
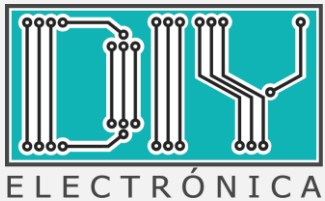
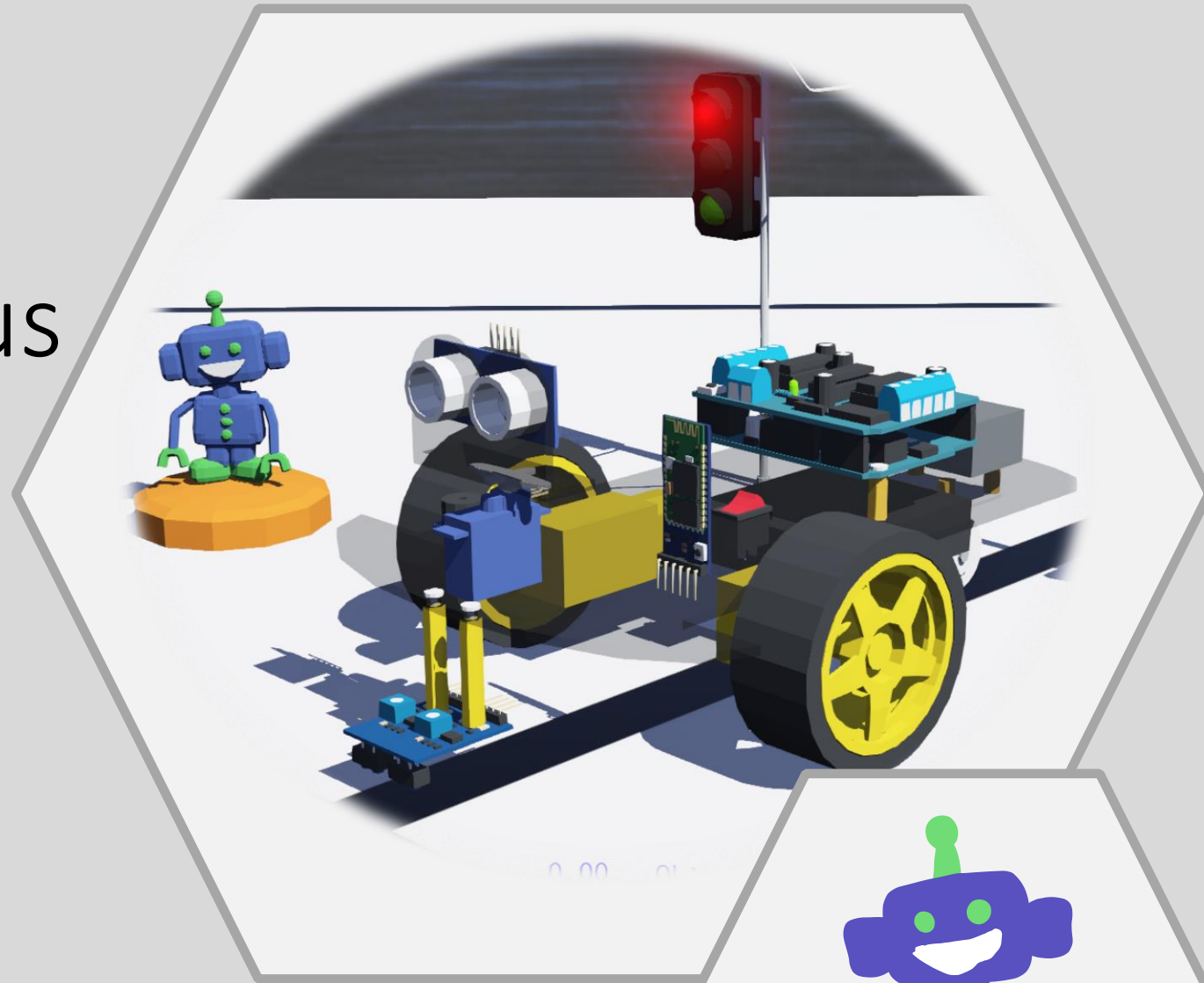


Kit Smart Car 2WD Plus

Código (Controlador)

