Robot panda

Kit de robot Panda 4 DOF
https://es.aliexpress.com/item/4000443267348.html?spm=a219c.12057483.0.0.422178f8xLFDQh

Lista de componentes:

Tablero 1pcs Nano V3.0 CH340
4 piezas SG90 Servo
Tablero de extensión Nano 1pcs
1 piezas MINI Cable
1 piezas 4PIN F-F Dupont Wire
Módulo Bluetooth 1pcs
1 Uds 18650 celda de batería
Sensor ultrasónico 1pcs
Destornillador 1pcs
Llave 1pcs
Tablero de acrílico 1set
Kit de tornillos 1set
Tutorial en CD 1pcs
Cinturón de 3 piezas
Alfombrilla de goma 4pcs

Control con placa ESPDUINO-32.

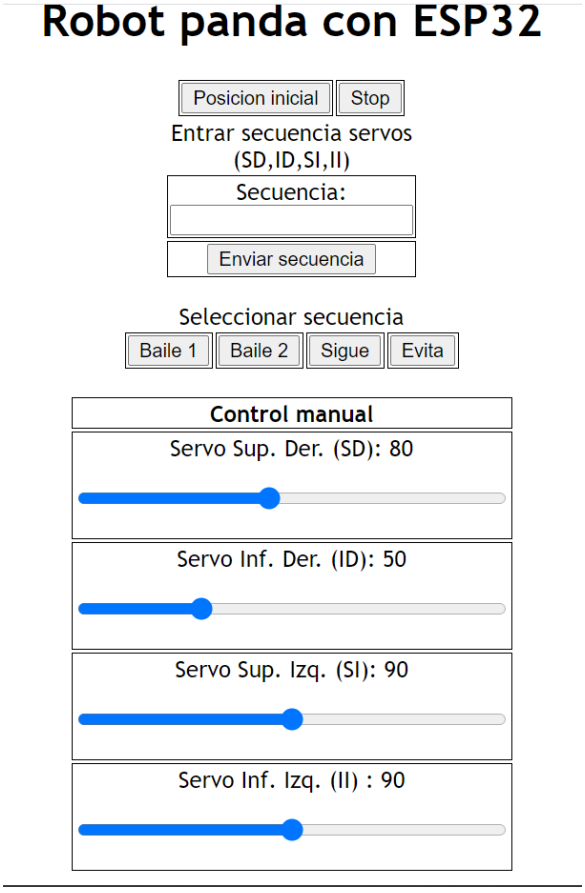
Vamos a utilizar para el control del robot la placa ESPDUINO-32 y el Sensor Shield 5.0 (para facilitar la conexión, se puede controlar solo con ESPDUINO-32).

Con ayuda de esta placa podemos controlar el robot por Bluetooht y por Wifi, ya que lo lleva incluido.

Asignamos los pines:

Función	Original (Arduino nano)	ESPDUINO-32
Servo Superior derecho (RU)	3	IO25
Servo Inferior derecho (RL)	5	IO16
Servo Superior Izquierdo (LU)	6	IO27
Servo Inferior Izquierdo (LL)	9	IO13
Sensor ultrasonidos: Echo	A0	I02 (E)
Sensor ultrasonidos: Trig	A1	IO4

Aplicación WEB



1.- Botones:

- Lleva servos a posición inicial.
- Detiene robot (pendiente interrupciones).

2.- Seleccionamos secuencia de bailes mediante botones.

- Sigue con ayuda del sensor de ultrasonidos.
- Evita con ayuda del sensor de ultrasonidos.

3.- Permite entrar secuencia (valores en grados de los 4 servos separador por comas).

4.- Deslizadores para mover cada servo.

Scrips:

Utilizamos 3 archivos:

ESP32_RobotPanda.ino

```
/*********
 * ESP32_RobotPanda
 * jarp 2020
 * Info:  https://randomnerdtutorials.com/esp32-servo-motor-web-server-arduino-ide/   Rui Santos
 * Librería: https://github.com/RoboticsBrno/ServoESP32

*********/
#include "baile.h"

void setup() {

   InicioSetup();
}

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

baile.cpp

```
#include <Arduino.h>
#include <WiFi.h>
#include <Servo.h>

Servo servos[4];

//Pines GPIO recomendados para la ESP32:2,4,12-19,21-23,25-27,32-33
static const int servosPins[4] = {25, 16, 27, 13};
int pos = 0;      // Posición en grados
const int array_cal[4] = {80,50,90,90};    // Define el angulo inicial de los servos (SD, ID, SI, II ) para
ajustar el montaje

// Variable para almacenar la solicitud HTTP
String header;

// Decodificar el valor HTTP GET
String valueString0 = String(5);
String valueString1 = String(5);
String valueString2 = String(5);
String valueString3 = String(5);
String ValorSecuencia = String(20);
int pos1 = 0;
int pos2 = 0;

// Tiempo actual
unsigned long currentTime = millis();
```

```
// Tiempo anterior
unsigned long previousTime = 0;
// Define tiempo timeouten milisegundos (examplo: 2000ms = 2s)
const long timeoutTime = 2000;
// Actualizar valores de red
  const char * ssid = "**********";
  const char * password = "**********";
// Set web server port number to 80
WiFiServer server(80);

int Echo = 2;  // 36
int Trig =4;  //
int Distance = 0;

const int delay_atras = 300, delay_adelante = 450;
//int vel_Dance1 = 30,    vel_Dance2 = 25;
int delay_Dance1 = 100, delay_Dance2 = 550, delay_Dance4 = 400;


const int num_dance1 = 10;
const int array_dance1[num_dance1][4] =
{
//slide to the left 0-4
    {0,-20,0,0},
    {0,-40,0,20},
    {0,-20,0,40},
    {0,0,0,20},
    {0,0,0,0},

//slide to the right 5-9
    {0,0,0,20},
    {0,-20,0,40},
    {0,-40,0,20},
    {0,-20,0,0},
    {0,0,0,0},
};

const int num_dance2 = 32;
const int array_dance2[num_dance2][4] =
{
//left foot support 0-15
    {20,0,40,0},
    {20,-30,40,-30},
    {20,-30,10,-30},
    {20,-30,40,-30},
    {20,-30,10,-30},

    {20,-30,40,-30},
    {20,0,40,-30},
    {20,80,40,-30},
    {20,0,40,-30},
    {20,-80,40,-30},
    {20,0,40,-30},
    {20,80,40,-30},
    {20,0,40,-30},
    {20,-30,40,-30},
```

```
    {20,0,40,0},
    {0,0,0,0},

//right foot support 16-31
    {-40,0,-20,0},
    {-40,40,-20,30},
    {-20,40,-20,30},
    {-40,40,-20,30},
    {-20,40,-20,30},

    {-40,40,-20,30},
    {-40,40,-20,0},
    {-40,40,-20,-80},
    {-40,40,-20,0},
    {-40,40,-20,80},
    {-40,40,-20,0},
    {-40,40,-20,-80},
    {-40,40,-20,0},
    {-40,40,-20,30},
    {-40,0,-20,0},
    {0,0,0,0},
};

const int num_dance4 = 20;      //In-situ ups and downs, increasing in angle
const int array_dance4[num_dance4][4] =
{
    {0,-20,0,20},
    {0,0,0,0},
    {0,-20,0,20},
    {0,0,0,0},
    {0,-20,0,20},
    {0,0,0,0},
    {0,-20,0,20},
    {0,0,0,0},

    {0,-50,0,50},
    {0,0,0,0},
    {0,-50,0,50},
    {0,0,0,0},
    {0,-50,0,50},
    {0,0,0,0},
    {0,-50,0,50},
    {0,0,0,0},

    {0,-40,0,40},
    {0,-50,0,50},
    {0,-60,0,60},
    {0,0,0,0,},
};

const int num1 = 8;
const int array_forward[num1][4] =
{

    {0,-40,0,-20},        //forward
    {30,-40,30,-20},
```

```
    {30,0,30,0},
    {0,20,0,40},
    {-30,20,-30,40},
    {-30,0,-30,0},
    {-15,0,-15,0},
    {0,0,0,0},
};

const int num2 = 8;
const int array_back[num2][4] =
{

    {0,-40,0,-20},        //Step-back
    {-30,-40,-30,-20},
    {-30,0,-30,0},
    {0,20,0,40},
    {30,20,30,40},
    {30,0,30,0},
    {15,0,15,0},
    {0,0,0,0},
};

void DireccionIP(){
IPAddress local_IP(192, 168, 1, 15);// Dirección  IP fija
IPAddress gateway(192, 168, 1, 1); // IP Gateway
IPAddress subnet(255, 255, 255, 0);//Mascara de red
//IPAddress primaryDNS(8, 8, 8, 8);   //optional
//IPAddress secondaryDNS(8, 8, 4, 4); //optional
// Configura Dirección IP fija
  if (!WiFi.config(local_IP, gateway, subnet)) {
    Serial.println("STA Error a configurar IP");
  }
  Serial.println("Ok Ip fija");
}


void ObtenerMac(){
  uint64_t chipid;
chipid=ESP.getEfuseMac();//The chip ID is essentially its MAC address(length: 6 bytes).
Serial.printf("ESP32 Chip ID = %04X",(uint16_t)(chipid>>32));//print High 2 bytes
Serial.printf("%08X\n",(uint32_t)chipid);//print Low 4bytes.
}

void Slide_2_Left(int times){ //desliza izquierda
    for(int time1 = 0; time1 < times; time1++) {
        for(int z=0; z<5; z++) {
            servos[0].write (array_cal[0] + array_dance1[z][0]);
            servos[1].write (array_cal[1] + array_dance1[z][1] );
            servos[2].write (array_cal[2] + array_dance1[z][2] );
            servos[3].write (array_cal[3] + array_dance1[z][3] );
            delay(delay_Dance1);
        }
    }
}

void Slide_2_Right(int times){ //desliza derecha
```

```
    for(int time1 = 0; time1 < times; time1++) {
        for(int z=5; z<10; z++) {
            servos[0].write (array_cal[0] + array_dance1[z][0]);
            servos[1].write (array_cal[1] + array_dance1[z][1] );
            servos[2].write (array_cal[2] + array_dance1[z][2] );
            servos[3].write (array_cal[3] + array_dance1[z][3] );
            delay(delay_Dance1);
        }
    }
}

void Left_Foot_Support(){   //Soporte para pie izquierdo
    for(int z=0; z<16; z++) { //z<12
        if ( z > 5 && z < 14) {   //z(1,10)
            delay_Dance2 = 200;
        }
        else {
        delay_Dance2 = 750;
        }

        servos[0].write (array_cal[0] + array_dance2[z][0] );
        servos[1].write (array_cal[1] + array_dance2[z][1] );
        servos[2].write (array_cal[2] + array_dance2[z][2]);
        servos[3].write (array_cal[3] + array_dance2[z][3] );
        delay(delay_Dance2);
    }
}

void Right_Foot_Support(){  //Soporte para pie derecho
    for(int z=16; z<32; z++) { //z<24
        if ( z > 21 && z < 30) {   //z(13,22)
            delay_Dance2 = 200;
        }
        else {
        delay_Dance2 = 750;
        }

        servos[0].write (array_cal[0] + array_dance2[z][0] );
        servos[1].write (array_cal[1] + array_dance2[z][1] );
        servos[2].write (array_cal[2] + array_dance2[z][2] );
        servos[3].write (array_cal[3] + array_dance2[z][3] );
        delay(delay_Dance2);
    }
}

void Ultrasonidos(){
  digitalWrite(Trig, LOW);   // Give the trigger pin low level 2μs
  delayMicroseconds(2);
  digitalWrite(Trig, HIGH);  // Give the trigger pin a high level of 10μs, here at least 10μs
  delayMicroseconds(10);
  digitalWrite(Trig, LOW);    // Continue to low voltage to the trigger pin
  float Fdistance = pulseIn(Echo, HIGH);  // Read high time (unit: microsecond)
  Fdistance= Fdistance/58;       //Why divide by 58 equals centimeter, Y m = (X seconds * 344) / 2
  // X seconds = ( 2 * Y meters) / 344 = = "X seconds = 0.0058 * Y meters = = "cm = microseconds /
58
  Serial.print("Distance:");      //Output distance (unit: cm)
```

```
  Serial.println(Fdistance);        //Display distance
  Distance = Fdistance;
}

void baile1(){
   Slide_2_Left(2);
   Left_Foot_Support();
   Slide_2_Right(2);
   Right_Foot_Support();
}

void baile2(){
   for(int z=0; z<num_dance4; z++) {
      if ( z > 17) {
         delay_Dance4 = 1500;
         }
      else {
         delay_Dance4 = 400;
         }
      servos[0].write (array_cal[0] + array_dance4[z][0] );
      servos[1].write (array_cal[1] + array_dance4[z][1] );
      servos[2].write (array_cal[2] + array_dance4[z][2] );
      servos[3].write (array_cal[3] + array_dance4[z][3] );
      delay(delay_Dance4);
}
}//<-



void adelante(){
   Serial.println("adelante");
  for(int z=0; z<1; z++) {
   for(int x=0; x<num1; x++) {

      servos[0].write  (array_cal[0] + array_forward[x][0]);
      servos[1].write (array_cal[1] + array_forward[x][1]);
      servos[2].write (array_cal[2] + array_forward[x][2]);
      servos[3].write (array_cal[3] + array_forward[x][3]);
      delay(delay_adelante);
   }

}
}

void atras(){
   Serial.println("atras");
  for(int z=0; z<3; z++) {
  for(int y=0; y<num2; y++) {
      servos[0].write  (array_cal[0] +  array_back[y][0]);
      servos[1].write (array_cal[1] +  array_back[y][1]);
       servos[2].write (array_cal[2] +  array_back[y][2]);
       servos[3].write (array_cal[3] +  array_back[y][3]);
     delay(delay_atras);
                    }
                   }
}
```

```
void sigue(){
   Ultrasonidos();
    if(Distance>20){
     adelante();
     Ultrasonidos();
     sigue();
    }

}//<-

void evita(){
  Ultrasonidos();
  if(Distance > 5){
       if(Distance >20){
          adelante();
        }else{
          atras();
        }
        evita(); // recursivo  ver como parar
  }
}//<-

void InicializaServos(){
  for(int i = 0; i < 4; ++i) {
        if(!servos[i].attach(servosPins[i])) {
            Serial.print("Servo ");
            Serial.print(i);
            Serial.println("attach error");
        }
    }
    Serial.println("Servos inicializados");
}

void PosicionInicial(){
Serial.println(array_cal[0]);
  servos[0].write(array_cal[0]); // Servo superior derecho a posición inicial de ajuste
  servos[1].write(array_cal[1]); // Servo inferior derecho a posición inicial de ajuste
  servos[2].write(array_cal[2]); // Servo superior izquierdo a posición inicial de ajuste
  servos[3].write(array_cal[3]); // Servo inferior izquierdo a posición inicial de ajuste
}

void InicioSetup(){
   Serial.begin(115200);
  pinMode(Echo, INPUT);    // Define ultrasonic input pin
  pinMode(Trig, OUTPUT);   // Define ultrasonic output pin
    InicializaServos();
    ObtenerMac();
    DireccionIP();
  // Connect to Wi-Fi network with SSID and password
  Serial.print("Conectado a ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
```

```
  }

  // Print local IP address and start web server
  Serial.println("");
  Serial.println("WiFi connected.");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  server.begin();
  PosicionInicial();
}

// RESET por botón
void(* resetFunc) (void) = 0;//declare reset function at address 0

void ControlWeb(){
  WiFiClient client = server.available();   // Escuchar a los clientes entrantes

 if (client) {                             // Si un nuevo cliente se conecta,
    currentTime = millis();
    previousTime = currentTime;
    Serial.println("New Client.");          // print a message out in the serial port
    String currentLine = "";                // hacer una cadena para contener los datos entrantes del
cliente
    while (client.connected() && currentTime - previousTime <= timeoutTime) { // Bucle mientras el
cliente está conectado
      currentTime = millis();
      if (client.available()) {             // Si hay bytes para leer del cliente,
        char c = client.read();             // leer un byte, luego
        Serial.write(c);                    // imprímalo en el monitor de serie
        header += c;
        if (c == '\n') {                    //si el byte es un carácter de nueva línea
          // si la línea actual está en blanco, tiene dos caracteres de nueva línea seguidos.
          // ese es el final de la solicitud HTTP del cliente, así que envíe una respuesta:
          if (currentLine.length() == 0) {
            // Los encabezados HTTP siempre comienzan con un código de respuesta (por ejemplo,
HTTP / 1.1 200 OK)
            // y un tipo de contenido para que el cliente sepa lo que viene, luego una línea en blanco:

            client.println("HTTP/1.1 200 OK");
            client.println("Content-type:text/html");
            client.println("Connection: close");
            client.println();

            // Display the HTML web page
            client.println("<!DOCTYPE html><html>");
            client.println("<head><meta name=\"viewport\" content=\"width=device-width,
initial-scale=1\">");
            client.println("<link rel=\"icon\" href=\"data:,\">");
            // CSS to style the on/off buttons
            // Feel free to change the background-color and font-size attributes to fit your preferences
            client.println("<style>body { text-align: center; font-family: \"Trebuchet MS\", Arial;
margin-left:auto; margin-right:auto;}");
            client.println(".slider { width: 300px; }");
             client.println(".center table, th, td {border: 1px solid black;text-align: center;}</style>");
            client.println("<script
src=\"https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js\"></script>");
```

