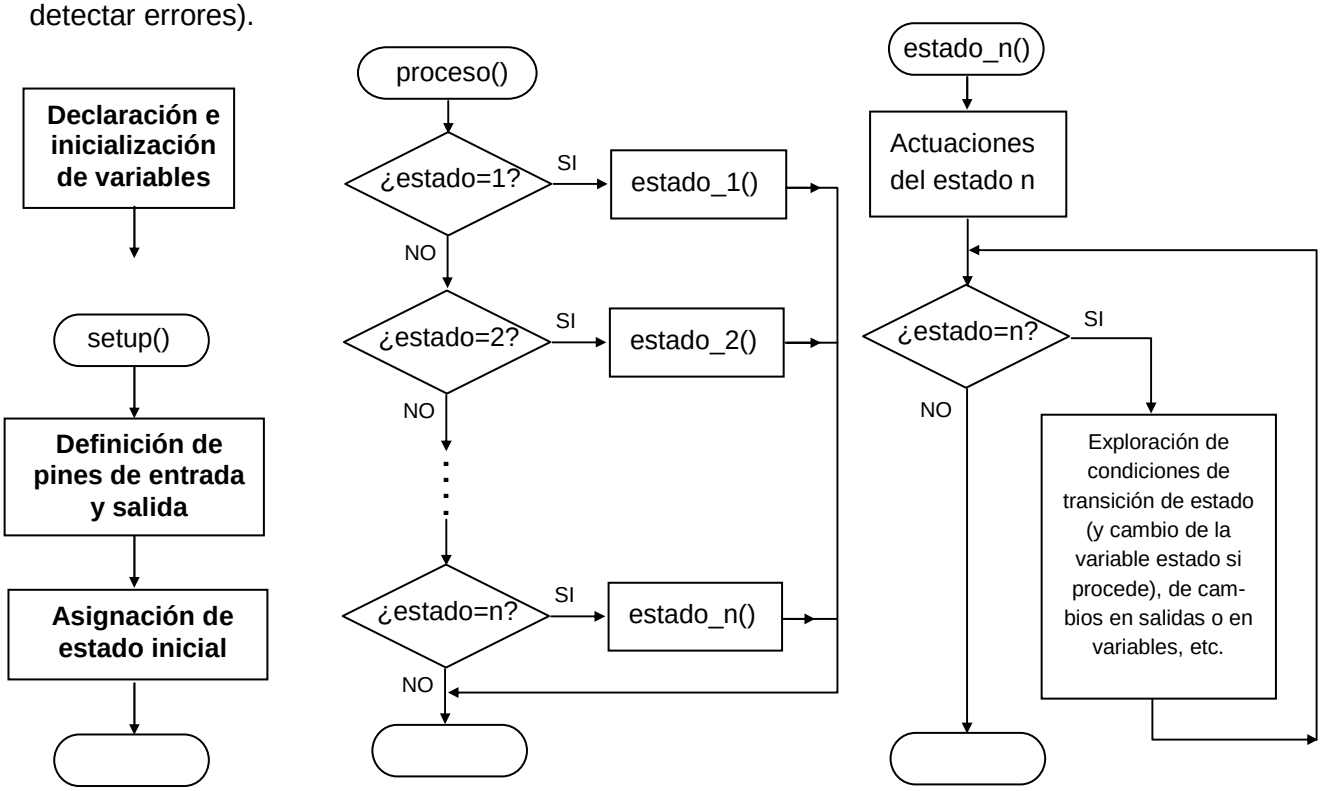
de los sensores, la situación inicial del sistema y asignar el estado inicial más adecuado. Puede haber otras instrucciones de inicialización que sean necesarias.

7.- Para cada uno de los estados hay que diseñar una función que, además de activar o desactivar las salidas correspondientes del sistema de control, vigile las condiciones que provocan las transiciones de este estado a otros o cambios en salidas o variable. Cuando se cumplen condiciones de cambio de estado, la función varía el valor de la variable *estado* y termina.