 Kit_Smart_Car_2WD_Plus_V2.ino



DrakerDG 28-02-2022





• Kit_Smart_Car_2WD_Plus_V2.ino

Librerías

- SoftwareSerial.h
 - Se utiliza para virtualizar un puerto serial y así conectar el módulo de Bluetooth HC-05
 - Declaración:

```
#include <SoftwareSerial.h>
```
 - Instancia:

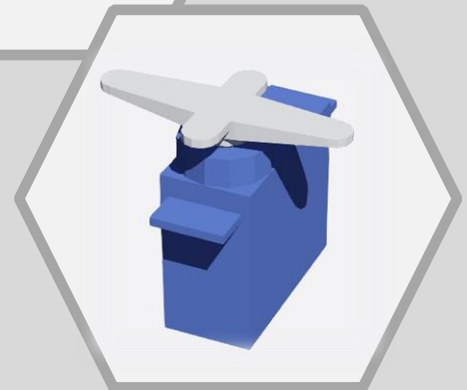
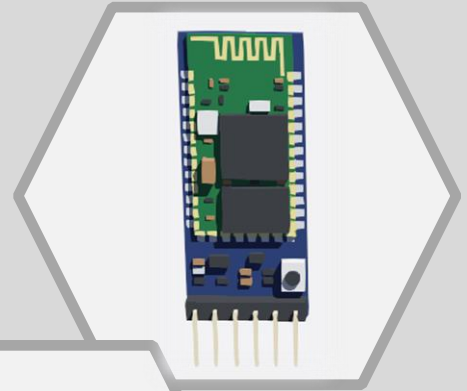
```
SoftwareSerial myBT(19, 18); // RX, TX
```
- AFMotor.h
 - Se utiliza para controlar los motores por medio del módulo Shield Motor Driver L293D
 - Declaración:

```
#include <AFMotor.h>
```
 - Instancias:

```
AF_DCMotor motorR(4);  
AF_DCMotor motorL(3);
```
- Servo.h
 - Se utiliza para controlar el servomotor SG90
 - Declaración:

```
#include <Servo.h>
```
 - Instancia:

```
Servo servoU;
```





• Kit_Smart_Car_2WD_Plus_V2.ino

Variables y constantes

- Variables de operación

```
char BT_cmd;  
int mode = 1;
```

- Estatus del robot

```
int on_Robot = 0;
```

- Cantidad de sensores

```
#define IR 2
```

- Pines de los sensores

```
const byte pSen[IR] = {17, 16};
```

limites de los sensores IR de línea

- Valores máximos de los sensores

```
int sen_max[] = {300, 300};
```

- Valores mínimos de los sensores

```
int sen_min[] = {300, 300};
```

- Valores de los sensores

```
int sen_val[] = {0, 0};
```

- Límite máximo del valor del sensor

```
const int KS = 1000;
```

- Variable de calibración

```
int cal_IRS = 0;
```

- Tiempo de calibración

```
int cal_TME = 0;
```

- Variable de detección de línea

```
int online = 0;
```



• Kit_Smart_Car_2WD_Plus_V2.ino

Constantes de PID

- Constante proporcional

```
#define Kp 0.13
```

- Constante integral

```
#define Ki 0.0001
```

- Constante derivativa

```
#define Kd 0.001
```

- Máxima velocidad de los motores

```
#define maxSR 255  
#define maxSL 240
```

Variables de PID

- Error proporcional

```
float P_error;
```

- Error integral

```
float I_error;
```

- Error derivativo

```
float D_error;
```

- Valor de PID

```
float PID_value;
```

- Ultimo error

```
float lastError;
```

- Tiempo actual

```
unsigned long cTime;
```

- Tiempo anterior

```
unsigned long pTime;
```

- Tiempo transcurrido

```
float eTime;
```

- Velocidad de los motores

```
int speedR = 0;  
int speedL = 0;
```



15 por compensación por
diferencia de velocidad
entre los motores



• Kit_Smart_Car_2WD_Plus_V2.ino

- Pines del sensor ultrasónico

```
#define trig1 15  
#define echo1 14
```

- Matriz de distancia a objetos

```
int distances[21];
```

- Vectores de detección de objetos

```
int xVec = 0;  
int yVec = 0;
```

- Tiempo para movimiento del servo

```
#define period 20
```

- Pasos del servomotor

```
#define lap1 9
```

- Contador de pasos

```
int i = 10;
```

- Dirección de giro

```
int CCW = 1;
```

- Límite de detección de obstáculos

```
#define lim1 45
```

- Límite de parada

```
#define vec1 -10
```

- Límite de retroceso

```
#define vec2 -9
```

- Constante proporcional de detección de obstáculos

```
#define Ke 1.5
```

- Tiempo de parada

```
#define Stop 200
```

- Tiempo de giro

```
#define Turn 350
```

- Estatus de detección del robot

```
int avoidSR = 0;
```

- Temporizadores de tono (sonido)

```
unsigned long Time0 = 0;  
unsigned long Time1;
```