```
          client.println("<center>");

          client.println("</head><body><h1>Robot panda con ESP32 </h1>");

          client.println("<table class='center'>");
          client.println("<tr><td><a href='/Posi0/'><button class='button'>Posicion
inicial</button></a></td>");
          client.println("<td><a href='/Stop/'><button
class='button'>Stop</button></a></td></tr></table>");

          client.println("<table class='center'>");
          client.println("<B><caption>Entrar secuencia servos (SD,ID,SI,II)</caption></B>");
          client.println("<tr><td> <form action='/'><label for='secuencia'>Secuencia:</label><br>");
          client.println(" <input type='text' name='secuencia'></a></td>");
          client.println("<tr><td> <input type='submit'  value='Enviar secuencia'></form></a></td>");
          client.println("</tr></table>");

          client.println("<p><table class='center'>");
          client.println("<B><caption>Seleccionar secuencia</caption></B>");
          client.println("<tr><td><a href='/Posi1/'><button class='button'>Baile 1</button></a></td>");
          client.println("<td><a href='/Posi2/'><button class='button'>Baile 2</button></a></td>");
          client.println("<td><a href='/Posi3/'><button class='button'>Sigue</button></a></td>");
          client.println("<td><a href='/Posi4/'><button class='button'>Evita</button></a></td>");
          client.println("</tr></table></p>");

          client.println("<table class='center'>");
          client.println("<th>Control manual </th>");
          client.println("<tr><td>Servo Sup. Der. (SD): <span id='servoPos0'></span>");
          client.println("<p><input type='range' min='0' max='180' value='" + String(array_cal[0]) +"'
class='slider' id='servoSlider0' onchange='servo0(this.value)' value0='"+valueString0+"'/></p>");
          client.println("</td></tr><tr><td>Servo Inf. Der. (ID): <span id='servoPos1'></span>");
          client.println("<p><input type='range' min='0' max='180' value='" + String(array_cal[1]) +"'
class='slider' id='servoSlider1' onchange='servo1(this.value)' value1='"+valueString1+"'/></p>");
          client.println("</td></tr><tr><td>Servo Sup. Izq. (SI): <span id='servoPos2'></span>");
          client.println("<p><input type='range' min='0' max='180' value='" + String(array_cal[2]) +"'
class='slider' id='servoSlider2' onchange='servo2(this.value)' value2='"+valueString2+"'/></p>");
          client.println("</td></tr><tr><td>Servo Inf.  Izq. (II) : <span id='servoPos3'></span>");
          client.println("<p><input type='range' min='0' max='180' value='" + String(array_cal[3]) +"'
class='slider' id='servoSlider3' onchange='servo3(this.value)' value3='"+valueString3+"'/></p>");
           client.println("</td></tr></table>");
          client.println("<script>var slider0 = document.getElementById('servoSlider0');");
          client.println("var servoP0 = document.getElementById('servoPos0'); servoP0.innerHTML =
slider0.value;");
          client.println("slider0.oninput = function() { slider0.value = this.value; servoP0.innerHTML =
this.value; }");
          client.println("$.ajaxSetup({timeout:1000}); function servo0(pos) { ");
          client.println("$.get('/?value0=' + pos + '&'); {Connection: close};}</script>");

          client.println("<script>var slider1 = document.getElementById('servoSlider1');");
          client.println("var servoP1 = document.getElementById('servoPos1'); servoP1.innerHTML =
slider1.value;");
          client.println("slider1.oninput = function() { slider1.value = this.value; servoP1.innerHTML =
this.value; }");
          client.println("$.ajaxSetup({timeout:1000}); function servo1(pos) { ");
          client.println("$.get('/?value1=' + pos + '&'); {Connection: close};}</script>");
```

```
          client.println("<script>var slider2 = document.getElementById('servoSlider2');");
          client.println("var servoP2 = document.getElementById('servoPos2'); servoP2.innerHTML =
slider2.value;");
          client.println("slider2.oninput = function() { slider2.value = this.value; servoP2.innerHTML =
this.value; }");
          client.println("$.ajaxSetup({timeout:1000}); function servo2(pos) { ");
          client.println("$.get('/?value2=' + pos + '&'); {Connection: close};}</script>");

          client.println("<script>var slider3 = document.getElementById('servoSlider3');");
          client.println("var servoP3 = document.getElementById('servoPos3'); servoP3.innerHTML =
slider3.value;");
          client.println("slider3.oninput = function() { slider3.value = this.value; servoP3.innerHTML =
this.value; }");
          client.println("$.ajaxSetup({timeout:1000}); function servo3(pos) { ");
          client.println("$.get('/?value3=' + pos + '&'); {Connection: close};}</script>");

          client.println("</body></html>");
          Serial.println("Recibe:" + header); //GET /?value=180& HTTP/1.1

          if(header.indexOf("GET /Stop/")>=0) {
             resetFunc(); //call reset
          }


          if(header.indexOf("GET /Posi0/")>=0) {
            PosicionInicial();

          }

          if(header.indexOf("GET /Posi1/")>=0) {
             baile1();
             Serial.println("baile1");
          }
          if(header.indexOf("GET /Posi2/")>=0) {
           baile2();
           Serial.println("baile2");
          }
          if(header.indexOf("GET /Posi3/")>=0) {
           sigue();

           Serial.println("sigue");
          }
          if(header.indexOf("GET /Posi4/")>=0) {
            evita();
            Serial.println("evita");
          }

          if(header.indexOf("secuencia")!= -1) {
            // ejecuta secuencia
            header.replace("GET /?", "");    // elimina valores de la cabecera

            header.replace("%2C", ",");          // Para que las , no salgan con ,
            header.replace(" HTTP/1.1", " ");
            pos1 = header.indexOf('=');
            pos2 = header.indexOf(' ');
            ValorSecuencia= header.substring(pos1+1, pos2);
```

```
  Serial.println("inicial:" + ValorSecuencia);
  //separamos caracteres
  char  secuencia_buffer[20] ;
    ValorSecuencia.toCharArray(secuencia_buffer, 20);
    char *p = secuencia_buffer; //puntero
    char *str;
    String info1;
    int i=0;
    while ((str = strtok_r(p, ",", &p)) != NULL) {// separados por comas
     // if (str.toInt()>180) {str=180;}   // para evitar valores mayores
     servos[i].write(int(str));//0:SD, 1:SI, 2:ID, 3: II
     Serial.println(String(str));
       i++;
    } // while


  Serial.println("separada:" + ValorSecuencia);
   }


if(header.indexOf("GET /?value0=")>=0) {
  pos1 = header.indexOf('=');
  pos2 = header.indexOf('&');
  valueString0 = header.substring(pos1+1, pos2);

  //Rotate the servo
  servos[0].write(valueString0.toInt());
  Serial.println(valueString0);
}

  if(header.indexOf("GET /?value1=")>=0) {
  pos1 = header.indexOf('=');
  pos2 = header.indexOf('&');
  valueString1 = header.substring(pos1+1, pos2);

  //Rotate the servo
  servos[1].write(valueString1.toInt());
  Serial.println(valueString1);
}

if(header.indexOf("GET /?value2=")>=0) {
  pos1 = header.indexOf('=');
  pos2 = header.indexOf('&');
  valueString2 = header.substring(pos1+1, pos2);

  //Rotate the servo
  servos[2].write(valueString2.toInt());
  Serial.println(valueString2);
}

 if(header.indexOf("GET /?value3=")>=0) {
  pos1 = header.indexOf('=');
  pos2 = header.indexOf('&');
  valueString3 = header.substring(pos1+1, pos2);

  //Rotate the servo
```

```
            servos[3].write(valueString3.toInt());
            Serial.println(valueString3);
          }
          // La respuesta HTTP termina con otra línea en blanco
          client.println();
          // Salir del bucle while
          break;
        } else { // si tiene una nueva línea, borre currentLine
          currentLine = "";
        }
      } else if (c != '\r') {  // si tiene algo más que un carácter de retorno de carro,
        currentLine += c;      // agréguelo al final de la línea actual
      }
}
   }
  } //(client)

   // Borrar la variable de encabezado
   header = "";
   // Cerrar la conexión
   client.stop();
//   Serial.println("Client desconectado.");
//   Serial.println("");

}
```

baile.h

```
#ifndef baile_h
#define baile_h


void DireccionIP();
void ObtenerMac();
void InicializaServos();
void PosicionInicial();

void Slide_2_Left(int times);
void Slide_2_Right(int times);
void Left_Foot_Support();
void Right_Foot_Support();
void Dancing1();
void InicioSetup();
void ControlWeb();
void secuencia(String mueve1);
void Ultrasonidos();
#endif
```

Asigna valores servos

//Pines GPIO recomendados para la ESP32:2,4,12-19,21-23,25-27,32-33

```
static const int servosPins[4] = {25, 16, 27, 13};
int pos = 0;      // Posición en grados
 // Define el angulo inicial de los servos (SD, ID, SI, II ) para ajustar el montaje
const int array_cal[4] = {80,50,90,90};  Servo servos[4];
```

Variables de la WEB

```
// Decodificar el valor HTTP GET
String valueString0 = String(5);
String valueString1 = String(5);
String valueString2 = String(5);
String valueString3 = String(5);
int pos1 = 0;
int pos2 = 0;
```

Variables de tiempo para conexión

```
// Tiempo actual
unsigned long currentTime = millis();
// Tiempo anterior
unsigned long previousTime = 0;
// Define tiempo timeouten milisegundos (examplo: 2000ms = 2s)
const long timeoutTime = 2000;
```

Asigna IP fija

```
void DireccionIP(){
IPAddress local_IP(192, 168, 1, 15);// Dirección  IP fija
IPAddress gateway(192, 168, 1, 1); // IP Gateway
IPAddress subnet(255, 255, 255, 0);//Mascara de red
//IPAddress primaryDNS(8, 8, 8, 8);   //optional
//IPAddress secondaryDNS(8, 8, 4, 4); //optional
// Configura Dirección IP fija
  if (!WiFi.config(local_IP, gateway, subnet)) {
    Serial.println("STA Error a configurar IP");
  }
}
```

Obtiene MAC placa ESP32

```
void ObtenerMac(){
  uint64_t chipid;
chipid=ESP.getEfuseMac();//The chip ID is essentially its MAC address(length: 6 bytes).
Serial.printf("ESP32 Chip ID = %04X",(uint16_t)(chipid>>32));//print High 2 bytes
Serial.printf("%08X\n",(uint32_t)chipid);//print Low 4bytes.
}
```

Inicializa Servos

```
void InicializaServos(){
  for(int i = 0; i < 4; ++i) {
        if(!servos[i].attach(servosPins[i])) {
            Serial.print("Servo ");
            Serial.print(i);
            Serial.println("attach error");
        }
    }
    Serial.println("Servos inicializados");
}
```

Mueve servos a la posición inicial

```
void PosicionInicial(){
Serial.println(array_cal[0]);
  servos[0].write(array_cal[0]); // Servo superior derecho a posición inicial de ajuste
  servos[1].write(array_cal[1]); // Servo inferior derecho a posición inicial de ajuste
  servos[2].write(array_cal[2]); // Servo superior izquierdo a posición inicial de ajuste
  servos[3].write(array_cal[3]); // Servo inferior izquierdo a posición inicial de ajuste
}
```

Calcula distancia sensor Ultrasonidos

```
void Ultrasonidos(){
  digitalWrite(Trig, LOW);   // Give the trigger pin low level 2μs
  delayMicroseconds(2);
  digitalWrite(Trig, HIGH);  // Give the trigger pin a high level of 10μs, here at least 10μs
  delayMicroseconds(10);
  digitalWrite(Trig, LOW);    // Continue to low voltage to the trigger pin
  float Fdistance = pulseIn(Echo, HIGH);  // Read high time (unit: microsecond)
  Fdistance= Fdistance/58;       //Why divide by 58 equals centimeter, Y m = (X seconds * 344) / 2
  // X seconds = ( 2 * Y meters) / 344 = = "X seconds = 0.0058 * Y meters = = "cm = microseconds /
58
  Serial.print("Distance:");      //Output distance (unit: cm)
  Serial.println(Fdistance);        //Display distance
  Distance = Fdistance;
}
```

RESET por botón web

```
void(* resetFunc) (void) = 0;//declare reset function at address 0
```

Movimientos del robot

```
const int num1 = 8;
const int array_forward[num1][4] =
{
```

```
    {0,-40,0,-20},        //forward
    {30,-40,30,-20},
    {30,0,30,0},
    {0,20,0,40},
    {-30,20,-30,40},
    {-30,0,-30,0},
    {-15,0,-15,0},
    {0,0,0,0},
};
```

Baile