8.- Por último, se define un procedimiento que sea llamado por la función recursiva *loop()* de forma que vigile la variable *estado* y en función de su valor 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 se asigna un estado inicial.
- El procedimiento llamado por loop() va examinando el valor de la variable estado y en función del mismo se ejecuta la función correspondiente al estado en que se encuentre el sistema. Este procedimiento constará de una función *switch case* o un grupo de bucles *if*.
- Las funciones correspondientes a cada estado constan normalmente de un conjunto de actuaciones de control (activaciones y desactivaciones de salidas) y un bucle en el que se exploran los posibles eventos, como las condiciones de cambio de estado, de cambios en las salidas o en variables; en el momento en el que se detecta que el sistema cambia de estado, se asigna a la variable *estado* el valor del nuevo estado y finaliza la función, volviendo el control al procedimiento recursivo loop(), que hará que se ejecute la función 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

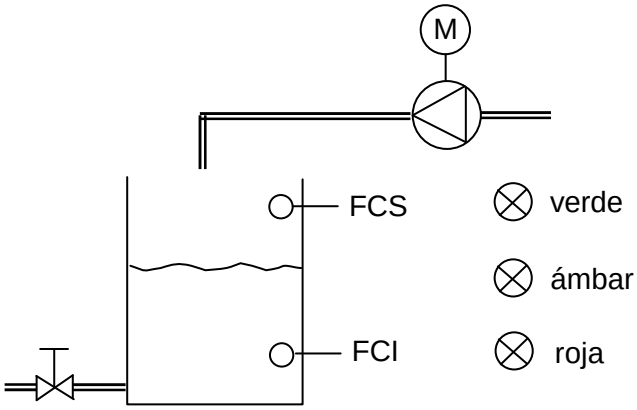
Queremos hacer el sistema de control de llenado de un depósito de agua por medio de una bomba.

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 (HIGH) se activan y cuando se hacen 0 lógico (LOW), se desactivan. Llamaremos FCS y FCI a los finales de carrera detectores de nivel y supondremos que dan un 1 lógico cuando el agua los cubre.

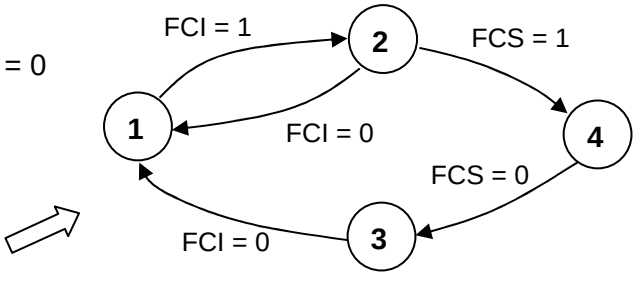
1- Primero analizamos los estados del sistema: detectamos a simple vista cuatro situaciones o estados diferentes en que puede estar el sistema:

- **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 LR ..... en pin 2
Salida LA ..... en pin 3
Salida LV ..... en pin 4
Motor..... en pines 5 y 6

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

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

```
#define LR 2
#define LA 3
#define LV 4
#define Ma 5
#define Mb 6
#define FCI 7
#define FCS 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 inicial 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(LR, OUTPUT);
  pinMode(LA, OUTPUT);
  pinMode(LV, OUTPUT);
  pinMode(Ma, OUTPUT);
  pinMode(Mb, OUTPUT);
  pinMode(FCI, OUTPUT);
  pinMode(FCS, OUTPUT);
  if(digitalRead(FCS)==HIGH) estado=4;
  else if (digitalRead(FCI)==LOW) estado=1;
  else estado=3;
}
```

7.- Cada estado tendrá una función asociada que active y desactive las salidas adecuadas y vigile las condiciones de cambio de estado para ver cuando tiene que producirse dicho cambio.

Las funciones correspondientes a los estados serían las siguientes:

```
void estado_1(){
  digitalWrite(Ma,HIGH);
  digitalWrite(Mb,LOW);
  digitalWrite(LR,HIGH);
  digitalWrite(LA,LOW);
  digitalWrite(LV,LOW);
  while(estado==1){
    if(digitalRead(FCI)==HIGH) estado=2;
  }
}
```

```
void estado_2(){
  digitalWrite(Ma,HIGH);
  digitalWrite(Mb,LOW);
  digitalWrite(LR,LOW);
  digitalWrite(LA,HIGH);
  digitalWrite(LV,LOW);
  while(estado==2){
    if(digitalRead(FCS)==HIGH) estado=4;
    if(digitalRead(FCI)==LOW) estado=1;
  }
}
```

```
void estado_3(){
  digitalWrite(Ma,LOW);
  digitalWrite(Mb,LOW);
  digitalWrite(LR,LOW);
  digitalWrite(LA,HIGH);
  digitalWrite(LV,LOW);
  while(estado==3){
    if(digitalRead(FCI)==LOW) estado=1;
  }
}
```

```
void estado_4(){
  digitalWrite(Ma,LOW);
  digitalWrite(Mb,LOW);
  digitalWrite(LR,LOW);
  digitalWrite(LA,LOW);
  digitalWrite(LV,HIGH);
  while(estado==4){
    if(digitalRead(FCS)==LOW) estado=3;
  }
}
```