• Kit_Smart_Car_2WD_Plus_V2.ino

- Declaración de módulos y funciones

```
// Declaración de módulos y funciones
int get_DIS(void); // Obtiene el valor de la distancia con el sensor ultrasónico
void set_INI(void); // Configura el estado inicial del robot
void set_IRS(void); // Calibración de sensores IR
long get_IRS(long); // Obtiene el valor de los sensores IR
void get_PID(void); // Obtiene el calculo del valor PID
void set_VEC(void); // Inicializando la matriz de vectores de distancia
void uss_RAD(void); // Escaneo del sensor ultrasónico
void get_VEC(void); // Cálculo del vector de distancia (error)
void snd_BZR(int) ; // Generador de sonido
void set_SPD(void); // Establece la velocidad de los motores
void run_FWD(void); // Desplazamiento hacia adelante
void run_BWD(void); // Desplazamiento hacia atrás
void run_STP(void); // Detenido
void run_TRN(int) ; // Giro (izquierda o derecha)
```



• Kit_Smart_Car_2WD_Plus_V2.ino

Módulo de configuración

```
void setup() {  
  myBT.begin(9600); // Velocidad de datos del puerto SoftwareSerial  
  
  servoU.attach(9); // Pin del servomotor (anexado)  
  servoU.write(i * lap1); // Posición central inicial  
  delay(100);  
  
  // Configuración de motores #4 y #3  
  motorR.setSpeed(200);  
  motorR.run(RELEASE);  
  motorL.setSpeed(200);  
  motorL.run(RELEASE);  
  
  // Configuración de los pines del sensor ultrasonico  
  pinMode (trig1, OUTPUT);  
  pinMode (echo1, INPUT);  
  
  set_VEC(); // Inicialización de la matriz del vector de distancias (Módulo)  
}
```



• Kit_Smart_Car_2WD_Plus_V2.ino

Módulo Principal

- Selección de modo de operación (Bluetooth)

```
void loop() {  
    if (myBT.available()) BT_cmd = myBT.read(); // Lee el puerto serial del Bluetooth  
  
    if (BT_cmd == '1') { // Modo seguidor de lineas seleccionado (1)  
        mode = 1;  
        set_INI(); // Inicialización de servomotor  
    }  
  
    else if (BT_cmd == '2') { // Modo para evitar obstaculos seleccionado (2)  
        mode = 2;  
        set_INI(); // Inicialización de servomotor  
    }  
  
    else if (BT_cmd == '3') { // Modo de control por Bluetooth seleccionado (3)  
        mode = 3;  
        set_INI(); // Inicialización de servomotor  
    }  
}
```




• Kit_Smart_Car_2WD_Plus_V2.ino

Módulo Principal

- Modo seguidor de línea

```
// Modo seguidor de linea
if (mode == 1) {
    if ((BT_cmd == 'A') && (cal_IRS == 1)) { // Inicia el modo seguidor de línea
        on_Robot = 1;
        snd_BZR(3);
        motorR.run(FORWARD);
        motorL.run(FORWARD);
    }
    else if (BT_cmd == 'B') { // Detiene el robot
        snd_BZR(3);
        on_Robot = 0;
        run_STP();
    }
    else if ((BT_cmd == 'C') && (cal_IRS == 0)) { // Modo de calibración (giro a la derecha)
        speedR = 160;
        speedL = 160;
        snd_BZR(3);
        run_TRN(1);
        set_IRS();
    }
}
```



• Kit_Smart_Car_2WD_Plus_V2.ino

Módulo Principal

- Modo seguidor de línea

```
else if ((BT_cmd == 'D') && (cal_IRS == 0)) { // Modo de calibración (giro a la izquierda)
    speedR = 160;
    speedL = 160;
    snd_BZR(3);
    run_TRN(-1);
    set_IRS();
}
else if ((BT_cmd == 'Z') && (cal_IRS == 0)) { // Detiene el robot
    run_STP();
}
if (on_Robot == 1) { // Seguidor de línea
    get_PID();
    motorR.setSpeed(speedR);
    motorL.setSpeed(speedL);
}
}
```