```
void baile2(){
   for(int z=0; z<num_dance4; z++) {
      if ( z > 17) {
         delay_Dance4 = 1500;
         }
      else {
         delay_Dance4 = 400;
         }
      servos[0].write (array_cal[0] + array_dance4[z][0] );
      servos[1].write (array_cal[1] + array_dance4[z][1] );
      servos[2].write (array_cal[2] + array_dance4[z][2] );
      servos[3].write (array_cal[3] + array_dance4[z][3] );
      delay(delay_Dance4);
}
}//<-
```

Enlaces:

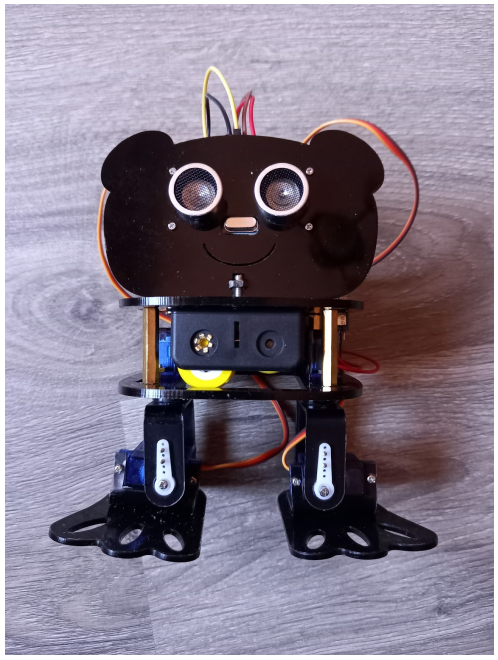
Librería: https://github.com/RoboticsBrno/ServoESP32

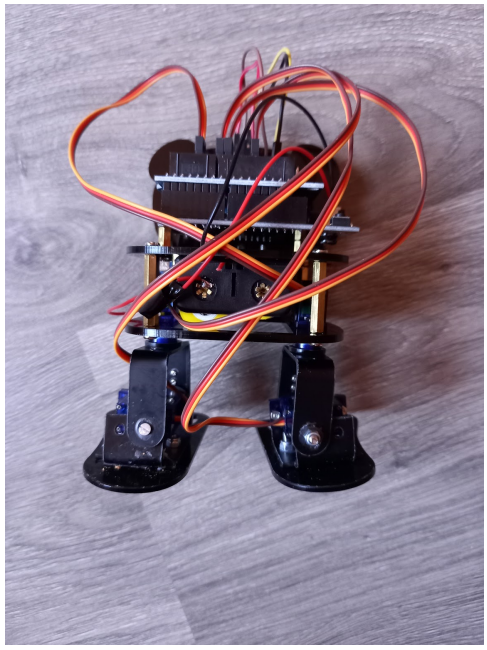
Control servo Wifi https://randomnerdtutorials.com/esp32-servo-motor-web-server-arduino-ide/

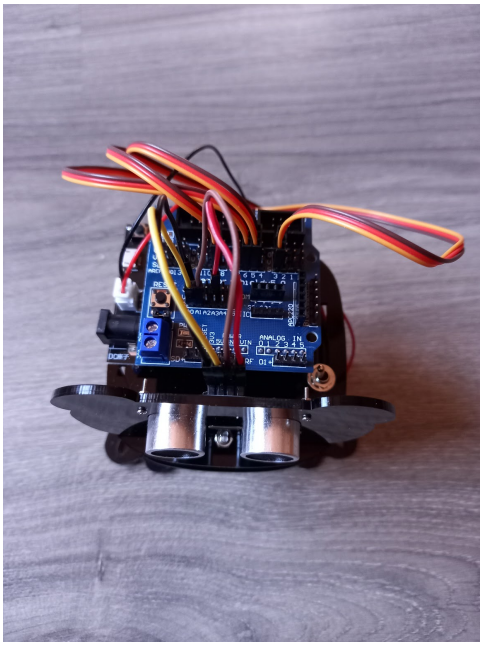
Mejoras:

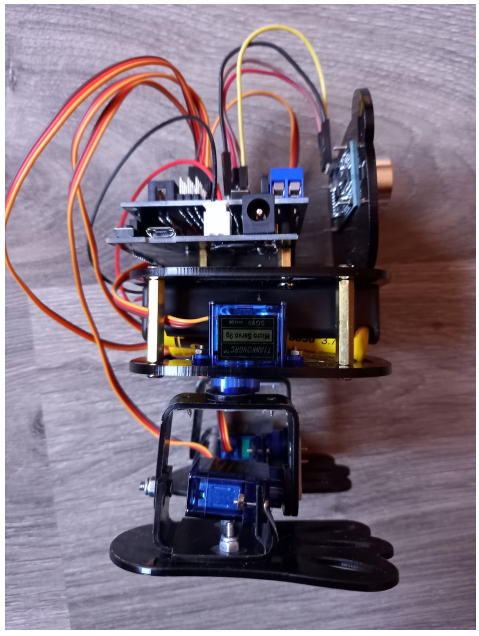
- Control mediante Bluetooht.
- Entrar secuencias de baile y Almacenar secuencias en SDCard.
- Detener baile mediante interrupciones.

Fotos:





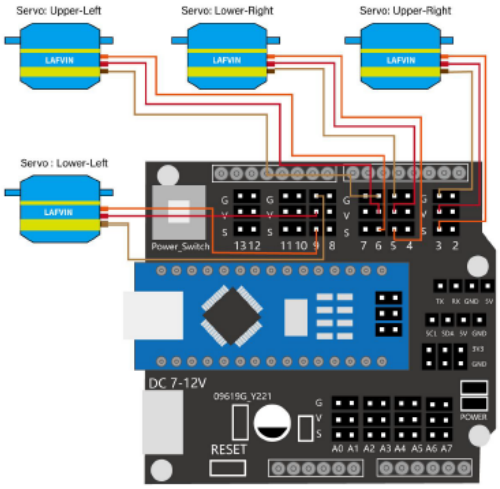




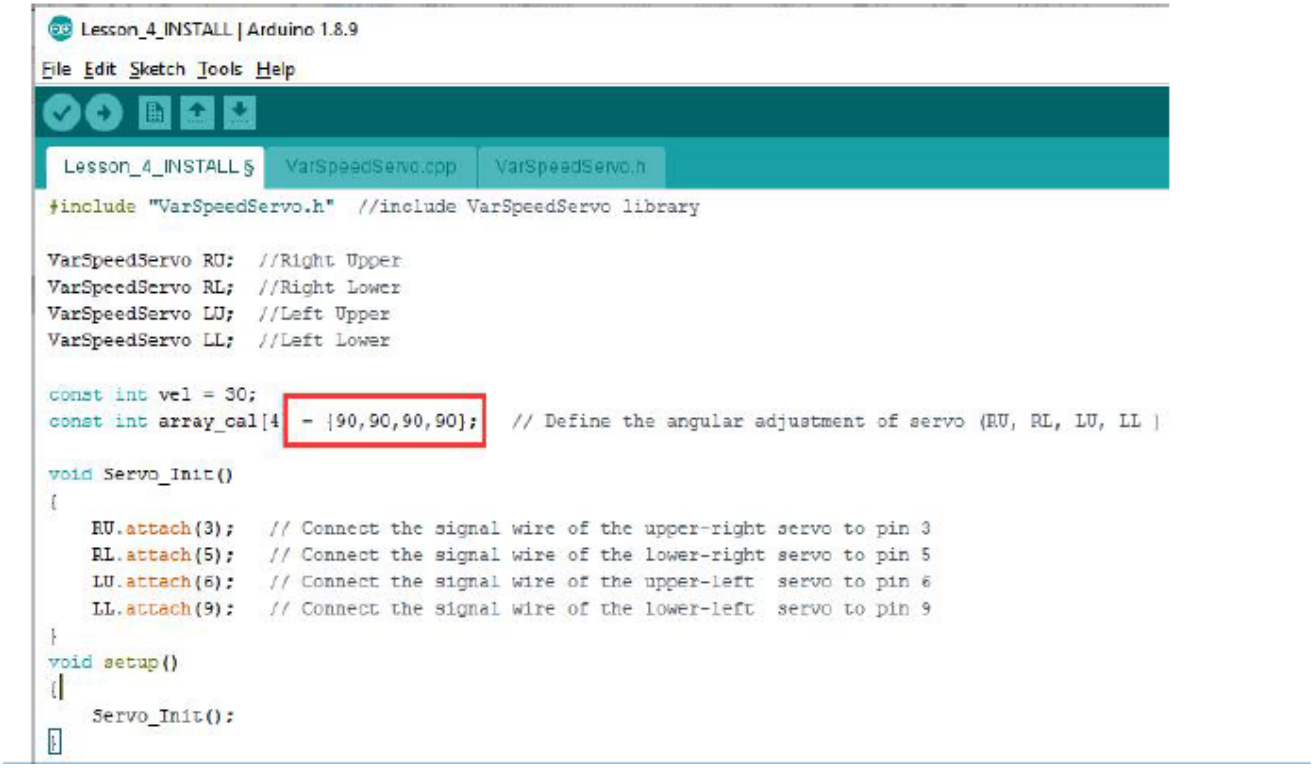
Control original Arduimo nano

Método de instalación

Antes de instalar el robot, debemos establecer la posición inicial de los cuatro servomotores. Una vez establecido el ángulo, puede instalarlo según la dirección vertical o paralela indicada por la imagen El método de calibración del ángulo del servomotor es el siguiente:
Primero, instale la placa Arduino Nano en la placa de expansión y luego conecte servo1 / servo2 / servo3 / servo4 a los pines 3/5/6/9 .



Conecte la placa Nano y la computadora con un cable USB. Abra el programa Lesson_4_INSTALL y cargue el programa en el tablero. Presione el botón de reinicio, el servomotor se moverá a la posición de 90 grados requerida para la instalación. Mantenga 90 grados antes de la instalación, no gire el engranaje del servomotor. Ahora comencemos a instalar la sección del servo.



Ahora los cuatro servomotores se giran a la posición de 90 grados. A continuación, instale el servomotor de acuerdo con la dirección indicada en la imagen. Puede ser vertical o paralelo. A continuación, comenzamos a instalar el robot.

Los archivos **VarSpeedServo.ccp** y **VarSpeedServo.h** contienen la libreria para el control de los servos.

Step 03

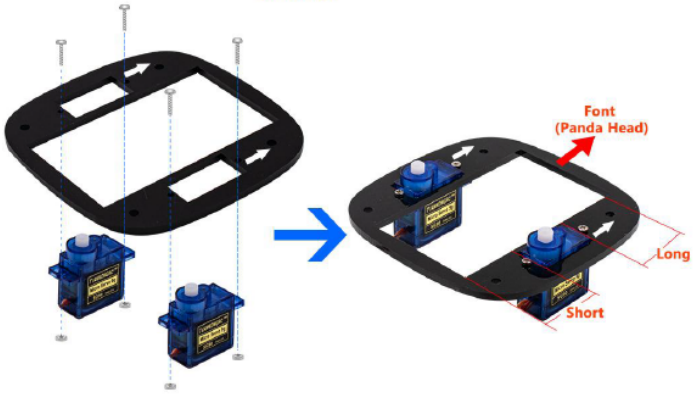
1. M1.5*5mm Self-tapping Screw

Step 04

1. M3*8mm Countersunk Screw
2. M3 Lock Nut

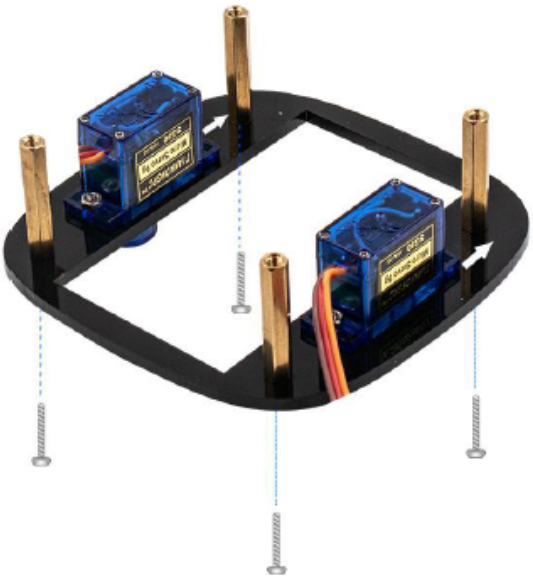
Step 05

1. M2*8mm Screw
2. M2 Nut



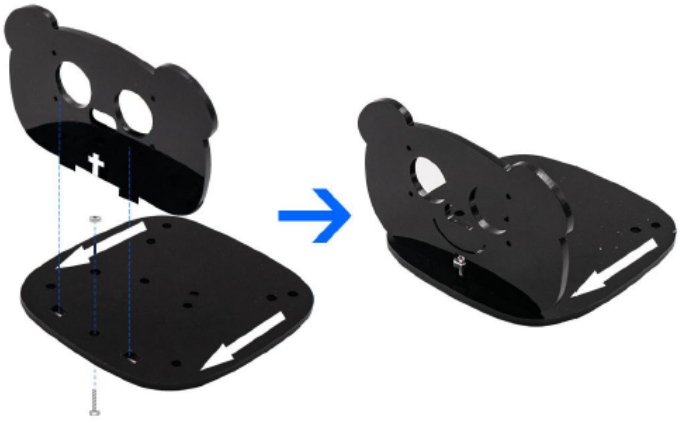
Step 06

1. M3*6mm Screw
2. M3*25mm Copper Standoff



Step 07

1. M3*8mm Countersunk Screw
2. M3 Nut

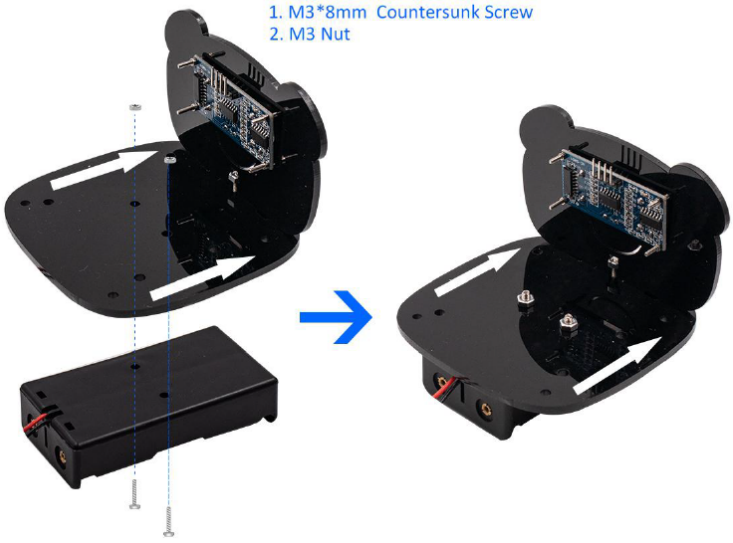


Step 08

1. M1.4*8mm Screw
2. M1.4 Nut

Step 09

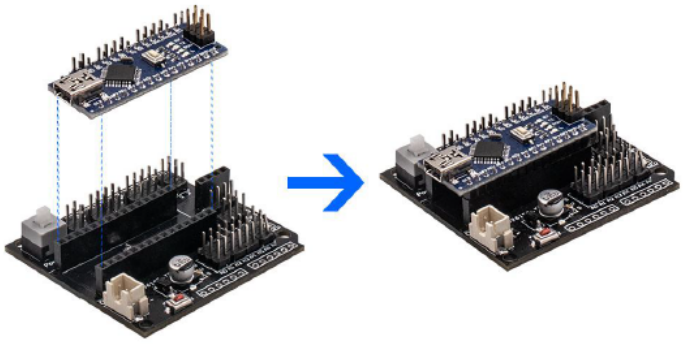
1. M3*8mm Countersunk Screw
2. M3 Nut



Step 10

1. M3*6mm Screw
2. M3*8mm Copper Standoff

Step 11

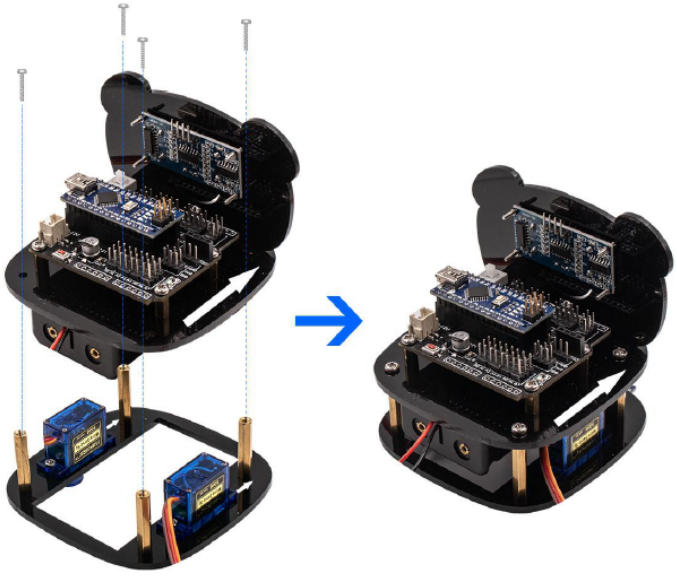


Step 12

1. M3*6mm Screw

Step 13

1. M3*6mm Screw

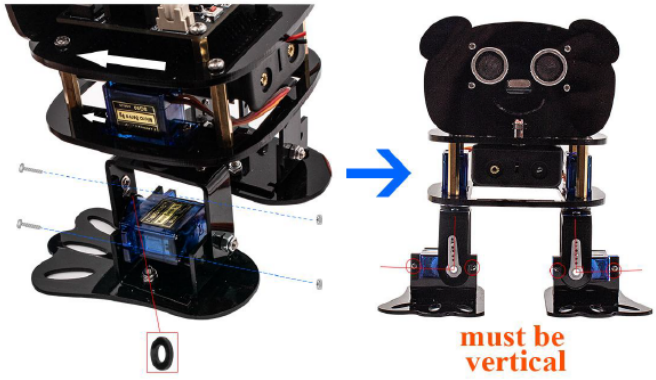


Step 14

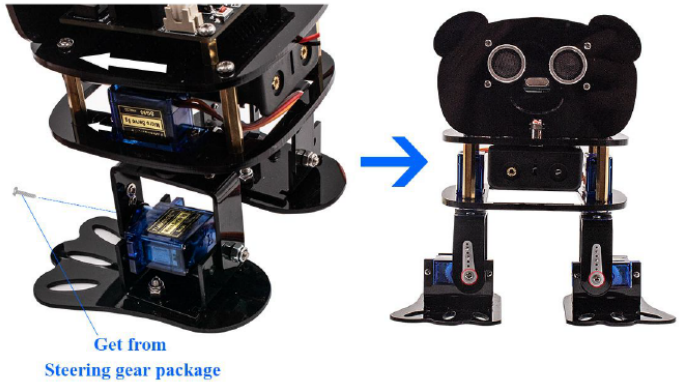
1. Gasket

Step 15

1. M2*8mm Screw
2. M2 Nut
3. Gasket



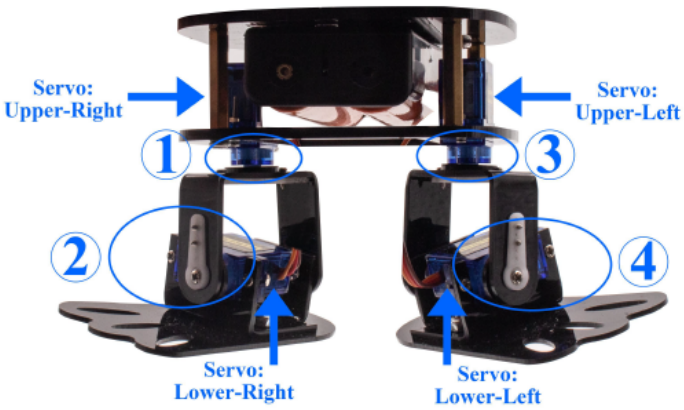
Step 16



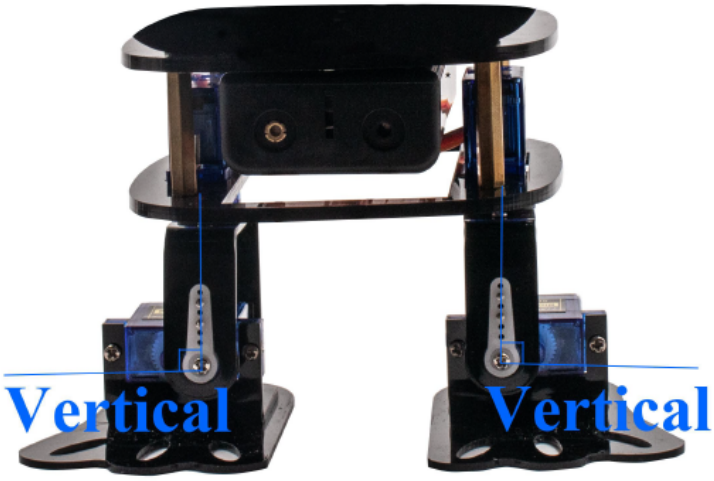
Lección 5 Posición inicial de calibración

Para hacer que su robot camine y baile más bellamente, después de la instalación, el robot debe calibrarse inicialmente.

Cuando lo instalemos, dejemos que el robot se mantenga erguido tanto como sea posible. Quizás la postura de la pierna del robot después de la instalación sea así.



Por lo tanto, necesitamos calibrar los servos del robot nuevamente en el programa de software. Al modificar el valor en el programa, controle el ángulo del mecanismo de dirección para que la postura de las piernas del robot sea la siguiente



Cómo calibrar la posición inicial

Primero abra el programa - .

Lesson_5_CALIBRATION

```
#include "VarSpeedServo.h"  //include VarSpeedServo library

VarSpeedServo RU;  //Right Upper
VarSpeedServo RL;  //Right Lower
VarSpeedServo LU;  //Left Upper
VarSpeedServo LL;  //Left Lower
```

```
const int vel = 30;
const int array_cal[4] = {81,80,90,92};   // Define the angular adjustment of servo (RU, RL, LU, LL )


void Servo_Init()
{
   RU.attach(3);   // Connect the signal wire of the upper-right servo to pin 3
   RL.attach(5);   // Connect the signal wire of the lower-right servo to pin 5
   LU.attach(6);   // Connect the signal wire of the upper-left  servo to pin 6
   LL.attach(9);   // Connect the signal wire of the lower-left  servo to pin 9
}

void setup()
{
   Servo_Init();
}

void loop()
{
   RU.slowmove (array_cal[0] , vel);  // Define the angle and speed of the upper-right servo.
   RL.slowmove (array_cal[1] , vel);
   LU.slowmove (array_cal[2] , vel);
   LL.slowmove (array_cal[3] , vel);
   delay(2000);
}
```

VarSpeedServo.ccp

```
/*
 Servo.cpp - Interrupt driven Servo library for Arduino using 16 bit timers- Version 2
 Copyright (c) 2009 Michael Margolis.  All right reserved.

 This library is free software; you can redistribute it and/or
 modify it under the terms of the GNU Lesser General Public
 License as published by the Free Software Foundation; either
 version 2.1 of the License, or (at your option) any later version.

 This library is distributed in the hope that it will be useful,
 but WITHOUT ANY WARRANTY; without even the implied warranty of
 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 Lesser General Public License for more details.

 You should have received a copy of the GNU Lesser General Public
 License along with this library; if not, write to the Free Software
 Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
 */

/*
  Function slowmove and supporting code added 2010 by Korman. Above limitations apply
  to all added code, except for the official maintainer of the Servo library. If he,
  and only he deems the enhancment a good idea to add to the official Servo library,
  he may add it without the requirement to name the author of the parts original to
  this version of the library.
*/

/*
  Updated 2013 by Philip van Allen (pva),
  -- updated for Arduino 1.0 +
  -- consolidated slowmove into the write command (while keeping slowmove() for compatibility
     with Korman's version)
  -- added wait parameter to allow write command to block until move is complete
  -- added sequence playing ability to asynchronously move the servo through a series of positions, must be called in a
loop

  A servo is activated by creating an instance of the Servo class passing the desired pin to the attach() method.
```

```
  The servos are pulsed in the background using the value most recently written using the write() method

  Note that analogWrite of PWM on pins associated with the timer are disabled when the first servo is attached.
  Timers are seized as needed in groups of 12 servos - 24 servos use two timers, 48 servos will use four.
  The sequence used to sieze timers is defined in timers.h

  The methods are:

   VarSpeedServo - Class for manipulating servo motors connected to Arduino pins.

   attach(pin )  - Attaches a servo motor to an i/o pin.
   attach(pin, min, max  ) - Attaches to a pin setting min and max values in microseconds
   default min is 544, max is 2400

   write(value)     - Sets the servo angle in degrees.  (invalid angle that is valid as pulse in microseconds is treated as
microseconds)
   write(value, speed) - speed varies the speed of the move to new position 0=full speed, 1-255 slower to faster
   write(value, speed, wait) - wait is a boolean that, if true, causes the function call to block until move is complete

   writeMicroseconds() - Sets the servo pulse width in microseconds
   read()      - Gets the last written servo pulse width as an angle between 0 and 180.
   readMicroseconds()  - Gets the last written servo pulse width in microseconds. (was read_us() in first release)
   attached()  - Returns true if there is a servo attached.
   detach()    - Stops an attached servos from pulsing its i/o pin.

   slowmove(value, speed) - The same as write(value, speed), retained for compatibility with Korman's version

   stop() - stops the servo at the current position

   sequencePlay(sequence, sequencePositions); // play a looping sequence starting at position 0
   sequencePlay(sequence, sequencePositions, loop, startPosition); // play sequence with number of positions, loop if
true, start at position
   sequenceStop(); // stop sequence at current position

 */

#include <avr/interrupt.h>
#include <Arduino.h> // updated from WProgram.h to Arduino.h for Arduino 1.0+, pva

#include "VarSpeedServo.h"

#define usToTicks(_us)    (( clockCyclesPerMicrosecond()* _us) / 8)     // converts microseconds to tick (assumes
prescale of 8)  // 12 Aug 2009
#define ticksToUs(_ticks) (( (unsigned)_ticks * 8)/ clockCyclesPerMicrosecond() ) // converts from ticks back to
microseconds


#define TRIM_DURATION       2                               // compensation ticks to trim adjust for digitalWrite delays // 12 August
2009

//#define NBR_TIMERS        (MAX_SERVOS / SERVOS_PER_TIMER)

static servo_t servos[MAX_SERVOS];                          // static array of servo structures
static volatile int8_t Channel[_Nbr_16timers ];             // counter for the servo being pulsed for each timer (or -1 if refresh
interval)

uint8_t ServoCount = 0;                                     // the total number of attached servos

// sequence vars

servoSequencePoint initSeq[] = {{0,100},{45,100}};

//sequence_t sequences[MAX_SEQUENCE];

// convenience macros
#define SERVO_INDEX_TO_TIMER(_servo_nbr) ((timer16_Sequence_t)(_servo_nbr / SERVOS_PER_TIMER)) //
returns the timer controlling this servo
#define SERVO_INDEX_TO_CHANNEL(_servo_nbr) (_servo_nbr % SERVOS_PER_TIMER)       // returns the index of
the servo on this timer
```