8.- Definimos la función que hará que se ejecute una u otra de las funciones asociadas a los estados, según el valor de la variable estado. Llamemos a está función, por ejemplo, proceso(). Su código podría realizarse con una función switch case o con un bloque de if. Esta función sería llamada por loop().

```
void proceso() {
  if(estado==1) estado_1();
  if(estado==2) estado_2();
  if(estado==3) estado_3();
  if(estado==4) estado_4();
}
```

```
void proceso() {
  switch(estado){
   case 1:
     estado_1();
     break;
   case 2:
     estado_2();
     break;
   case 3:
     estado_3();
     break;
   case 4:
     estado_4();
     break;
  }
}
```

La función loop() quedaría

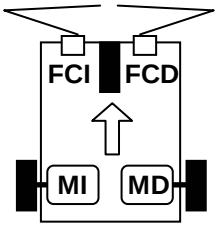
```
void loop(){
  proceso();
}
```

Si nuestro programa sólo tuviera que controlar el llenado del depósito, la función proceso(() podría coincidir con la propia función loop().

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:

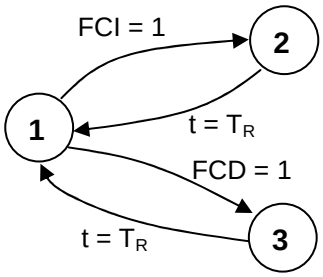
Notación: Llamaremos MI y MD a los motores izquierdo y derecho respectivamente y FCI y FCD a los finales de carrera izquierdo y derecho respectivamente. Supondremos que FCI y FCD se ponen a 1 lógico (HIGH) cuando son accionados. El tiempo de retroceso lo designamos por T_R.

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 MIa..... en pin 5	Entrada FCI..... en pin 2
Salida MIb ..... en pin 6	Entrada FCD ..... en pin 4
Salida MDa ..... en pin 10	
Salida MDb ..... en pin 11	

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

```
int MIa=5;   int MIb=6;
int MDa=10;  int MDb=11;
int FCI=2;   int FCD=3;
int TR=1500; //tiempo de retroceso en 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(MIa, OUTPUT); pinMode(MIb, OUTPUT);
  pinMode(MDa, OUTPUT); pinMode(MDb, OUTPUT);
  pinMode(FCI, INPUT);  pinMode(FCD, INPUT);
  if(digitalRead(FCI)==HIGH)estado=2;
  else if (digitalRead(FCD)==HIGH)estado=3;
  else estado=1;
}
```

7.- Las funciones correspondientes a los estados y a la función loop() serían las siguientes:

```
void estado_1(){
  digitalWrite(MIa,LOW);
  digitalWrite(MIb,HIGH);
  digitalWrite(MDa,HIGH);
  digitalWrite(MDb,LOW);
  while(estado==1){
    if(digitalRead(FCD)==HIGH)estado=3;
    if(digitalRead(FCI)==HIGH)estado=2;
  }
}
```

```
void estado_2(){
   digitalWrite(MIa,LOW);
   digitalWrite(MIb,LOW);
   digitalWrite(MDa,LOW);
   digitalWrite(MDb,HIGH);
   delay(TR);
   estado=1;
 }
```

```
void estado_3(){
   digitalWrite(MIa,HIGH);
   digitalWrite(MIb,LOW);
   digitalWrite(MDa,LOW);
   digitalWrite(MDb,LOW);
   delay(TR);
   estado=1;
 }
```

8.- Definimos la función recursiva loop(), que es siempre del mismo tipo en el método de la asignación de estados:

```
void loop(){
  if(estado==1)estado_1();
  if(estado==2)estado_2();
  if(estado==3)estado_3();
}
```