• Kit_Smart_Car_2WD_Plus_V2.ino

Módulo Principal

- Modo para evitar obstáculos

```
// Modo para evitar obstaculos
else if (mode == 2) {
    if (BT_cmd == 'A') { // Inicia el modo para evitar obstaculos
        snd_BZR(3);
        on_Robot = 1;
    }
    else if (BT_cmd == 'B') { // Detiene el robot
        snd_BZR(3);
        on_Robot = 0;
        run_STP();
    }
}
```



• Kit_Smart_Car_2WD_Plus_V2.ino

Módulo Principal

- Modo para evitar obstáculos

```
if (on_Robot == 1) { // Desplazamiento hacia adelante y escanea
  uss_RAD();
  if (avoidSR == 0) {
    set_SPD();
    run_FWD();
  }
  else if (avoidSR == 1) { // Detiene el robot cuando detecta un obstaculo
    run_STP();
    delay(Stop);
    avoidSR = 2;
  }
  else if (avoidSR == 2) { // Se desplaza hacia atras para evitar un obstaculo
    run_BWD();
  }
  else if (avoidSR == 3) { // Turn to avoid an obstacle - Gira para evitar un obstacle
    run_TRN(xVec/abs(xVec));
    delay(Turn);
    avoidSR = 0;
  }
}
```



• Kit_Smart_Car_2WD_Plus_V2.ino

Módulo Principal

- Modo de control por Bluetooth

```
// Modo de control por Bluetooth
else if (mode == 3) {
    if (BT_cmd == 'A') { // Desplazamiento hacia adelante
        snd_BZR(3);
        speedR = maxSR;
        speedL = maxSL;
        run_FWD();
    }
    else if (BT_cmd == 'B') { // Desplazamiento hacia atrás
        snd_BZR(3);
        run_BWD();
    }
    else if (BT_cmd == 'C') { // Giro hacia la derecha
        snd_BZR(3);
        run_TRN(1);
    }
}
```



• Kit_Smart_Car_2WD_Plus_V2.ino

Módulo Principal

- Modo de control por Bluetooth

```
    else if (BT_cmd == 'D') { // Giro hacia la izquierda
        snd_BZR(3);
        run_TRN(-1);
    }
    else if (BT_cmd == 'Z') { // Detiene el robot
        run_STP();
    }
}

BT_cmd = '\n';
}
```



• Kit_Smart_Car_2WD_Plus_V2.ino

Módulos y funciones (Detalle)

```
// Declaración de módulos y funciones
int get_DIS(void); // Obtiene el valor de la distancia con el sensor ultrasónico
void set_INI(void); // Configura el estado inicial del robot
void set_IRS(void); // Calibración de sensores IR
long get_IRS(long); // Obtiene el valor de los sensores IR
void get_PID(void); // Obtiene el calculo del valor PID
void set_VEC(void); // Inicializando la matriz de vectores de distancia
void uss_RAD(void); // Escaneo del sensor ultrasónico
void get_VEC(void); // Cálculo del vector de distancia (error)
void snd_BZR(int) ; // Generador de sonido
void set_SPD(void); // Establece la velocidad de los motores
void run_FWD(void); // Desplazamiento hacia adelante
void run_BWD(void); // Desplazamiento hacia atrás
void run_STP(void); // Detenido
void run_TRN(int) ; // Giro (izquierda o derecha)
```



- Kit_Smart_Car_2WD_Plus_V2.ino
- Obtiene el valor de la distancia con el sensor ultrasónico

```
// Obtiene el valor de la distancia con el sensor ultrasónico
int get_DIS(void){
    long Duration1;
    int Distance1;
    digitalWrite(trig1, LOW);
    delayMicroseconds(2);
    digitalWrite(trig1, HIGH);
    delayMicroseconds(10);
    digitalWrite(trig1, LOW);
    Duration1 = pulseIn(echo1, HIGH);
    Distance1 = Duration1*0.03432/2; // Cálculo de la distancia
    return Distance1;
}
```




- Kit_Smart_Car_2WD_Plus_V2.ino

- Configura el estado inicial del robot

```
// Configura el estado inicial del robot
void set_INI(void) {
    on_Robot = 0;           // Estado del robot
    servoU.write(90);       // Posición frontal
    snd_BZR(5);             // Sonido
}
```