```
#define SERVO_INDEX(_timer,_channel)  ((_timer*SERVOS_PER_TIMER) + _channel)     // macro to access servo
index by timer and channel
#define SERVO(_timer,_channel)  (servos[SERVO_INDEX(_timer,_channel)])          // macro to access servo class by
timer and channel

#define SERVO_MIN() (MIN_PULSE_WIDTH - this->min * 4)  // minimum value in uS for this servo
#define SERVO_MAX() (MAX_PULSE_WIDTH - this->max * 4)  // maximum value in uS for this servo

/************ static functions common to all instances ***********************/

static inline void handle_interrupts(timer16_Sequence_t timer, volatile uint16_t *TCNTn, volatile uint16_t* OCRnA)
{
  if( Channel[timer] < 0 )
    *TCNTn = 0; // channel set to -1 indicated that refresh interval completed so reset the timer
  else{
    if( SERVO_INDEX(timer,Channel[timer]) < ServoCount && SERVO(timer,Channel[timer]).Pin.isActive == true )
      digitalWrite( SERVO(timer,Channel[timer]).Pin.nbr,LOW); // pulse this channel low if activated
  }

  Channel[timer]++;    // increment to the next channel
  if( SERVO_INDEX(timer,Channel[timer]) < ServoCount && Channel[timer] < SERVOS_PER_TIMER) {

            // Extension for slowmove
            if (SERVO(timer,Channel[timer]).speed) {
                        // Increment ticks by speed until we reach the target.
                        // When the target is reached, speed is set to 0 to disable that code.
                        if (SERVO(timer,Channel[timer]).target > SERVO(timer,Channel[timer]).ticks) {
                                    SERVO(timer,Channel[timer]).ticks += SERVO(timer,Channel[timer]).speed;
                                    if (SERVO(timer,Channel[timer]).target <= SERVO(timer,Channel[timer]).ticks) {
                                                SERVO(timer,Channel[timer]).ticks = SERVO(timer,Channel[timer]).target;
                                                SERVO(timer,Channel[timer]).speed = 0;
                                    }
                        }
                        else {
                                    SERVO(timer,Channel[timer]).ticks -= SERVO(timer,Channel[timer]).speed;
                                    if (SERVO(timer,Channel[timer]).target >= SERVO(timer,Channel[timer]).ticks) {
                                                SERVO(timer,Channel[timer]).ticks = SERVO(timer,Channel[timer]).target;
                                                SERVO(timer,Channel[timer]).speed = 0;
                                    }
                        }
            }
            // End of Extension for slowmove

            // Todo

    *OCRnA = *TCNTn + SERVO(timer,Channel[timer]).ticks;
    if(SERVO(timer,Channel[timer]).Pin.isActive == true)     // check if activated
      digitalWrite( SERVO(timer,Channel[timer]).Pin.nbr,HIGH); // its an active channel so pulse it high
  }
  else {
    // finished all channels so wait for the refresh period to expire before starting over
    if( (unsigned)*TCNTn <  (usToTicks(REFRESH_INTERVAL) + 4) )  // allow a few ticks to ensure the next OCR1A not
missed
      *OCRnA = (unsigned int)usToTicks(REFRESH_INTERVAL);
    else
      *OCRnA = *TCNTn + 4;  // at least REFRESH_INTERVAL has elapsed
    Channel[timer] = -1; // this will get incremented at the end of the refresh period to start again at the first channel
  }
}

#ifndef WIRING // Wiring pre-defines signal handlers so don't define any if compiling for the Wiring platform
// Interrupt handlers for Arduino
#if defined(_useTimer1)
SIGNAL (TIMER1_COMPA_vect)
{
  handle_interrupts(_timer1, &TCNT1, &OCR1A);
}
#endif
```

```
#if defined(_useTimer3)
SIGNAL (TIMER3_COMPA_vect)
{
  handle_interrupts(_timer3, &TCNT3, &OCR3A);
}
#endif

#if defined(_useTimer4)
SIGNAL (TIMER4_COMPA_vect)
{
  handle_interrupts(_timer4, &TCNT4, &OCR4A);
}
#endif

#if defined(_useTimer5)
SIGNAL (TIMER5_COMPA_vect)
{
  handle_interrupts(_timer5, &TCNT5, &OCR5A);
}
#endif

#elif defined WIRING
// Interrupt handlers for Wiring
#if defined(_useTimer1)
void Timer1Service()
{
  handle_interrupts(_timer1, &TCNT1, &OCR1A);
}
#endif
#if defined(_useTimer3)
void Timer3Service()
{
  handle_interrupts(_timer3, &TCNT3, &OCR3A);
}
#endif
#endif


static void initISR(timer16_Sequence_t timer)
{
#if defined (_useTimer1)
  if(timer == _timer1) {
    TCCR1A = 0;             // normal counting mode
    TCCR1B = _BV(CS11);     // set prescaler of 8
    TCNT1 = 0;              // clear the timer count
#if defined(__AVR_ATmega8__)|| defined(__AVR_ATmega128__)
    TIFR |= _BV(OCF1A);      // clear any pending interrupts;
    TIMSK |=  _BV(OCIE1A) ;  // enable the output compare interrupt
#else
    // here if not ATmega8 or ATmega128
    TIFR1 |= _BV(OCF1A);     // clear any pending interrupts;
    TIMSK1 |=  _BV(OCIE1A) ; // enable the output compare interrupt
#endif
#if defined(WIRING)
    timerAttach(TIMER1OUTCOMPAREA_INT, Timer1Service);
#endif
  }
#endif

#if defined (_useTimer3)
  if(timer == _timer3) {
    TCCR3A = 0;             // normal counting mode
    TCCR3B = _BV(CS31);     // set prescaler of 8
    TCNT3 = 0;              // clear the timer count
#if defined(__AVR_ATmega128__)
    TIFR |= _BV(OCF3A);     // clear any pending interrupts;
            ETIMSK |= _BV(OCIE3A);  // enable the output compare interrupt
#else
    TIFR3 = _BV(OCF3A);     // clear any pending interrupts;
```

```
    TIMSK3 =  _BV(OCIE3A) ; // enable the output compare interrupt
#endif
#if defined(WIRING)
    timerAttach(TIMER3OUTCOMPAREA_INT, Timer3Service);  // for Wiring platform only
#endif
  }
#endif

#if defined (_useTimer4)
  if(timer == _timer4) {
    TCCR4A = 0;             // normal counting mode
    TCCR4B = _BV(CS41);     // set prescaler of 8
    TCNT4 = 0;              // clear the timer count
    TIFR4 = _BV(OCF4A);     // clear any pending interrupts;
    TIMSK4 =  _BV(OCIE4A) ; // enable the output compare interrupt
  }
#endif

#if defined (_useTimer5)
  if(timer == _timer5) {
    TCCR5A = 0;             // normal counting mode
    TCCR5B = _BV(CS51);     // set prescaler of 8
    TCNT5 = 0;              // clear the timer count
    TIFR5 = _BV(OCF5A);     // clear any pending interrupts;
    TIMSK5 =  _BV(OCIE5A) ; // enable the output compare interrupt
  }
#endif
}

static void finISR(timer16_Sequence_t timer)
{
    //disable use of the given timer
#if defined WIRING   // Wiring
  if(timer == _timer1) {
    #if defined(__AVR_ATmega1281__)||defined(__AVR_ATmega2561__)
    TIMSK1 &=  ~_BV(OCIE1A) ;  // disable timer 1 output compare interrupt
    #else
    TIMSK &=  ~_BV(OCIE1A) ;  // disable timer 1 output compare interrupt
    #endif
    timerDetach(TIMER1OUTCOMPAREA_INT);
  }
  else if(timer == _timer3) {
    #if defined(__AVR_ATmega1281__)||defined(__AVR_ATmega2561__)
    TIMSK3 &= ~_BV(OCIE3A);    // disable the timer3 output compare A interrupt
    #else
    ETIMSK &= ~_BV(OCIE3A);    // disable the timer3 output compare A interrupt
    #endif
    timerDetach(TIMER3OUTCOMPAREA_INT);
  }
#else
    //For arduino - in future: call here to a currently undefined function to reset the timer
#endif
}

static boolean isTimerActive(timer16_Sequence_t timer)
{
  // returns true if any servo is active on this timer
  for(uint8_t channel=0; channel < SERVOS_PER_TIMER; channel++) {
    if(SERVO(timer,channel).Pin.isActive == true)
      return true;
  }
  return false;
}


/****************** end of static functions ******************************/

VarSpeedServo::VarSpeedServo()
{
```

```
  if( ServoCount < MAX_SERVOS) {
    this->servoIndex = ServoCount++;                    // assign a servo index to this instance
              servos[this->servoIndex].ticks = usToTicks(DEFAULT_PULSE_WIDTH);   // store default values  - 12 Aug 2009
    this->curSeqPosition = 0;
    this->curSequence = initSeq;
  }
  else
    this->servoIndex = INVALID_SERVO ;  // too many servos
}

uint8_t VarSpeedServo::attach(int pin)
{
  return this->attach(pin, MIN_PULSE_WIDTH, MAX_PULSE_WIDTH);
}

uint8_t VarSpeedServo::attach(int pin, int min, int max)
{
  if(this->servoIndex < MAX_SERVOS ) {
    pinMode( pin, OUTPUT) ;                             // set servo pin to output
    servos[this->servoIndex].Pin.nbr = pin;
    // todo min/max check: abs(min - MIN_PULSE_WIDTH) /4 < 128
    this->min  = (MIN_PULSE_WIDTH - min)/4; //resolution of min/max is 4 uS
    this->max  = (MAX_PULSE_WIDTH - max)/4;
    // initialize the timer if it has not already been initialized
    timer16_Sequence_t timer = SERVO_INDEX_TO_TIMER(servoIndex);
    if(isTimerActive(timer) == false)
      initISR(timer);
    servos[this->servoIndex].Pin.isActive = true;  // this must be set after the check for isTimerActive
  }
  return this->servoIndex ;
}

void VarSpeedServo::detach()
{
  servos[this->servoIndex].Pin.isActive = false;
  timer16_Sequence_t timer = SERVO_INDEX_TO_TIMER(servoIndex);
  if(isTimerActive(timer) == false) {
    finISR(timer);
  }
}

void VarSpeedServo::write(int value)
{
  if(value < MIN_PULSE_WIDTH)
  {  // treat values less than 544 as angles in degrees (valid values in microseconds are handled as microseconds)
    // updated to use constrain() instead of if(), pva
    value = constrain(value, 0, 180);
    value = map(value, 0, 180, SERVO_MIN(),  SERVO_MAX());
  }
  this->writeMicroseconds(value);
}

void VarSpeedServo::writeMicroseconds(int value)
{
  // calculate and store the values for the given channel
  byte channel = this->servoIndex;
  if( (channel >= 0) && (channel < MAX_SERVOS) )   // ensure channel is valid
  {
    if( value < SERVO_MIN() )          // ensure pulse width is valid
      value = SERVO_MIN();
    else if( value > SERVO_MAX() )
      value = SERVO_MAX();

            value -= TRIM_DURATION;
    value = usToTicks(value);  // convert to ticks after compensating for interrupt overhead - 12 Aug 2009

    uint8_t oldSREG = SREG;
    cli();
    servos[channel].ticks = value;
```

```
    SREG = oldSREG;

            // Extension for slowmove
            // Disable slowmove logic.
            servos[channel].speed = 0;
            // End of Extension for slowmove
  }
}

// Extension for slowmove
/*
  write(value, speed) - Just like write but at reduced speed.

  value - Target position for the servo. Identical use as value of the function write.
  speed - Speed at which to move the servo.
        speed=0 - Full speed, identical to write
        speed=1 - Minimum speed
        speed=255 - Maximum speed
*/
void VarSpeedServo::write(int value, uint8_t speed) {
            // This fuction is a copy of write and writeMicroseconds but value will be saved
            // in target instead of in ticks in the servo structure and speed will be save
            // there too.

  int degrees = value;

            if (speed) {
                        if (value < MIN_PULSE_WIDTH) {
                                    // treat values less than 544 as angles in degrees (valid values in microseconds are handled
as microseconds)
                        // updated to use constrain instead of if, pva
                        value = constrain(value, 0, 180);
                        value = map(value, 0, 180, SERVO_MIN(),  SERVO_MAX());
                        }
                        // calculate and store the values for the given channel
                        byte channel = this->servoIndex;
                        if( (channel >= 0) && (channel < MAX_SERVOS) ) {   // ensure channel is valid
                        // updated to use constrain instead of if, pva
                        value = constrain(value, SERVO_MIN(), SERVO_MAX());

                                    value = value - TRIM_DURATION;
                                    value = usToTicks(value);  // convert to ticks after compensating for interrupt overhead - 12
Aug 2009

                                    // Set speed and direction
                                    uint8_t oldSREG = SREG;
                                    cli();
                                    servos[channel].target = value;
                                    servos[channel].speed = speed;
                                    SREG = oldSREG;
                        }
            }
            else {
                        write (value);
            }
}

void VarSpeedServo::write(int value, uint8_t speed, bool wait) {
  write(value, speed);
  if (wait) { // block until the servo is at its new position
    if (value < MIN_PULSE_WIDTH) {
      while (read() != value) {
        delay(5);
      }
    } else {
      while (readMicroseconds() != value) {
        delay(5);
      }
    }
```

```
  }
}

void VarSpeedServo::stop() {
  write(read());
}

void VarSpeedServo::slowmove(int value, uint8_t speed) {
  // legacy function to support original version of VarSpeedServo
  write(value, speed);
}

// End of Extension for slowmove


int VarSpeedServo::read() // return the value as degrees
{
  return  map( this->readMicroseconds()+1, SERVO_MIN(), SERVO_MAX(), 0, 180);
}

int VarSpeedServo::readMicroseconds()
{
  unsigned int pulsewidth;
  if( this->servoIndex != INVALID_SERVO )
    pulsewidth = ticksToUs(servos[this->servoIndex].ticks)  + TRIM_DURATION ;   // 12 aug 2009
  else
    pulsewidth  = 0;

  return pulsewidth;
}

bool VarSpeedServo::attached()
{
  return servos[this->servoIndex].Pin.isActive ;
}

uint8_t VarSpeedServo::sequencePlay(servoSequencePoint sequenceIn[], uint8_t numPositions, bool loop, uint8_t
startPos) {
  uint8_t oldSeqPosition = this->curSeqPosition;

  if( this->curSequence != sequenceIn) {
    //Serial.println("newSeq");
    this->curSequence = sequenceIn;
    this->curSeqPosition = startPos;
    oldSeqPosition = 255;
  }

  if (read() == sequenceIn[this->curSeqPosition].position && this->curSeqPosition != CURRENT_SEQUENCE_STOP) {
    this->curSeqPosition++;

    if (this->curSeqPosition >= numPositions) { // at the end of the loop
      if (loop) { // reset to the beginning of the loop
        this->curSeqPosition = 0;
      } else { // stop the loop
        this->curSeqPosition = CURRENT_SEQUENCE_STOP;
      }
    }
  }

  if (this->curSeqPosition != oldSeqPosition && this->curSeqPosition != CURRENT_SEQUENCE_STOP) {
    // CURRENT_SEQUENCE_STOP position means the animation has ended, and should no longer be played
    // otherwise move to the next position
    write(sequenceIn[this->curSeqPosition].position, sequenceIn[this->curSeqPosition].speed);
    //Serial.println(this->seqCurPosition);
  }

  return this->curSeqPosition;
}
```

```
uint8_t VarSpeedServo::sequencePlay(servoSequencePoint sequenceIn[], uint8_t numPositions) {
  return sequencePlay(sequenceIn, numPositions, true, 0);
}

void VarSpeedServo::sequenceStop() {
  write(read());
  this->curSeqPosition = CURRENT_SEQUENCE_STOP;
}

/*
        To do
int VarSpeedServo::targetPosition() {
        byte channel = this->servoIndex;
        return map( servos[channel].target+1, SERVO_MIN(), SERVO_MAX(), 0, 180);
}

int VarSpeedServo::targetPositionMicroseconds() {
        byte channel = this->servoIndex;
        return servos[channel].target;
}

bool VarSpeedServo::isMoving() {
        byte channel = this->servoIndex;
        int servos[channel].target;
}
*/
```

VarSpeedServo.h

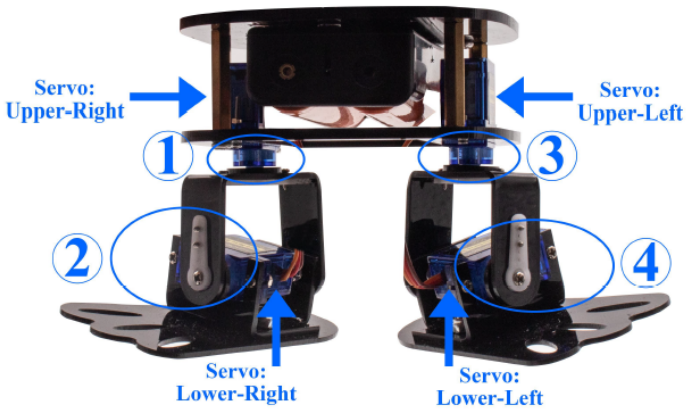
```
/*
  VarSpeedServo.h - Interrupt driven Servo library for Arduino using 16 bit timers- Version 2
  Copyright (c) 2009 Michael Margolis.  All right reserved.