• Kit_Smart_Car_2WD_Plus_V2.ino

- Calibración de sensores IR

```
// Calibración de sensores IR
void set_IRS(void){
    int senx = 0;
    for (int k = 0; k < IR; k++){
        senx = analogRead(pSen[k]);
        if (senx < sen_min[k]) sen_min[k] = senx; // Determinación del valor mínimo
        if (senx > sen_max[k]) sen_max[k] = senx; // Determinación del valor máximo
    }
    cal_TME += 1;
    if ((cal_TME > 20) && (sen_max[0] > sen_min[0]) && (sen_max[1] > sen_min[1])) {
        run_STP(); // Detiene el robot
        snd_BZR(3); // Hace un sonido
        delay(500);
        noTone(10);
        delay(50);
        snd_BZR(10); // Hace un sonido
        delay(500);
        noTone(10);
        cal_IRS = 1; // Calibración finalizada
    }
}
```



• Kit_Smart_Car_2WD_Plus_V2.ino

- Obtiene el valor de los sensores IR

```
// Obtiene el valor de los sensores IR
long get_IRS(long Pant){
    online = 0;
    unsigned long Perr;
    unsigned long avgS = 0;
    unsigned int sumS = 0;
    for(int k = 0; k < IR; k++){
        sen_val[k] = analogRead(pSen[k]); // Leyendo los sensores
        sen_val[k] = map(sen_val[k], sen_min[k], sen_max[k], 0, KS); // Mapeando los valores (Entre 0 y 1000)
        sen_val[k] = constrain(sen_val[k], 0, KS); // Limitando los valores entre 0 y 1000 (KS)
        if (sen_val[k]>150) online = 1; // Condicion de detección de linea
        // Adición de valores de sensor por factor de posición del sensor (K * KS) donde KS = 1000
        avgS += (long)sen_val[k]*(k * KS);
        sumS += sen_val[k]; // Adición de valores de sensores
    }
    if (online == 1) Perr = avgS/sumS - 500; // Si detecta la línea, Perr es el valor promedio menos 500
    else Perr = Pant; // Si no detecta la línea, Perr es igual a Pant (último valor: memoria)
    delayMicroseconds(140); // Tiempo para procesar los valores del sensor en el microcontrolador
    return Perr;
}
```



• Kit_Smart_Car_2WD_Plus_V2.ino

- Obtiene el calculo del valor PID

```
// Obtiene el calculo del valor PID
void get_PID(void){
    int medR, medL;
    P_error = get_IRS(P_error); // Obtiene el error P
    cTime = millis(); // Obtiene el tiempo del microprocesador
    eTime = (float)(cTime - pTime) / 1000; // Calcula el tiempo transcurrido
    I_error = I_error * 2 / 3 + P_error * eTime; // Calcula el error I
    D_error = (P_error - lastError) / eTime; // Calcula el error D
    PID_value = Kp * P_error + Ki * I_error + Kd * D_error; // Calcula el valor PID

    lastError = P_error; // Salva el error P

    medR = maxSR - abs(PID_value); // Calcula la velocidad base del motor derecho
    medL = maxSL - abs(PID_value); // Calcula la velocidad base del motor izquierdo
    speedR = medR - PID_value; // Calcula la velocidad del motor derecho
    speedL = medL + PID_value; // Calcula la velocidad del motor izquierdo
}
```



- Kit_Smart_Car_2WD_Plus_V2.ino

- Inicializando la matriz de vectores de distancia

```
// Inicializando la matriz de vectores de distancia
void set_VEC(void){
    for (int j = 0; j <= 20; j++){
        distances[j] = lim1;
    }
}
```



• Kit_Smart_Car_2WD_Plus_V2.ino

- Escaneo del sensor ultrasónico

```
// Escaneo del sensor ultrasónico
void uss_RAD(void) {
    servoU.write(i * lap1); // Establece la posición del servomotor
    delay(period);
    distances[i] = get_DIS(); // Obtiene la distancia del objeto
    get_VEC(); // Obtiene el vector de posición
    if (distances[i] < lim1) snd_BZR(distances[i]); // Sonido de tono dependiendo de la distancia
    if ((yVec < vec1) and (avoidSR == 0)) avoidSR = 1; // Stop robot
    if ((yVec > vec2) and (avoidSR == 2)) avoidSR = 3; // Turn robot

    i = i + CCW; // Incrementa o decrementa el contador de posición del servomotor
    if (i < 0) {
        i = 1;
        CCW = 1;
    }
    if (i > 20) {
        i = 19;
        CCW = -1;
    }
}
```