  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General Public
  License along with this library; if not, write to the Free Software
  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
*/


/*
  Function slowmove and supporting code added 2010 by Korman. Above limitations apply
  to all added code, except for the official maintainer of the Servo library. If he,
  and only he deems the enhancment a good idea to add to the official Servo library,
  he may add it without the requirement to name the author of the parts original to
  this version of the library.
*/

/*
  Updated 2013 by Philip van Allen (pva),
  -- updated for Arduino 1.0 +
  -- consolidated slowmove into the write command (while keeping slowmove() for compatibility
     with Korman's version)
  -- added wait parameter to allow write command to block until move is complete
  -- added sequence playing ability to asynchronously move the servo through a series of positions, must be called in a
loop
```

```
  A servo is activated by creating an instance of the Servo class passing the desired pin to the attach() method.
  The servos are pulsed in the background using the value most recently written using the write() method

  Note that analogWrite of PWM on pins associated with the timer are disabled when the first servo is attached.
  Timers are seized as needed in groups of 12 servos - 24 servos use two timers, 48 servos will use four.
  The sequence used to sieze timers is defined in timers.h

  The methods are:

   VarSpeedServo - Class for manipulating servo motors connected to Arduino pins.

   attach(pin )  - Attaches a servo motor to an i/o pin.
   attach(pin, min, max  ) - Attaches to a pin setting min and max values in microseconds
   default min is 544, max is 2400

   write(value)     - Sets the servo angle in degrees.  (invalid angle that is valid as pulse in microseconds is treated as
microseconds)
   write(value, speed) - speed varies the speed of the move to new position 0=full speed, 1-255 slower to faster
   write(value, speed, wait) - wait is a boolean that, if true, causes the function call to block until move is complete

   writeMicroseconds() - Sets the servo pulse width in microseconds
   read()      - Gets the last written servo pulse width as an angle between 0 and 180.
   readMicroseconds()  - Gets the last written servo pulse width in microseconds. (was read_us() in first release)
   attached()  - Returns true if there is a servo attached.
   detach()    - Stops an attached servos from pulsing its i/o pin.

   slowmove(value, speed) - The same as write(value, speed), retained for compatibility with Korman's version

   stop() - stops the servo at the current position

   sequencePlay(sequence, sequencePositions); // play a looping sequence starting at position 0
   sequencePlay(sequence, sequencePositions, loop, startPosition); // play sequence with number of positions, loop if
true, start at position
   sequenceStop(); // stop sequence at current position

 */

#ifndef VarSpeedServo_h
#define VarSpeedServo_h

#include <inttypes.h>

/*
 * Defines for 16 bit timers used with  Servo library
 *
 * If _useTimerX is defined then TimerX is a 16 bit timer on the curent board
 * timer16_Sequence_t enumerates the sequence that the timers should be allocated
 * _Nbr_16timers indicates how many 16 bit timers are available.
 *
 */

// Say which 16 bit timers can be used and in what order
#if defined(__AVR_ATmega1280__)  || defined(__AVR_ATmega2560__)
#define _useTimer5
#define _useTimer1
#define _useTimer3
#define _useTimer4
typedef enum { _timer5, _timer1, _timer3, _timer4, _Nbr_16timers } timer16_Sequence_t ;

#elif defined(__AVR_ATmega32U4__)
#define _useTimer3
#define _useTimer1
typedef enum { _timer3, _timer1, _Nbr_16timers } timer16_Sequence_t ;

#elif defined(__AVR_AT90USB646__) || defined(__AVR_AT90USB1286__)
#define _useTimer3
#define _useTimer1
typedef enum { _timer3, _timer1, _Nbr_16timers } timer16_Sequence_t ;
```

```
#elif defined(__AVR_ATmega128__) ||defined(__AVR_ATmega1281__)||defined(__AVR_ATmega2561__)
#define _useTimer3
#define _useTimer1
typedef enum { _timer3, _timer1, _Nbr_16timers } timer16_Sequence_t ;

#else  // everything else
#define _useTimer1
typedef enum { _timer1, _Nbr_16timers } timer16_Sequence_t ;
#endif

#define VarSpeedServo_VERSION          2      // software version of this library

#define MIN_PULSE_WIDTH       544     // the shortest pulse sent to a servo
#define MAX_PULSE_WIDTH      2400     // the longest pulse sent to a servo
#define DEFAULT_PULSE_WIDTH  1500     // default pulse width when servo is attached
#define REFRESH_INTERVAL    20000     // minumim time to refresh servos in microseconds

#define SERVOS_PER_TIMER       12     // the maximum number of servos controlled by one timer
#define MAX_SERVOS   (_Nbr_16timers  * SERVOS_PER_TIMER)

#define INVALID_SERVO         255     // flag indicating an invalid servo index

#define CURRENT_SEQUENCE_STOP   255    // used to indicate the current sequence is not used and sequence
should stop


typedef struct  {
  uint8_t nbr        :6 ;           // a pin number from 0 to 63
  uint8_t isActive   :1 ;           // true if this channel is enabled, pin not pulsed if false
} ServoPin_t   ;

typedef struct {
  ServoPin_t Pin;
  unsigned int ticks;
            unsigned int target;                                  // Extension for slowmove
            uint8_t speed;                                                // Extension for slowmove
} servo_t;

typedef struct {
  uint8_t position;
  uint8_t speed;
} servoSequencePoint;

class VarSpeedServo
{
public:
  VarSpeedServo();
  uint8_t attach(int pin);          // attach the given pin to the next free channel, sets pinMode, returns channel number or 0
if failure
  uint8_t attach(int pin, int min, int max); // as above but also sets min and max values for writes.
  void detach();
  void write(int value);            // if value is < 200 its treated as an angle, otherwise as pulse width in microseconds
  void write(int value, uint8_t speed); // Move to given position at reduced speed.
        // speed=0 is identical to write, speed=1 slowest and speed=255 fastest.
        // On the RC-Servos tested, speeds differences above 127 can't be noticed,
        // because of the mechanical limits of the servo.
  void write(int value, uint8_t speed, bool wait); // wait parameter causes call to block until move completes
  void writeMicroseconds(int value); // Write pulse width in microseconds
  void slowmove(int value, uint8_t speed);
  void stop(); // stop the servo where it is

  int read();                     // returns current pulse width as an angle between 0 and 180 degrees
  int readMicroseconds();           // returns current pulse width in microseconds for this servo (was read_us() in first
release)
  bool attached();                  // return true if this servo is attached, otherwise false

  uint8_t sequencePlay(servoSequencePoint sequenceIn[], uint8_t numPositions, bool loop, uint8_t startPos);
  uint8_t sequencePlay(servoSequencePoint sequenceIn[], uint8_t numPositions); // play a looping sequence starting at
```

```
position 0
  void sequenceStop(); // stop movement
private:
   uint8_t servoIndex;              // index into the channel data for this servo
   int8_t min;                     // minimum is this value times 4 added to MIN_PULSE_WIDTH
   int8_t max;                      // maximum is this value times 4 added to MAX_PULSE_WIDTH
   servoSequencePoint * curSequence; // for sequences
   uint8_t curSeqPosition; // for sequences

};

#endif
```

Luego observe la postura de cada parte del robot que ha instalado, como las siguientes



1 la pierna derecha está mirando hacia afuera
2 el pie derecho está mirando hacia afuera
3 la pierna izquierda está mirando hacia afuera
4 el pie izquierdo está mirando hacia afuera.
En este momento, el ángulo de instalación del programa es de 90 grados.

Lesson_4_INSTALL | Arduino 1.8.9

File Edit Sketch Tools Help

Lesson_4_INSTALL | VarSpeedServo.cpp | VarSpeedServo.h

```
#include "VarSpeedServo.h"  //包含 VarSpeedServo library

VarSpeedServo RU;  //Right Upper
VarSpeedServo RL;  //Right Lower
VarSpeedServo LU;  //Left Upper
VarSpeedServo LL;  //Left Lower

const int vel = 30;
const int array_cal[4] = {90,90,90,90};  // Define the angular adjustment of servo (RU, RL, LU, LL )

void Servo_Init()
{
    RU.attach(3);   // Connect the signal wire of the upper-right servo to pin 3
    RL.attach(5);   // Connect the signal wire of the lower-right servo to pin 5
    LU.attach(6);   // Connect the signal wire of the upper-left  servo to pin 6
    LL.attach(9);   // Connect the signal wire of the lower-left  servo to pin 9
}
```

Por tanto, podemos calibrar de la siguiente manera:

① Disminuya el ángulo del servo superior derecho

Cambie 90 grados a 85 (array_cal [0]: es el ángulo de rotación del servo superior derecho);

② Disminuya el ángulo del servo inferior derecho

Cambie 90 grados a 85 (array_cal [1]: es el ángulo de rotación del servo inferior derecho);

③Aumentar el ángulo del servo superior izquierdo

Cambie 90 grados a 95 (array_cal [2]: es el ángulo de rotación del servo superior izquierdo);

④Aumentar el ángulo del servo inferior izquierdo

Cambie 90 grados a 95 (ar, ray_cal [3]: es el ángulo de rotación del servo inferior izquierdo);

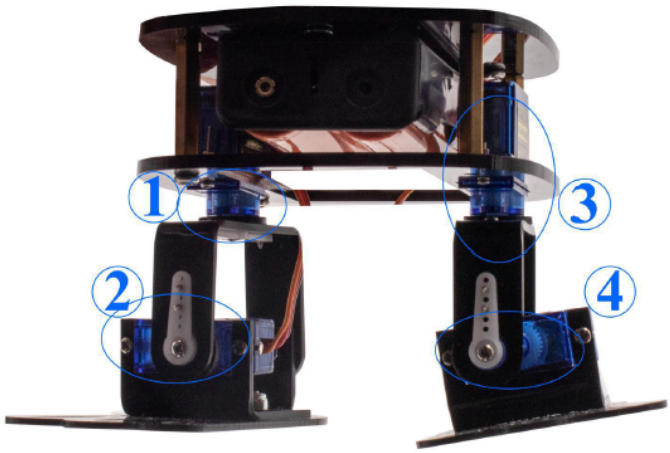
El programa modificado es el siguiente. Después de conectarse, cargue el código en su placa nano.

File Edit Sketch Tools Help

Lesson_5_CALIBRATION§ | VarSpeedServo.cpp | VarSpeedServo.h

```
#include "VarSpeedServo.h"  //include VarSpeedServo library
VarSpeedServo RU;  //Right Upper
VarSpeedServo RL;  //Right Lower
VarSpeedServo LU;  //Left Upper
VarSpeedServo LL;  //Left Lower

const int vel = 30;
const int array_cal[4] = {85,85,95,95};  // Define the angular adjustment of servo (RU, RL, LU, LL )
void Servo_Init()
{
    RU.attach(3);   // Connect the signal wire of the upper-right servo to pin 3
    RL.attach(5);   // Connect the signal wire of the lower-right servo to pin 5
    LU.attach(6);   // Connect the signal wire of the upper-left  servo to pin 6
    LL.attach(9);   // Connect the signal wire of the lower-left  servo to pin 9
}
void setup()
{
    Servo_Init();
}
void loop()
{
    RU.slowmove (array_cal[0] , vel);  // Define the angle and speed of the upper-right servo.
    RL.slowmove (array_cal[1] , vel);
    LU.slowmove (array_cal[2] , vel);
    LL.slowmove (array_cal[3] , vel);
    delay(2000);
```

Cargue el programa modificado, puede ver la postura del robot calibrado de la siguiente manera, luego observe el efecto ajustado, reduzca el ajuste y continuar cargando el programa. Por ejemplo, la pierna superior izquierda ya está alineada, entonces los parámetros del no es necesario modificar la parte superior de la pierna izquierda, siga observando y modifique los otros tres valores.



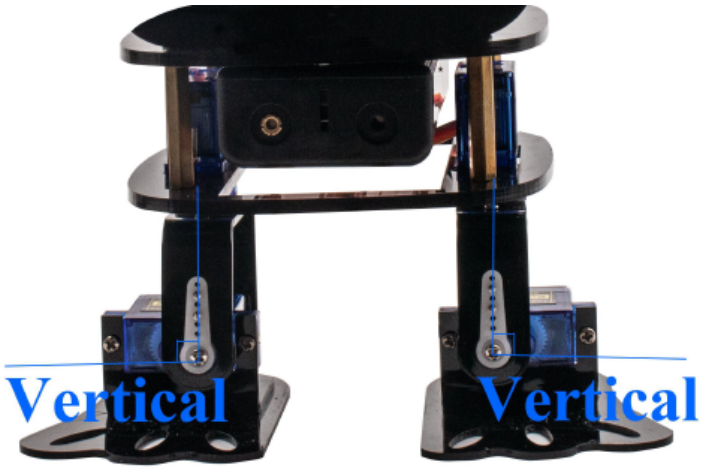
① Disminuya el ángulo del servo superior derecho
Cambie 85 grados a 82 (array_cal [0]: es el ángulo de rotación del servo superior derecho);
② Disminuya el ángulo del servo inferior derecho
Cambie 85 grados a 83 (array_cal [1]: es el ángulo de rotación del servo inferior derecho);
④Aumentar el ángulo del servo inferior izquierdo
Cambie 95 grados a 97 (array_cal [3]: es el ángulo de rotación del servo inferior izquierdo);
El programa modificado es el siguiente. Después de conectarse, cargue el código en su placa nano.

File Edit Sketch Tools Help

Lesson_5_CALIBRATION§ VarSpeedServo.cpp VarSpeedServo.h

```
#include "VarSpeedServo.h"  //include VarSpeedServo library
VarSpeedServo RU;  //Right Upper
VarSpeedServo RL;  //Right Lower
VarSpeedServo LU;  //Left Upper
VarSpeedServo LL;  //Left Lower

const int vel = 30;
const int array_cal[4] = {82,83,95,97};  // Define the angular adjustment of servo (RU, RL, LU, LL )
void Servo_Init()
{
    RU.attach(3);   // Connect the signal wire of the upper-right servo to pin 3
    RL.attach(5);   // Connect the signal wire of the lower-right servo to pin 5
    LU.attach(6);   // Connect the signal wire of the upper-left  servo to pin 6
    LL.attach(9);   // Connect the signal wire of the lower-left  servo to pin 9
}
void setup()
{
    Servo_Init();
}
```

Observe los cuatro servos para asegurarse de que estén en el ángulo correcto, luego se completa la calibración del servo. Tu puedes hacer ajuste fino con cambio de valor de “1” cada vez, si hay una pequeña desviación. Ajuste hasta que la pierna del robot esté en un posición completamente vertical para completar el ajuste. El valor obtenido en el software después del ajuste es el valor de calibración del robot que instaló usted mismo. En las lecciones 6 a 9, utilizará la calibración del robot.
El valor de calibración reemplaza el valor en el programa de referencia.



Lección 6 Prueba de movimiento

Acerca de esta lección:

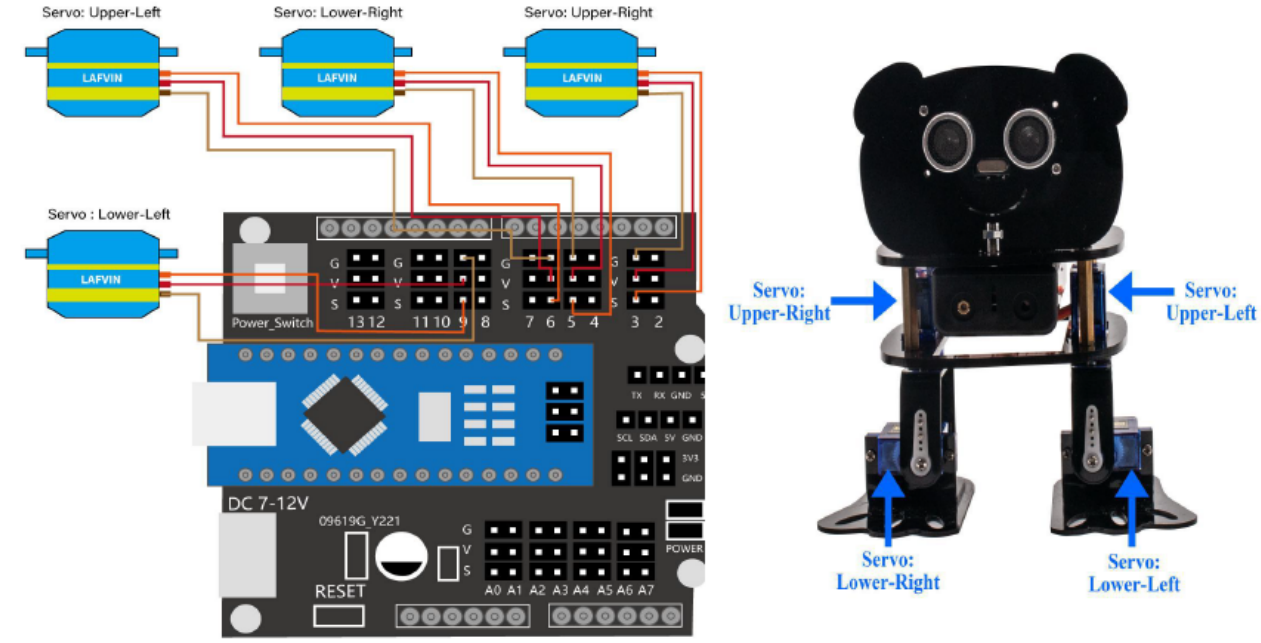
En esta lección, aprenderemos cómo controlar el servomotor para girar un ángulo específico a través del tablero de control principal del robot.

Introducción

Los servomotores son excelentes dispositivos que pueden girar a una posición específica y, por lo general, tienen un brazo servo que puede girar 180 grados. Utilizando Arduino, podemos decirle a un servo que vaya a una posición específica y que irá allí. ¡Tan simple como eso! Los servomotores se utilizaron por primera vez en el Mundo de control remoto (RC), generalmente para controlar la dirección de los coches RC o los flaps en un avión RC. Con el tiempo, encontraron sus usos en robótica, automatización y, por supuesto, el mundo Arduino.

Hay dos formas de controlar un servomotor con Arduino. Uno es utilizar un puerto de sensor digital común de Arduino para producir onda cuadrada con ciclo de trabajo diferente para simular la señal PWM y usar esa señal para controlar el posicionamiento del motor. De otra manera es usar directamente la función Servo del Arduino para controlar el motor. De esta forma, el programa será más fácil. A continuación, aprenderemos a controlar el servo El servomotor tiene tres cables. El color de los cables varía entre los servomotores, pero el cable rojo es siempre de 5 V y GND será marrón, el rojo es el cable de alimentación y debe estar conectado al puerto de 5v, que suele ser naranja.

Wiring diagram



Instrucciones de prueba

Después de conectarse, abra el programa. Los siguientes parámetros se basan en la calibración del robot que instalé. Antes de usar esto programa de referencia, debe calibrar el robot que instaló (consulte la Lección 5 para conocer el método de calibración) y cambiar los parámetros en el cuadro rojo para Sus datos calibrados. Una vez completada la modificación, cargue el código - Lesson_6_Motion_Test en su placa Arduino Nano.

Se vuelven hacia adentro y se aplanan, y la simple acción del ciclo continuo

Lesson_6_Motion_Test

```
#include "VarSpeedServo.h"  //include VarSpeedServo library

VarSpeedServo RU;  //Right Upper
VarSpeedServo RL;  //Right Lower
VarSpeedServo LU;  //Left Upper
VarSpeedServo LL;  //Left Lower

const int vel = 30;
const int array_cal[4] = {81,80,90,92};   // Definir el ajuste angular del servo (RU, RL, LU, LL )



const int num_dance = 8;
const int array_dance[num_dance][4] =
{
   {0,-40,0,40},    //En un lugar para subir y bajar
   {20,-40,-20,40},
```

```
    {20,-20,-20,20},
    {20,0,-20,0},
    {-20,-10,20,10},
    {-10,-30,10,30},
    {0,-40,0,40},
    {0,0,0,0},
};


void Servo_Init()
{
    RU.attach(3);   // Conecte el cable de señal del upper-right servo al pin 3
    RL.attach(5);   // Conecte el cable de señal del lower-right servo al pin 5
    LU.attach(6);   // Conecte el cable de señal del upper-left  servo al pin 6
    LL.attach(9);   // Conecte el cable de señal del lower-left  servo al pin 9