- Kit_Smart_Car_2WD_Plus_V2.ino

- Cálculo del vector de distancia (error)

```
// Cálculo del vector de distancia (error)
void get_VEC(void) {
    long vectx = 0;
    long vecty = 0;
    int objects = 0;
    for (int j = 0; j <= 20; j++) {
        if (distances[j] < lim1) {
            vectx += sin((j - 10) * lap1 * PI / 180) * (distances[j] - lim1); // Componente x
            vecty += cos((j - 10) * lap1 * PI / 180) * (distances[j] - lim1); // Componente y
            objects += 1;
        }
    }
    if (objects > 0) { // Calcula los promedios del error en "x" y "y"
        xVec = vectx / objects;
        yVec = vecty / objects;
    }
    else {
        xVec = 0;
        yVec = 0;
    }
}
```



- Kit_Smart_Car_2WD_Plus_V2.ino

- Generador de sonido

```
// Generador de sonido
void snd_BZR(int distx) {
    int Interval;
    Time1 = millis();
    // El valor del intervalo del tono, depende de la distancia
    Interval = 15 * distx;
    Interval = constrain(Interval, 50, 2500);
    if (Time1 - Time0 >= Interval) {
        Time0 = Time1;
        // La frecuencia del tono, depende de la distancia
        tone(10, 1760 + 1320 - distx * 66, 25);
    }
}
```




• Kit_Smart_Car_2WD_Plus_V2.ino

- Establece la velocidad de los motores

```
// Establece la velocidad de los motores
void set_SPD(void) {
    int errP = int(xVec * Ke);    // Calcula el error proporcional

    speedR = maxSR - abs(errP);  // Calcula la velocidad base del motor derecho
    speedL = maxSL - abs(errP);  // Calcula la velocidad base del motor izquierdo

    speedR = speedR + errP;      // Calcula la velocidad del motor derecho
    speedL = speedL - errP;      // Calcula la velocidad del motor izquierdo

    // Restringe el valor de velocidad máxima y mínima (derecha e izquierda)
    speedR = constrain(speedR, 0, 255);
    speedL = constrain(speedL, 0, 255);
}
```



- Kit_Smart_Car_2WD_Plus_V2.ino

- Desplazamiento hacia adelante y Desplazamiento hacia atrás

```
// Desplazamiento hacia adelante
void run_FWD(void) {
    motorR.run(FORWARD);
    motorL.run(FORWARD);
    motorR.setSpeed(speedL);
    motorL.setSpeed(speedL);
    delay(3);
}
```

```
// Desplazamiento hacia atrás
void run_BWD(void) {
    motorR.run(BACKWARD);
    motorL.run(BACKWARD);
    motorR.setSpeed(maxSR);
    motorL.setSpeed(maxSL);
    delay(3);
}
```



- Kit_Smart_Car_2WD_Plus_V2.ino

- Detenido y Giro (izquierda o derecha)

```
// Detenido
void run_STP(void) {
    motorR.run(RELEASE);
    motorL.run(RELEASE);
    delay(3);
}

// Giro (izquierda o derecha)
void run_TRN(int CW) {
    if (CW > 0) {
        motorR.run(FORWARD);
        motorL.run(BACKWARD);
    }
    else {
        motorR.run(BACKWARD);
        motorL.run(FORWARD);
    }
    motorR.setSpeed(maxSR);
    motorL.setSpeed(maxSL);
    delay(3);
}
```



• Kit_Smart_Car_2WD_Plus_V2.ino

Calibración de constantes PID (Prueba y Error)

1. Establecer la velocidad máxima de cada motor
2. Con los valores de Ki y Kd en cero, incrementar el valor de Kp hasta que el robot mantenga una oscilación pero sin perder la línea
 - Establecer Kp aproximadamente a la mitad del valor del paso anterior
 - Si el robot es inestable y pierde la línea, puede ser necesario disminuir la velocidad de cada motor e iniciar nuevamente desde el paso 2
3. Incrementar Ki hasta que el robot sea inestable pero sin perder la línea
 - Establecer Ki entre el 10% al 50% del último valor de Ki
4. Incrementar Kd hasta obtener la menor oscilación posible

```
// Máxima velocidad de los motores
// 15 por compensación por diferencia de velocidad entre los motores
#define maxSR 255 // 240 + 15
#define maxSL 240

// Constantes de PID
#define Kp 0.13 // Constante proporcional
#define Ki 0.0001 // Constante integral
#define Kd 0.001 // Constante derivativa
```