    RU.slowmove (array_cal[0] , vel);  // Defina el ángulo y la velocidad del servo superior derecho.
    RL.slowmove (array_cal[1] , vel);
    LU.slowmove (array_cal[2] , vel);
    LL.slowmove (array_cal[3] , vel);
    delay(2000);
}

void setup()
{
    Servo_Init();
}

void loop()
{
  In_place();
}

void In_place()
{ int vel_Dance=40;
  for(int x = 0; x < 3; x++) {
       for(int z=0; z<6; z++) {
          RU.slowmove (array_cal[0] + array_dance[z][0] , vel_Dance);
          RL.slowmove (array_cal[1] + array_dance[z][1] , vel_Dance);
          LU.slowmove (array_cal[2] + array_dance[z][2] , vel_Dance);
          LL.slowmove (array_cal[3] + array_dance[z][3] , vel_Dance);
          delay(300);
       }
    }

     for(int z=6; z<8; z++) {
          RU.slowmove (array_cal[0] + array_dance[z][0] , 30);
          RL.slowmove (array_cal[1] + array_dance[z][1] , 30);
          LU.slowmove (array_cal[2] + array_dance[z][2] , 30);
          LL.slowmove (array_cal[3] + array_dance[z][3] , 30);
          delay(550);
       }
  }
```

Lección 7 Sígueme / Evítame

Acerca de esta lección:

En esta lección, aprendemos el uso de sensores ultrasónicos y luego usamos sensores ultrasónicos para detectar distancias para lograr que los robots sigan a su maestro.
Podemos utilizar sensores ultrasónicos para controlar la distancia. Cuando hay un obstáculo en la parte delantera, el robot deja de avanzar.

Introducción

Sensor ultrasónico

El módulo de sensor ultrasónico HC-SR04 proporciona una función de medición sin contacto de 2 cm a 400 cm, la precisión de rango puede llegar a 3 mm. Los módulos Incluye transmisores ultrasónicos, receptor y circuito de control. El principio básico del trabajo:

(1) Usando disparador IO para al menos 10us de señal de alto nivel,
(2) El Módulo envía automáticamente ocho a 40 kHz y detecta si hay una señal de pulso de regreso.
(3) SI la señal regresa, a través de un nivel alto, el tiempo de duración de la E / S de alta salida es el tiempo desde que se envía el desgarro ultrasónico.

Distancia de prueba = (tiempo de alto nivel × velocidad del sonido (340 m / s) / 2

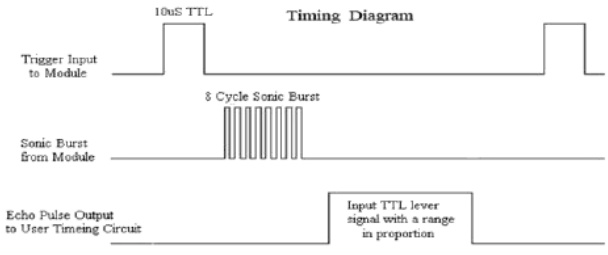
El diagrama de tiempos se muestra a continuación. Solo necesita suministrar un pulso corto de 10us a la entrada del disparador para iniciar el rango, y luego el módulo enviará emitir una ráfaga de 8 ciclos de ultrasonido a 40 kHz y aumentar su eco. El eco es un objeto de distancia que tiene el ancho de pulso y el rango en proporción.

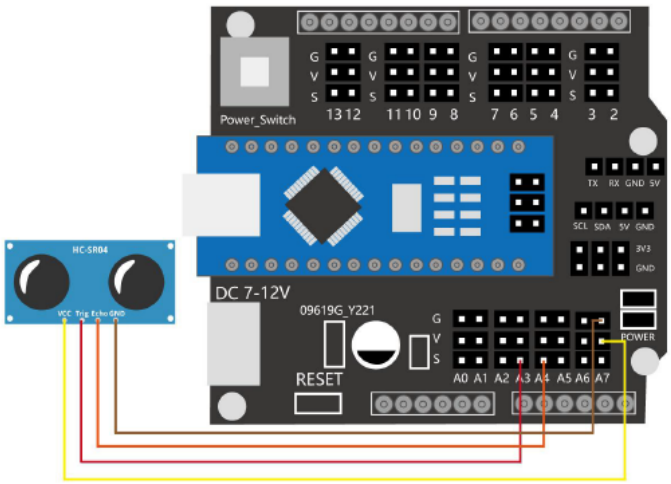
Calcule el rango a través del intervalo de tiempo entre el envío de la señal de activación y la recepción de la señal de eco.

Fórmula: us / 58 = centímetros o us / 148 = pulgada;

o: el rango = tiempo de alto nivel * velocidad (340M / S) / 2;

Sugerimos utilizar un ciclo de medición de más de 60 ms, para evitar la señal de disparo al eco.





Instrucciones de prueba

Sígueme:

Abra el programa en la carpeta de códigos - Lesson_7_Follow_Me. Antes de usar este programa de referencia, debe calibrar el robot que instalado (vea la Lección 5 para el método de calibración) y cambie los parámetros en el cuadro rojo a Sus datos calibrados. Después de descargar el programa, la distancia detectada por el sensor ultrasónico se imprimirá en la ventana de serie, puede ver la siguiente información en Herramientas -> Serie

Lesson_7_Follow_Me

```
#include "VarSpeedServo.h"  //include VarSpeedServo library

VarSpeedServo RU;  //Right Upper
VarSpeedServo RL;  //Right Lower
VarSpeedServo LU;  //Left Upper
VarSpeedServo LL;  //Left Lower



int Echo = A4;  //
int Trig =A3;  //
int Distance = 0;
                                    //vel(min), delay_Forward(max) = (5, 2000)
const int vel = 40, vel_Back = 10;                  //vel(mid), delay_Forward(mid) = (20, 750)
const int delay_Forward = 300, delay_Back = 1000;    //vel(max), delay_Forward(min)= (256, 50)
                                    //wonderful ---> (10, 700) (50, 500) (100, 100) (100, 300) (100, 500)
const int array_cal[4] = {81,80,90,92};   // Define the angular adjustment of servo (RU, RL, LU, LL )
const int delay_tim = 300;  //Delay 750ms
int RU_Degree = 0, LU_Degree = array_cal[2] + 5;

const int num1 = 8;
const int array_forward[num1][4] =
{

   {0,-40,0,-20},       //forward
   {30,-40,30,-20},
   {30,0,30,0},
   {0,20,0,40},
   {-30,20,-30,40},
   {-30,0,-30,0},
   {-15,0,-15,0},
```

```
    {0,0,0,0},
};
const int num2 = 6;
const int array_back[num2][4] ={
    {-40,0,-20,0},
    {-40,30,-20,30},
    {0,30,0,30},

    {40,0,20,0},
    {40,-30,20,-30},
    {0,-30,0,-30},
 };



void Servo_Init()
{
    RU.attach(3);   // Connect the signal wire of the upper-right servo to pin 9
    RL.attach(5);   // Connect the signal wire of the lower-right servo to pin 10
    LU.attach(6);   // Connect the signal wire of the upper-left  servo to pin 11
    LL.attach(9);   // Connect the signal wire of the lower-left  servo to pin 12
}

void Adjust()                          // Avoid the servo's fast spinning in initialization
{                                   // RU,LU goes from array_cal[0] - 5 ,array_cal[2] + 5 degrees to array_cal[0],array_cal[2]
degrees
    for(RU_Degree = array_cal[0] - 5; RU_Degree <= array_cal[0]; RU_Degree += 1) {
        RU.write(RU_Degree);             // in steps of 1 degree
        LU.write(LU_Degree--);           // tell servo to go to RU_Degreeition, LU_Degreeition in variable 'RU_Degree',
'LU_Degree'
        delay(15);                       // waits 15ms for the servo to reach the RU_Degreeition
    }
}

void Ultrasonic()
{

  digitalWrite(Trig, LOW);   // Give the trigger pin low level 2μs
  delayMicroseconds(2);
  digitalWrite(Trig, HIGH);  // Give the trigger pin a high level of 10μs, here at least 10μs
  delayMicroseconds(10);
  digitalWrite(Trig, LOW);    // Continue to low voltage to the trigger pin
  float Fdistance = pulseIn(Echo, HIGH);  // Read high time (unit: microsecond)
  Fdistance= Fdistance/58;       //Why divide by 58 equals centimeter, Y m = (X seconds * 344) / 2
  // X seconds = ( 2 * Y meters) / 344 = = "X seconds = 0.0058 * Y meters = = "cm = microseconds / 58
  Serial.print("Distance:");      //Output distance (unit: cm)
  Serial.println(Fdistance);        //Display distance
  Distance = Fdistance;
}

void Forward()
{for(int z=0; z<1; z++) {
    for(int x=0; x<num1; x++) {
        RU.slowmove (array_cal[0] + array_forward[x][0] , vel);
        RL.slowmove (array_cal[1] + array_forward[x][1] , vel);
        LU.slowmove (array_cal[2] + array_forward[x][2] , vel);
        LL.slowmove (array_cal[3] + array_forward[x][3] , vel);
        delay(delay_Forward);
    }

}
}

void Backward()
{
  for(int y=0; y<num2; y++) {
      RU.slowmove (array_cal[0] + array_back[y][0] , vel_Back);
      RL.slowmove (array_cal[1] + array_back[y][1] , vel_Back);
```

```
    LU.slowmove (array_cal[2] + array_back[y][2] , vel_Back);
    LL.slowmove (array_cal[3] + array_back[y][3] , vel_Back);
    delay(delay_Back);
  }
}


void setup()
{
  Serial.begin(9600);
   Servo_Init();
   RU.slowmove (array_cal[0] , vel);
   RL.slowmove (array_cal[1] , vel);
   LU.slowmove (array_cal[2] , vel);
   LL.slowmove (array_cal[3] , vel);
   delay(2000);
 pinMode(Echo, INPUT);    // Define ultrasonic input pin
 pinMode(Trig, OUTPUT);   // Define ultrasonic output pin
}

void loop()
{
   Ultrasonic();
   if(Distance>20)
   {
     Forward();
   }

}
```

Monitor del software Arduino IDE.

Después de descargar el programa y observar la información de distancia impresa por la ventana del puerto serie, desenchufe el cable de datos USB y encienda en el interruptor de encendido del robot. Puede mover la mano cerca de la parte frontal del sensor ultrasónico. Cuando el robot está a menos de 20 cm de distancia de su mano, se detendrá. Mueva su mano hacia adelante. Cuando el robot esté a más de 20 cm de su mano, el robot se acercará Tu mano.

Evitarme:

Abra el programa en la carpeta de códigos - Lesson_7_Avoid_Me. Antes de utilizar este programa de referencia, debe calibrar el robot que instalado (vea la Lección 5 para el método de calibración) y cambie los parámetros en el cuadro rojo a Sus datos calibrados.

Lesson_7_Avoid_Me

```
#include "VarSpeedServo.h"  //include VarSpeedServo library

VarSpeedServo RU;  //Right Upper
VarSpeedServo RL;  //Right Lower
VarSpeedServo LU;  //Left Upper
VarSpeedServo LL;  //Left Lower



int Echo = A4;  //
int Trig =A3;  //
int Distance = 0;
                                        //vel(min), delay_Forward(max) = (5, 2000)
const int vel = 40, vel_Back = 40;                  //vel(mid), delay_Forward(mid) = (20, 750)
const int delay_Forward = 300, delay_Back = 450;    //vel(max), delay_Forward(min)= (256, 50)
                                        //wonderful ---> (10, 700) (50, 500) (100, 100) (100, 300) (100, 500)
const int array_cal[4] = {81,80,90,92};   // Define the angular adjustment of servo (RU, RL, LU, LL )
const int delay_tim = 300;  //Delay 750ms
int RU_Degree = 0, LU_Degree = array_cal[2] + 5;

const int num1 = 8;
const int array_forward[num1][4] =
{

   {0,-40,0,-20},       //forward
   {30,-40,30,-20},
   {30,0,30,0},
   {0,20,0,40},
   {-30,20,-30,40},
   {-30,0,-30,0},
   {-15,0,-15,0},
   {0,0,0,0},
};

const int num2 = 8;
const int array_back[num2][4] =
{

   {0,-40,0,-20},       //Step-back
   {-30,-40,-30,-20},
   {-30,0,-30,0},
   {0,20,0,40},
   {30,20,30,40},
   {30,0,30,0},
   {15,0,15,0},
   {0,0,0,0},
};


void Servo_Init()
{
   RU.attach(3);   // Connect the signal wire of the upper-right servo to pin 9
   RL.attach(5);   // Connect the signal wire of the lower-right servo to pin 10
   LU.attach(6);   // Connect the signal wire of the upper-left  servo to pin 11
   LL.attach(9);   // Connect the signal wire of the lower-left  servo to pin 12
}

void Adjust()                          // Avoid the servo's fast spinning in initialization
{                                  // RU,LU goes from array_cal[0] - 5 ,array_cal[2] + 5 degrees to array_cal[0],array_cal[2]
degrees
   for(RU_Degree = array_cal[0] - 5; RU_Degree <= array_cal[0]; RU_Degree += 1) {
      RU.write(RU_Degree);             // in steps of 1 degree
      LU.write(LU_Degree--);           // tell servo to go to RU_Degreeition, LU_Degreeition in variable 'RU_Degree',
'LU_Degree'
      delay(15);                       // waits 15ms for the servo to reach the RU_Degreeition
   }
}
```

```
void Ultrasonic()
{

  digitalWrite(Trig, LOW);   // Give the trigger pin low level 2μs
  delayMicroseconds(2);
  digitalWrite(Trig, HIGH);  // Give the trigger pin a high level of 10μs, here at least 10μs
  delayMicroseconds(10);
  digitalWrite(Trig, LOW);    // Continue to low voltage to the trigger pin
  float Fdistance = pulseIn(Echo, HIGH);  // Read high time (unit: microsecond)
  Fdistance= Fdistance/58;       //Why divide by 58 equals centimeter, Y m = (X seconds * 344) / 2
  // X seconds = ( 2 * Y meters) / 344 = = "X seconds = 0.0058 * Y meters = = "cm = microseconds / 58
  Serial.print("Distance:");      //Output distance (unit: cm)
  Serial.println(Fdistance);        //Display distance
  Distance = Fdistance;
}

void Forward()
{for(int z=0; z<1; z++) {
    for(int x=0; x<num1; x++) {
        RU.slowmove (array_cal[0] + array_forward[x][0] , vel);
        RL.slowmove (array_cal[1] + array_forward[x][1] , vel);
        LU.slowmove (array_cal[2] + array_forward[x][2] , vel);
        LL.slowmove (array_cal[3] + array_forward[x][3] , vel);
        delay(delay_Forward);
    }

}
}

void Backward()
{for(int z=0; z<3; z++) {
  for(int y=0; y<num2; y++) {
      RU.slowmove (array_cal[0] + array_back[y][0] , vel_Back);
      RL.slowmove (array_cal[1] + array_back[y][1] , vel_Back);
      LU.slowmove (array_cal[2] + array_back[y][2] , vel_Back);
      LL.slowmove (array_cal[3] + array_back[y][3] , vel_Back);
      delay(delay_Back);
                        }
                      }
}



void setup()
{
   Serial.begin(9600);
    Servo_Init();
    RU.slowmove (array_cal[0] , vel);
    RL.slowmove (array_cal[1] , vel);
    LU.slowmove (array_cal[2] , vel);
    LL.slowmove (array_cal[3] , vel);
    delay(2000);
  pinMode(Echo, INPUT);    // Define ultrasonic input pin
  pinMode(Trig, OUTPUT);   // Define ultrasonic output pin
}

void loop()
{
    Ultrasonic();
    if(Distance>20)
    {
      Forward();
    }else
    {Backward();
      }

}
```

Después de descargar el programa, la distancia detectada por el sensor ultrasónico se imprimirá en la ventana de serie, puede ver la siguiente información en Herramientas -> Serie Monitor del software Arduino IDE.

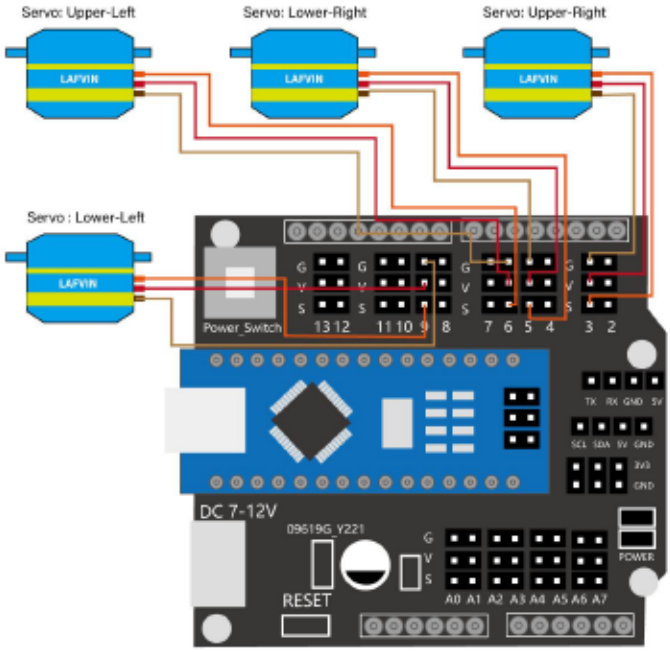
Después de descargar el programa y observar la información de distancia impresa por la ventana del puerto serie, desenchufe el cable de datos USB y encienda en el interruptor de encendido del robot. Puede mover su mano cerca de la parte frontal del sensor ultrasónico. El robot sigue avanzando hasta que el robot está a menos de 20 cm de su mano y retrocederá tres pasos. Al modificar el programa, puede personalizar esta distancia.

Lección 8 Movimiento de baile básico

Acerca de esta lección:

En esta lección, aprendemos a escribir un programa para controlar el robot para completar el grupo de acción de baile. El cableado del hardware no debe ser cambiado. Principalmente escribimos el grupo de acción de baile en el programa.

Diagrama de cableado



Instrucciones de prueba

Por favor abra el programa en la carpeta de código- Lesson_8_Basic_Dance_Movement. Antes de usar este programa de referencia, debe calibrar el robot que instaló (consulte la Lección 5 para conocer el método de calibración) y cambie los parámetros en el cuadro rojo a sus datos calibrados.

Conecte arduino Nano a la computadora usando un cable USB, haga clic en SUBIR para cargar el programa. El robot entrará en el estado de baile, completa 6 bailes en una fila, y se detendrá después de que se complete el baile. Si desea que el robot continúe bailando, presione el botón de reinicio en la placa nano.

Lesson_8_Basic_Dance_Movement

```
#include "VarSpeedServo.h"  //include the VarSpeedServo library
//#include <Servo.h>

VarSpeedServo RU;  //Right Upper
VarSpeedServo RL;  //Right Lower
VarSpeedServo LU;  //Left Upper
VarSpeedServo LL;  //Left Lower

                                        //vel(min), delay_Forward(max) = (5, 2000)
const int vel = 40, vel_Back = 15;                  //vel(mid), delay_Forward(mid) = (20, 750)
const int delay_Forward = 300, delay_Back = 750;    //vel(max), delay_Forward(min)= (256, 50)
                                        //wonderful ---> (10, 700) (50, 500) (100, 100) (100, 300) (100, 500)

int vel_Dance1 = 30,    vel_Dance2 = 25,    vel_Dance3 = 40;
int delay_Dance1 = 100, delay_Dance2 = 550, delay_Dance3 = 50;

int vel_Dance4 = 40,    vel_Dance5 = 40,    vel_Dance6 = 30;
int delay_Dance4 = 200, delay_Dance5 = 200, delay_Dance6 = 300;

const int array_cal[4] = {81,80,90,92};////first
//const int array_cal[4] = {90,95,90,88};    //second
int RU_Degree = 0, LU_Degree = array_cal[2] + 5;
```