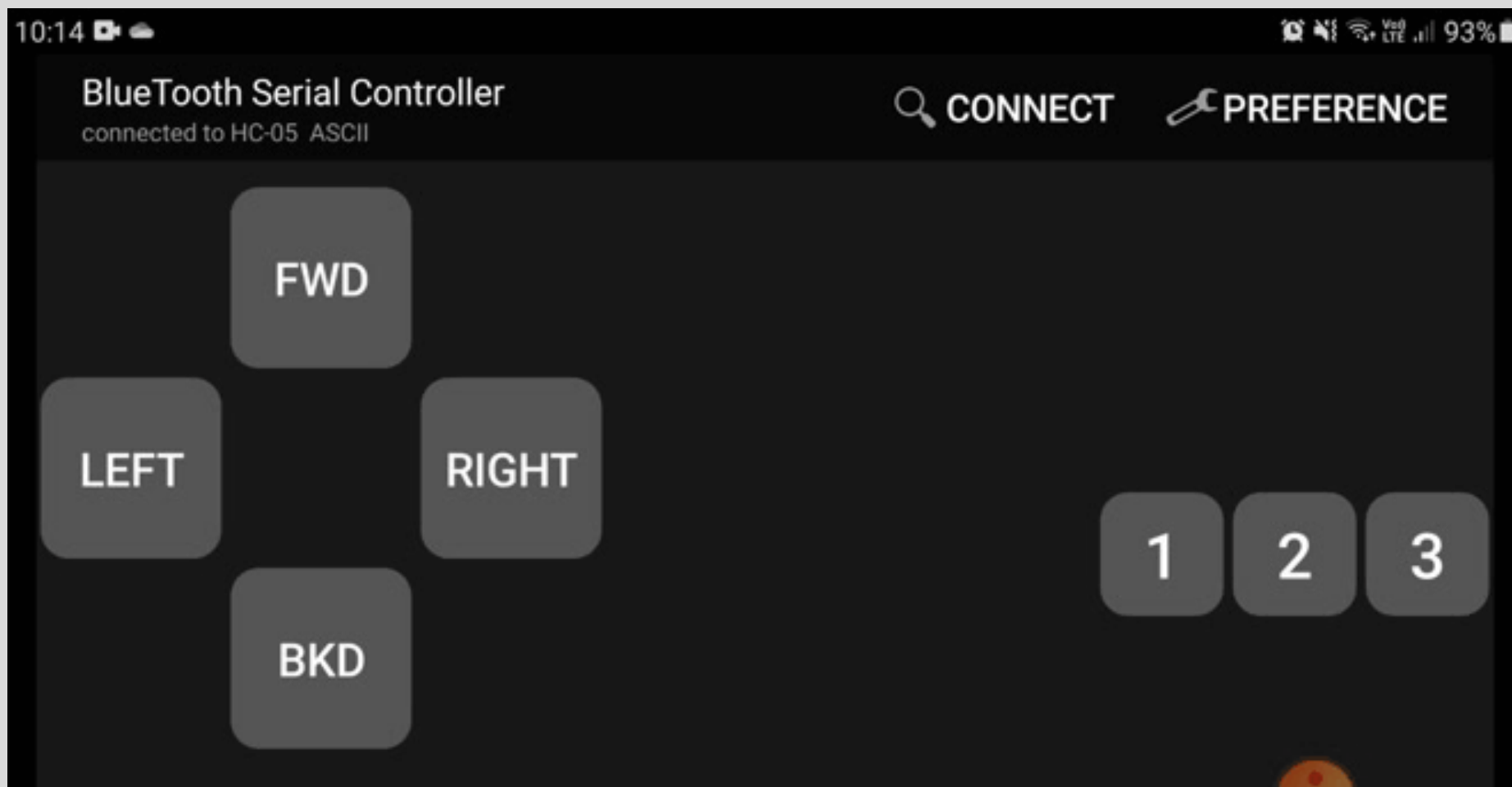


• Bluetooth Serial Controller

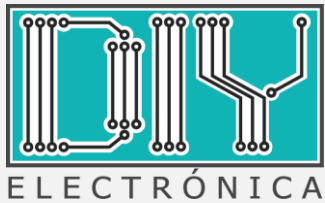
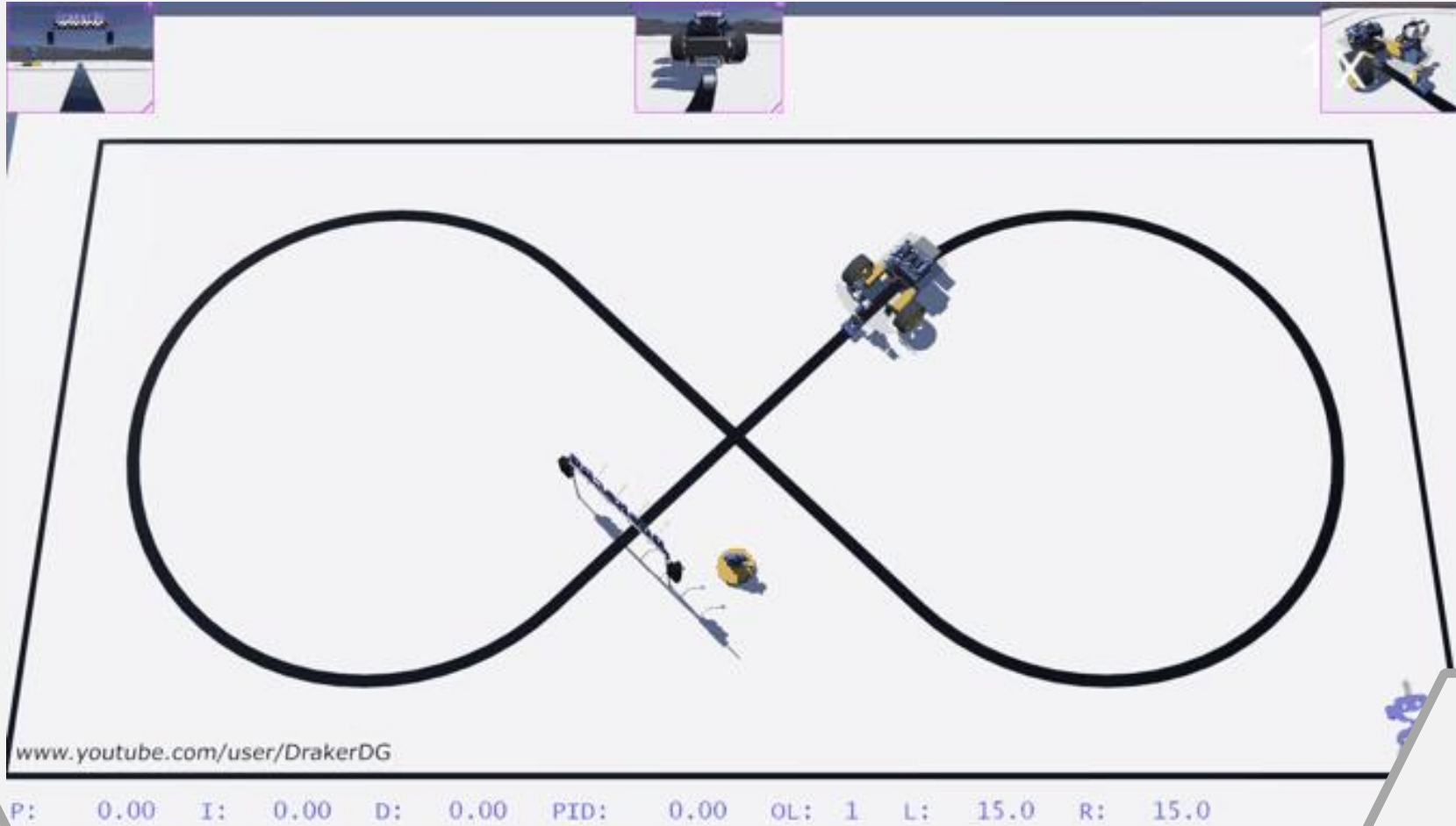


Controlador serial por Bluetooth

- <https://play.google.com/store/apps/details?id=nextprototypes.BTSerialController>



Kit Smart Car 2WD Plus



DrakerDG 28-02-2022



Kit_Smart_Car_2WD_Plus_V2.ino