```
const int num_dance1 = 10;
const int array_dance1[num_dance1][4] =
{
//slide to the left 0-4
    {0,-20,0,0},
    {0,-40,0,20},
    {0,-20,0,40},
    {0,0,0,20},
    {0,0,0,0},

//slide to the right 5-9
    {0,0,0,20},
    {0,-20,0,40},
    {0,-40,0,20},
    {0,-20,0,0},
    {0,0,0,0},
};

const int num_dance2 = 32;
const int array_dance2[num_dance2][4] =
{
//left foot support 0-15
    {20,0,40,0},
    {20,-30,40,-30},
    {20,-30,10,-30},
    {20,-30,40,-30},
    {20,-30,10,-30},

    {20,-30,40,-30},
    {20,0,40,-30},
    {20,80,40,-30},
    {20,0,40,-30},
    {20,-80,40,-30},
    {20,0,40,-30},
    {20,80,40,-30},
    {20,0,40,-30},
    {20,-30,40,-30},
    {20,0,40,0},
    {0,0,0,0},

//right foot support 16-31
    {-40,0,-20,0},
    {-40,40,-20,30},
    {-20,40,-20,30},
    {-40,40,-20,30},
    {-20,40,-20,30},

    {-40,40,-20,30},
    {-40,40,-20,0},
    {-40,40,-20,-80},
    {-40,40,-20,0},
    {-40,40,-20,80},
    {-40,40,-20,0},
    {-40,40,-20,-80},
    {-40,40,-20,0},
    {-40,40,-20,30},
    {-40,0,-20,0},
    {0,0,0,0},
};

const int num_dance3 = 8;       //Take a small step
const int array_dance3[num_dance3][4] =
{
    {0,-40,0,0},
    {20,-30,20,20},
    {40,0,40,30},
    {0,0,0,40},
    {-20,-20,-20,30},
    {-40,-30,-40,0},
```

```
    {0,-40,0,0},
    {0,0,0,0},
};

const int num_dance4 = 20;      //In-situ ups and downs, increasing in angle
const int array_dance4[num_dance4][4] =
{
    {0,-20,0,20},
    {0,0,0,0},
    {0,-20,0,20},
    {0,0,0,0},
    {0,-20,0,20},
    {0,0,0,0},
    {0,-20,0,20},
    {0,0,0,0},

    {0,-50,0,50},
    {0,0,0,0},
    {0,-50,0,50},
    {0,0,0,0},
    {0,-50,0,50},
    {0,0,0,0},
    {0,-50,0,50},
    {0,0,0,0},

    {0,-40,0,40},
    {0,-50,0,50},
    {0,-60,0,60},
    {0,0,0,0,},
};

const int num_dance5 = 12;
const int array_dance5[num_dance5][4] =
{


    {0,-40,0,40},    //Inner flip foot
    {-30,-40,-20,40},
    {0,-40,0,40},
    {20,-40,30,40},


    {0,-40,0,40},    //In place to rise and fall
    {20,-40,-20,40},
    {20,-20,-20,20},
    {20,0,-20,0},
    {-20,-10,20,10},
    {-10,-30,10,30},

    {0,-40,0,40},
    {0,0,0,0},
};

const int num_dance6 = 32;
const int array_dance6[num_dance6][4] =
{
    {0,-40,0,-20},        //Outer Eight Steps - Advance
    {25,-40,18,-20},
    {25,0,18,0},
    {0,20,0,40},
    {-18,20,-25,40},
    {-18,0,-25,0},

    {0,-40,0,-20},        //Outer Eight Steps - Stepping In Place
    {25,-40,18,-20},
    {0,0,0,0},
    {0,20,0,40},
    {-18,20,-25,40},
```

```
    {0,0,0,0},

    {0,-40,0,-20},       //Outer Eight Steps - Backward
    {-25,-40,-18,-20},
    {-25,0,-18,0},
    {0,20,0,40},
    {18,20,25,40},
    {18,0,25,0},

    {0,-40,0,-20},       //Pace - advance
    {30,-40,30,-20},
    {30,0,30,0},
    {0,20,0,40},
    {-30,20,-30,40},
    {-30,0,-30,0},

    {0,-40,0,-20},       //Step-back
    {-30,-40,-30,-20},
    {-30,0,-30,0},
    {0,20,0,40},
    {30,20,30,40},
    {30,0,30,0},

    {15,0,15,0},
    {0,0,0,0},
};

//#define INSTALL
//#define CALIBRATION
#define RUN

void Servo_Init()
{
    RU.attach(3);   // Connect the signal wire of the upper-right servo to pin 9
    RL.attach(5);   // Connect the signal wire of the lower-right servo to pin 10
    LU.attach(6);   // Connect the signal wire of the upper-left  servo to pin 11
    LL.attach(9);   // Connect the signal wire of the lower-left  servo to pin 12
}

void Adjust()                       // Avoid the servo's fast spinning in initialization
{                                   // RU,LU goes from array_cal[0] - 5 ,array_cal[2] + 5 degrees to array_cal[0],array_cal[2]
degrees
    for(RU_Degree = array_cal[0] - 5; RU_Degree <= array_cal[0]; RU_Degree += 1) {
        RU.write(RU_Degree);            // in steps of 1 degree
        LU.write(LU_Degree--);          // tell servo to go to RU_Degreeition, LU_Degreeition in variable 'RU_Degree',
'LU_Degree'
        delay(15);                      // waits 15ms for the servo to reach the RU_Degreeition
    }
}



void Slide_2_Left(int times)
{
    for(int time1 = 0; time1 < times; time1++) {
        for(int z=0; z<5; z++) {
            RU.slowmove (array_cal[0] + array_dance1[z][0] , vel_Dance1);
            RL.slowmove (array_cal[1] + array_dance1[z][1] , vel_Dance1);
            LU.slowmove (array_cal[2] + array_dance1[z][2] , vel_Dance1);
            LL.slowmove (array_cal[3] + array_dance1[z][3] , vel_Dance1);
            delay(delay_Dance1);
        }
    }
}

void Slide_2_Right(int times)
{
    for(int time1 = 0; time1 < times; time1++) {
        for(int z=5; z<10; z++) {
```

```
            RU.slowmove (array_cal[0] + array_dance1[z][0] , vel_Dance1);
            RL.slowmove (array_cal[1] + array_dance1[z][1] , vel_Dance1);
            LU.slowmove (array_cal[2] + array_dance1[z][2] , vel_Dance1);
            LL.slowmove (array_cal[3] + array_dance1[z][3] , vel_Dance1);
            delay(delay_Dance1);
        }
    }
}

void Left_Foot_Support()
{
    for(int z=0; z<16; z++) { //z<12
        if ( z > 5 && z < 14) {   //z(1,10)
            vel_Dance2 = 50;
            delay_Dance2 = 200;
        }
        else {
        vel_Dance2 = 25;
        delay_Dance2 = 750;
        }

        RU.slowmove (array_cal[0] + array_dance2[z][0] , vel_Dance2);
        RL.slowmove (array_cal[1] + array_dance2[z][1] , vel_Dance2);
        LU.slowmove (array_cal[2] + array_dance2[z][2] , vel_Dance2);
        LL.slowmove (array_cal[3] + array_dance2[z][3] , vel_Dance2);
        delay(delay_Dance2);
    }
}

void Right_Foot_Support()
{
    for(int z=16; z<32; z++) { //z<24
        if ( z > 21 && z < 30) {   //z(13,22)
            vel_Dance2 = 50;
            delay_Dance2 = 200;
        }
        else {
        vel_Dance2 = 25;
        delay_Dance2 = 750;
        }

        RU.slowmove (array_cal[0] + array_dance2[z][0] , vel_Dance2);
        RL.slowmove (array_cal[1] + array_dance2[z][1] , vel_Dance2);
        LU.slowmove (array_cal[2] + array_dance2[z][2] , vel_Dance2);
        LL.slowmove (array_cal[3] + array_dance2[z][3] , vel_Dance2);
        delay(delay_Dance2);
    }
}

void Dancing1_2()
{
    Slide_2_Left(2);
    Left_Foot_Support();

    Slide_2_Right(2);
    Right_Foot_Support();
}

void Dancing3(int Times = 1, int Vel = 40, int Delay = 250, int low = 0, int high = 0)
{
    for(int time3 = 0; time3 < Times; time3++) {
        for(int z=0; z<6; z++) {
            if ( time3 > 1 && time3 < 4) {
            vel_Dance3 = Vel;
            delay_Dance3 = Delay;
            }
            else {
            vel_Dance3 = 40;
            delay_Dance3 = 200;
```

```
        }

        RU.slowmove (array_cal[0] + array_dance3[z][0] , vel_Dance3);
        RL.slowmove (array_cal[1] + array_dance3[z][1] , vel_Dance3);
        LU.slowmove (array_cal[2] + array_dance3[z][2] , vel_Dance3);
        LL.slowmove (array_cal[3] + array_dance3[z][3] , vel_Dance3);
        delay(delay_Dance3);
      }
   }
   for(int z=6; z<8; z++) {
        RU.slowmove (array_cal[0] + array_dance3[z][0] , vel_Dance3);
        RL.slowmove (array_cal[1] + array_dance3[z][1] , vel_Dance3);
        LU.slowmove (array_cal[2] + array_dance3[z][2] , vel_Dance3);
        LL.slowmove (array_cal[3] + array_dance3[z][3] , vel_Dance3);
        delay(delay_Dance3);
      }
}

void Dancing4()
{
   for(int z=0; z<num_dance4; z++) {
      if ( z > 17) {
        vel_Dance4 = 10;
        delay_Dance4 = 1500;
        }
      else {
        vel_Dance4 = 40;
        delay_Dance4 = 400;
        }

      RU.slowmove (array_cal[0] + array_dance4[z][0] , vel_Dance4);
      RL.slowmove (array_cal[1] + array_dance4[z][1] , vel_Dance4);
      LU.slowmove (array_cal[2] + array_dance4[z][2] , vel_Dance4);
      LL.slowmove (array_cal[3] + array_dance4[z][3] , vel_Dance4);
      delay(delay_Dance4);
   }
}

void Dancing5()
{
   for(int x = 0; x < 3; x++) {
      for(int z=0; z<5; z++) {
        RU.slowmove (array_cal[0] + array_dance5[z][0] , vel_Dance5);
        RL.slowmove (array_cal[1] + array_dance5[z][1] , vel_Dance5);
        LU.slowmove (array_cal[2] + array_dance5[z][2] , vel_Dance5);
        LL.slowmove (array_cal[3] + array_dance5[z][3] , vel_Dance5);
        delay(delay_Dance5);
      }
   }

   for(int x = 0; x < 2; x++) {
      for(int z=0; z<4; z++) {
        RU.slowmove (array_cal[0] + array_dance5[z][0] , 30);
        RL.slowmove (array_cal[1] + array_dance5[z][1] , 30);
        LU.slowmove (array_cal[2] + array_dance5[z][2] , 30);
        LL.slowmove (array_cal[3] + array_dance5[z][3] , 30);
        delay(550);
      }
   }

   for(int x = 0; x < 3; x++) {
      for(int z=4; z<10; z++) {
        RU.slowmove (array_cal[0] + array_dance5[z][0] , vel_Dance5);
        RL.slowmove (array_cal[1] + array_dance5[z][1] , vel_Dance5);
        LU.slowmove (array_cal[2] + array_dance5[z][2] , vel_Dance5);
        LL.slowmove (array_cal[3] + array_dance5[z][3] , vel_Dance5);
        delay(300);
      }
   }
```

```
    for(int z=10; z<12; z++) {
          RU.slowmove (array_cal[0] + array_dance5[z][0] , 10);
          RL.slowmove (array_cal[1] + array_dance5[z][1] , 10);
          LU.slowmove (array_cal[2] + array_dance5[z][2] , 10);
          LL.slowmove (array_cal[3] + array_dance5[z][3] , 10);
          delay(1500);
       }
}

void Dancing6()
{
   const int array_cal_0 = array_cal[0] + 10 , array_cal_2 = array_cal[2] - 10;

   for(int x = 0; x < 3; x++) {
      for(int z=0; z<6; z++) {
          RU.slowmove (array_cal_0 + array_dance6[z][0] , vel_Dance6);
          RL.slowmove (array_cal[1] + array_dance6[z][1] , vel_Dance6);
          LU.slowmove (array_cal_2 + array_dance6[z][2] , vel_Dance6);
          LL.slowmove (array_cal[3] + array_dance6[z][3] , vel_Dance6);
          delay(delay_Dance6);
      }
   }

   for(int x = 0; x < 3; x++) {
      for(int z=6; z<12; z++) {
          RU.slowmove (array_cal_0 + array_dance6[z][0] , 40);
          RL.slowmove (array_cal[1] + array_dance6[z][1] , 40);
          LU.slowmove (array_cal_2 + array_dance6[z][2] , 40);
          LL.slowmove (array_cal[3] + array_dance6[z][3] , 40);
          delay(400);
      }
   }

   for(int x = 0; x < 3; x++) {
      for(int z=12; z<18; z++) {
          RU.slowmove (array_cal_0 + array_dance6[z][0] , vel_Dance6);
          RL.slowmove (array_cal[1] + array_dance6[z][1] , vel_Dance6);
          LU.slowmove (array_cal_2 + array_dance6[z][2] , vel_Dance6);
          LL.slowmove (array_cal[3] + array_dance6[z][3] , vel_Dance6);
          delay(delay_Dance6);
      }
   }
   for(int x = 0; x < 3; x++) {
      for(int z=18; z<24; z++) {
          RU.slowmove (array_cal[0] + array_dance6[z][0] , vel_Dance6);
          RL.slowmove (array_cal[1] + array_dance6[z][1] , vel_Dance6);
          LU.slowmove (array_cal[2] + array_dance6[z][2] , vel_Dance6);
          LL.slowmove (array_cal[3] + array_dance6[z][3] , vel_Dance6);
          delay(delay_Dance6);
      }
   }

   for(int x = 0; x < 3; x++) {
      for(int z=24; z<30; z++) {
          RU.slowmove (array_cal[0] + array_dance6[z][0] , vel_Dance6);
          RL.slowmove (array_cal[1] + array_dance6[z][1] , vel_Dance6);
          LU.slowmove (array_cal[2] + array_dance6[z][2] , vel_Dance6);
          LL.slowmove (array_cal[3] + array_dance6[z][3] , vel_Dance6);
          delay(delay_Dance6);
      }
   }
   for(int z=30; z<32; z++) {
          RU.slowmove (array_cal[0] + array_dance5[z][0] , 10);
          RL.slowmove (array_cal[1] + array_dance5[z][1] , 10);
          LU.slowmove (array_cal[2] + array_dance5[z][2] , 10);
          LL.slowmove (array_cal[3] + array_dance5[z][3] , 10);
          delay(1500);
       }
}
```

```
void setup()
{
#ifdef INSTALL
   Servo_Init();

   RU.slowmove (90 , vel);
   RL.slowmove (90 , vel);
   LU.slowmove (90 , vel);
   LL.slowmove (90 , vel);
   while(1);
#endif

#ifdef CALIBRATION
   Servo_Init();
   Adjust();

   RL.slowmove (array_cal[1] , vel);
   LL.slowmove (array_cal[3] , vel);
   delay(2000);
   while(1);
#endif

#ifdef RUN
   Servo_Init();
   Adjust();

   RL.slowmove (array_cal[1] , vel);
   LL.slowmove (array_cal[3] , vel);
   delay(2000);
#endif
}

void loop()
{
   Dancing1_2();
   delay(500);
   Dancing3(5,20,400);
   delay(500);
   Dancing4();
   delay(500);
   Dancing5();
   delay(500);
   Dancing6();
   delay(500);
   while(1);

}
```

Lección 9 Robot de control de aplicaciones

Acerca de esta lección:

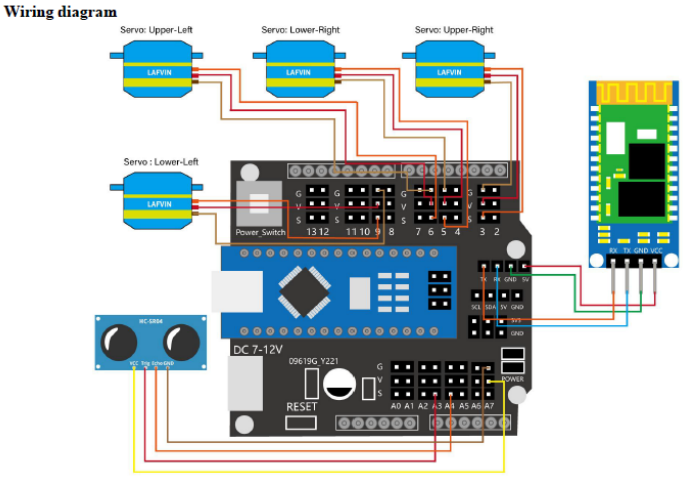
Esta lección se divide en dos partes, primero aprende la prueba de comunicación del módulo Bluetooth y luego aprende la aplicación para controlar el robot brazo.

1) prueba de Bluetooth

Introducción:

El HC06 es un módulo Bluetooth de puerto serie que tiene una modulación de 3 Mbps Bluetooth V2.0 + EDR (velocidad de datos mejorada) con Transceptor de radio completo de 2.4GHz y banda base. Utiliza CSR Blue core 04-Sistema Bluetooth de un solo chip externo con tecnología CMOS y con AFH (función de salto de frecuencia adaptable).

El módulo Bluetooth HC-06 a Nano:
VCC >>>> 5v
GND >>>> GND
TXD >>>> RX
RXD >>>> TX



Subir código

Después del cableado, abra el programa en la carpeta de códigos - Lesson_9_Bluetooth_Test y haga clic en CARGAR para cargar el programa.Atención: el módulo bluetooth debe extraerse antes de cargar el programa cada vez, o no se podrá cargar el programa. Después de que el código se haya cargado correctamente en la placa Nano, vuelva a conectar el Bluetooth.

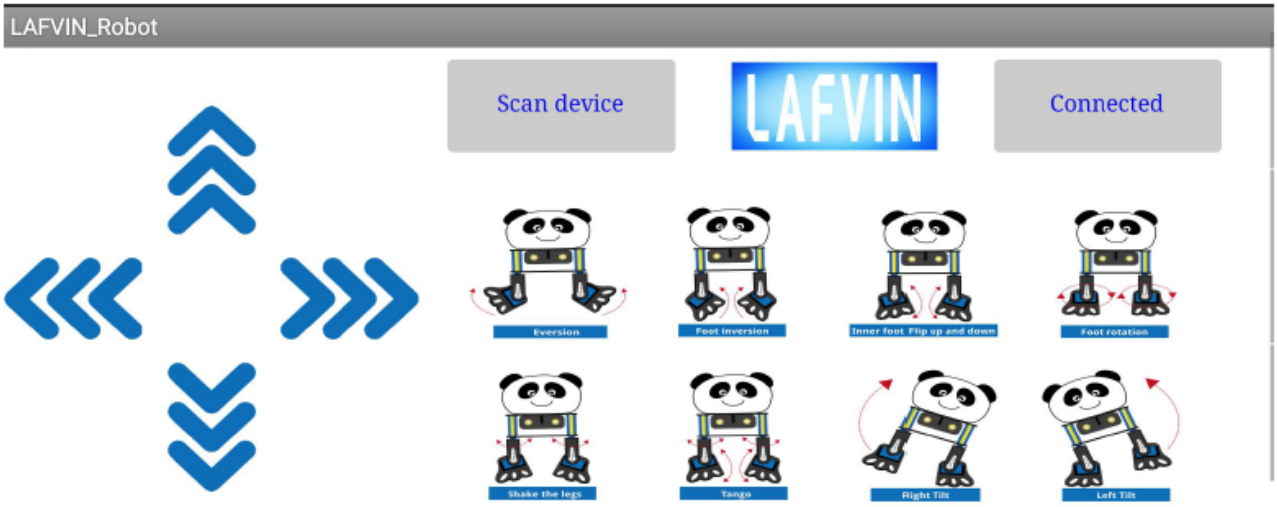
Lesson_9_Bluetooth_Test

```
char val;
int incomingByte = 0;          // Received data byte
String inputString = "";         // Used to store received content
boolean newLineReceived = false; //  Previous data end flag
boolean startBit  = false;  //Acceptance Agreement Start Sign
int num_reveice=0;
void setup()
{
  Serial.begin(9600); //  set the baud rate to 9600
}

void loop()
{
while (Serial.available())
  {
    incomingByte = Serial.read();             //One byte by byte, the next sentence is read into a string array to form a
completed packet
    if (incomingByte == '%')
    {
      num_reveice = 0;
      startBit = true;
    }
    if (startBit == true)
    {
      num_reveice++;
      inputString += (char) incomingByte;
    }
    if (startBit == true && incomingByte == '#')
    {
      newLineReceived = true;
      startBit = false;
```

```
    }

    if(num_reveice >= 20)
    {
      num_reveice = 0;
      startBit = false;
      newLineReceived = false;
      inputString = "";
    }
  }

if(newLineReceived)
{
     Serial.println(inputString);
     inputString = "";   // clear the string
     newLineReceived = false;
}
}
```

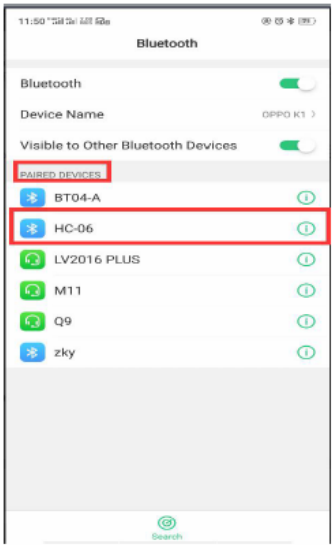
Instrucciones para el uso de la aplicación

En primer lugar, descargue el archivo LAFVIN_Robot.apk de la carpeta a su teléfono móvil e instálelo en un software de aplicación.



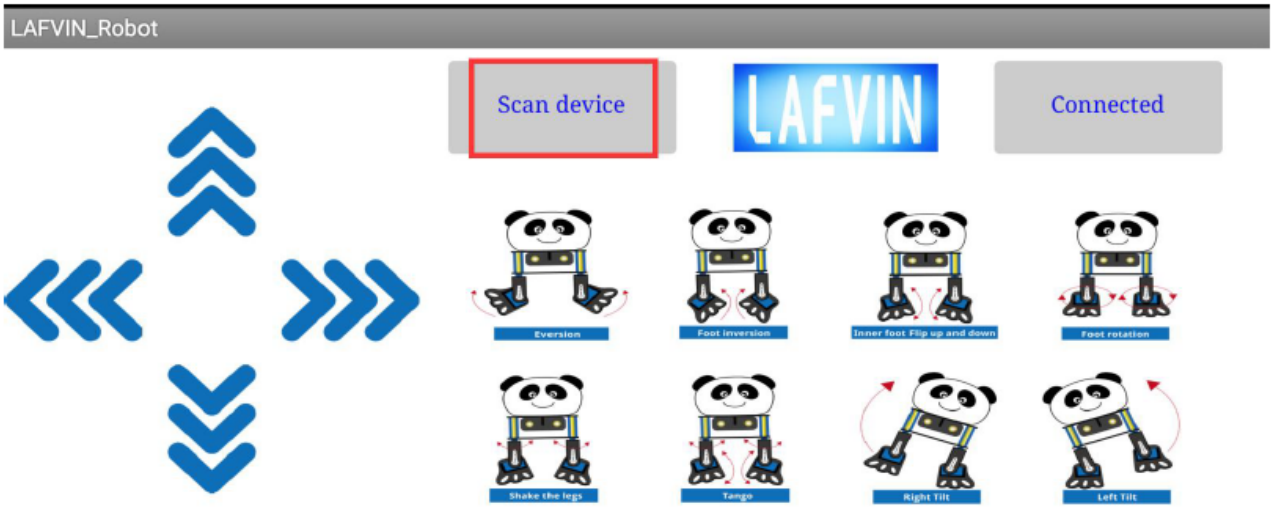
Luego asegúrese de que el módulo Bluetooth esté conectado. Empareje su teléfono con HC-06. para hacer esto, vaya a Configuración-> Bluetooth-> Escanear dispositivo-> seleccione HC-06 y emparejarlo. El código de acceso para emparejar es '1234'. Abra el software del terminal Bluetooth, vaya a las opciones y seleccione la opción 'conectar un dispositivo - seguro'. Eso eso pida el código de acceso, ingrese 1234. Si su teléfono está conectado al módulo Bluetooth, verá un dispositivo utilizable llamado HC-06 en el PAIRED

DISPOSITIVOS (como se muestra a continuación). Si el HC-06 no aparece en los DISPOSITIVOS

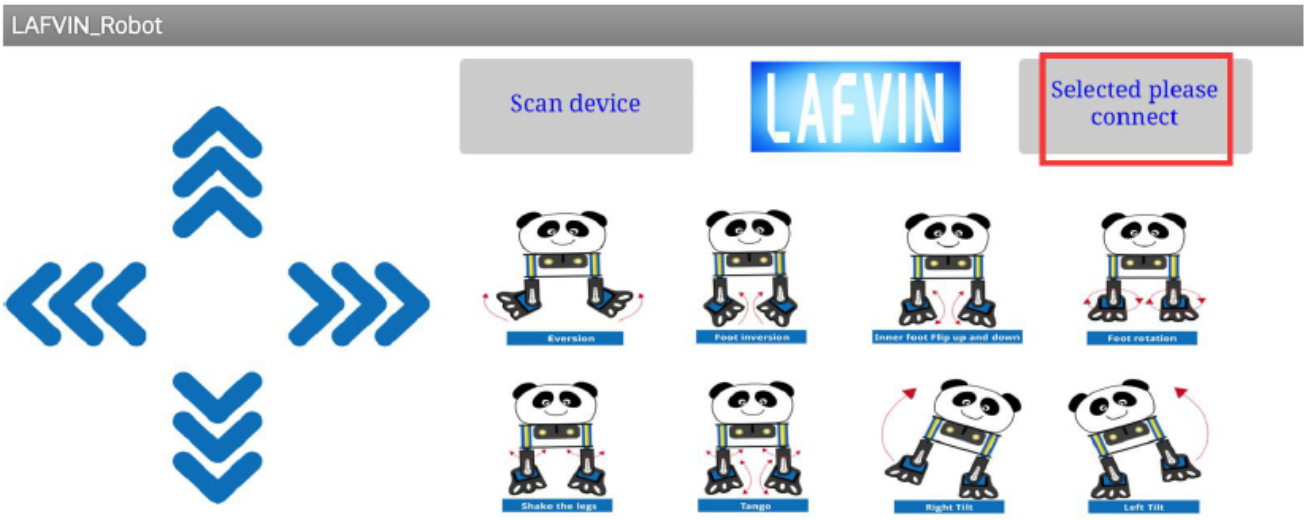


EMPAREJADOS, vuelva a operar los pasos anteriores.

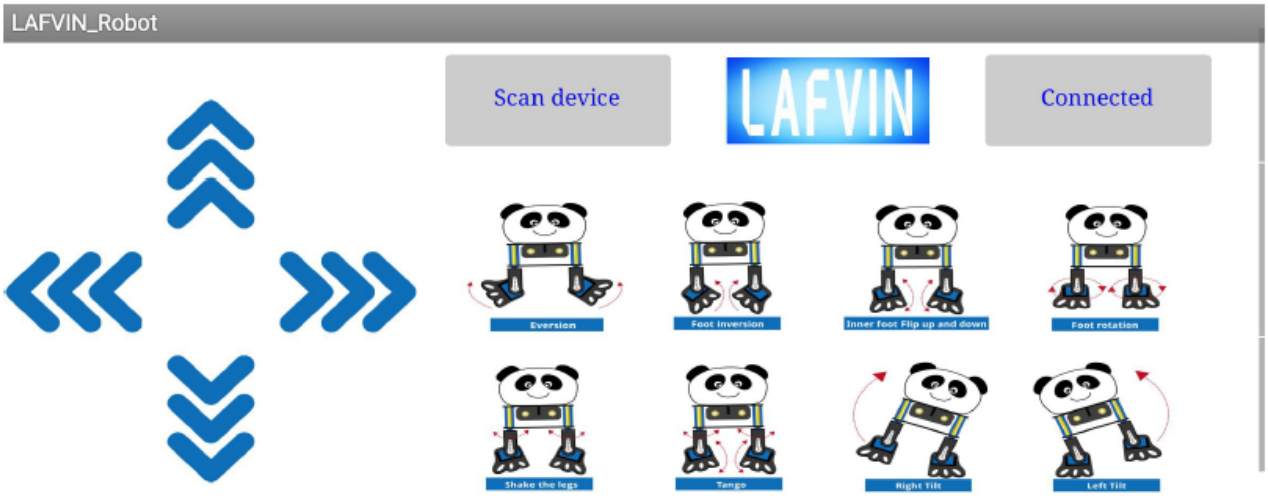
Después de completar los pasos anteriores, abrimos la aplicación LAFVIN_Robot.



Luego haga clic en el dispositivo de escaneo y el HC-06 aparecerá en nuestros resultados de escaneo. Seleccione HC-06.

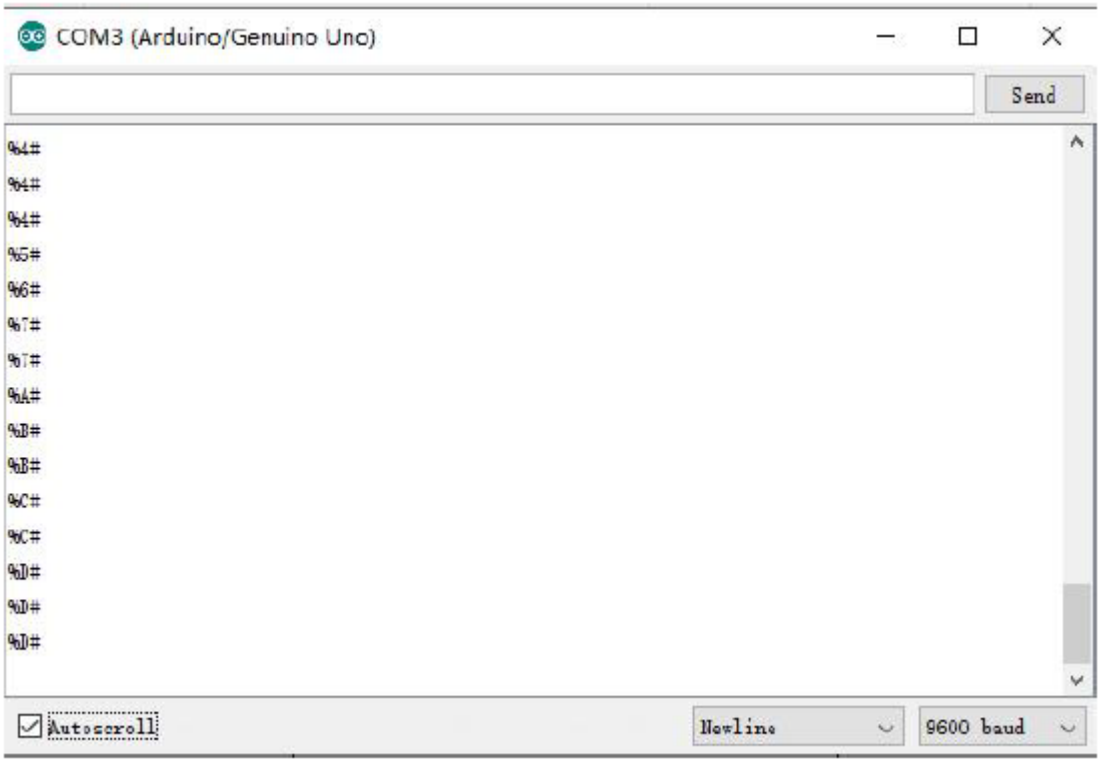


Después de seleccionar el dispositivo hc-06, aparecerá seleccionado por favor conectado en el cuadro rojo de la aplicación, haga clic en el botón seleccionado por favor conectado y espere 5 segundos. El botón muestra el conectado, lo que significa que la aplicación está conectada al módulo Bluetooth correctamente. (Nota: cuando el módulo Bluetooth no está conectado correctamente, la luz LED roja seguirá parpadeando. Cuando la conexión es exitosa, el LED rojo la luz permanecerá encendida.)



Instrucciones de prueba

Cargue el código, conecte bien y encienda. Después de conectar la APLICACIÓN Bluetooth, abra el monitor en serie y configure la velocidad en baudios en 9600, presione presionando la tecla de control en la APLICACIÓN, debería ver el valor correspondiente. Mostrado a continuación.

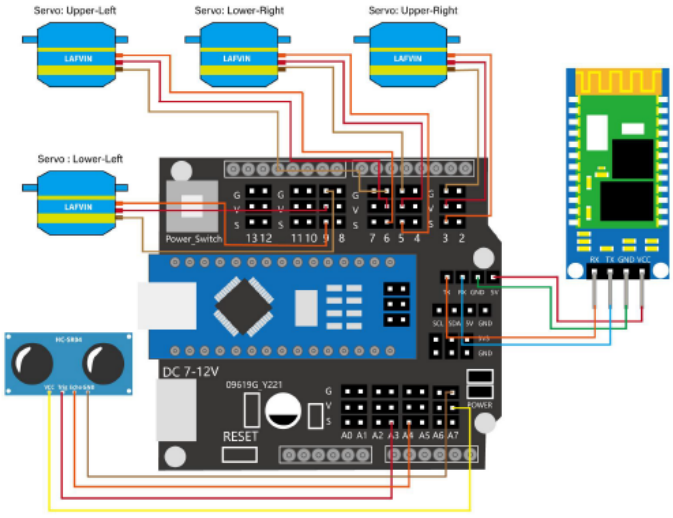


Por ejemplo,% 4 # es el comando de comunicación de Bluetooth y la aplicación, '%' es el encabezado del marco del comando de comunicación, lo que significa que Bluetooth y la aplicación solo aceptan la instrucción que comienza con '%', y '4' representa el contenido de la instrucción, que se utiliza para determinar el robot de control. Qué acción completar. '#' es el final del comando de comunicación, lo que indica que la instrucción se ha enviado.

2) Robot de control de Bluetooth

Después de aprender a enviar instrucciones al módulo Bluetooth, aprenda a controlar el robot a través de la aplicación.

Diagrama de cableado



Instrucciones de prueba

Abra el programa en la carpeta de códigos - Lesson_9_APP_Control_Robot y haga clic en CARGAR para cargar el programa. Atención: ell módulo bluetooth debe extraerse antes de cargar el programa cada vez, o no se podrá cargar el programa.

Lesson_9_APP_Control_Robot

```
#include "VarSpeedServo.h"  //include the VarSpeedServo library
//#include <Servo.h>

VarSpeedServo RU;  //Right Upper
VarSpeedServo RL;  //Right Lower
VarSpeedServo LU;  //Left Upper
VarSpeedServo LL;  //Left Lower


//String val="";
int incomingByte = 0;          // Received data byte
String inputString = "";         // Used to store received content
boolean newLineReceived = false; // Previous data end mark
boolean startBit  = false;  //Agreement start sign
int num_reveice=0;



//vel(min), delay_Forward(max) = (5, 2000)
const int vel = 40, vel_Back = 40;//15,vel_Right= 30,vel_Left= 30;                  //vel(mid), delay_Forward(mid) = (20, 750)
const int delay_Forward = 300, delay_Back = 450; //750    //vel(max), delay_Forward(min)= (256, 50)
const int delay_Right = 200, delay_Left = 200;                                      //wonderful ---> (10, 700) (50, 500) (100,
100) (100, 300) (100, 500)

int vel_Dance1 = 30,    vel_Dance2 = 25,    vel_Dance3 = 40;
int delay_Dance1 = 300, delay_Dance2 = 750, delay_Dance3 = 200;

int vel_Dance4 = 40,    vel_Dance5 = 40,    vel_Dance6 = 30;
int delay_Dance4 = 400, delay_Dance5 = 400, delay_Dance6 = 500;

const int array_cal[4] = {81,80,90,92};
//const int array_cal[4] = {77,82,95,90};

int RU_Degree = 0, LU_Degree = array_cal[2] + 5;

const int num_dance1 = 10;
const int array_dance1[num_dance1][4] =
{
//slide to the left 0-4
    {0,-20,0,0},
    {0,-40,0,20},
    {0,-20,0,40},
    {0,0,0,20},
    {0,0,0,0},

//slide to the right 5-9
    {0,0,0,20},
    {0,-20,0,40},
    {0,-40,0,20},
    {0,-20,0,0},
    {0,0,0,0},
};

const int num_dance2 = 32;
const int array_dance2[num_dance2][4] =
{
//left foot support 0-15
    {20,0,40,0},
    {20,-30,40,-30},
    {20,-30,10,-30},
    {20,-30,40,-30},
    {20,-30,10,-30},

    {20,-30,40,-30},
```

```
    {20,0,40,-30},
    {20,80,40,-30},
    {20,0,40,-30},
    {20,-80,40,-30},
    {20,0,40,-30},
    {20,80,40,-30},
    {20,0,40,-30},
    {20,-30,40,-30},
    {20,0,40,0},
    {0,0,0,0},

//right foot support 16-31
    {-40,0,-20,0},
    {-40,40,-20,30},
    {-20,40,-20,30},
    {-40,40,-20,30},
    {-20,40,-20,30},

    {-40,40,-20,30},
    {-40,40,-20,0},
    {-40,40,-20,-80},
    {-40,40,-20,0},
    {-40,40,-20,80},
    {-40,40,-20,0},
    {-40,40,-20,-80},
    {-40,40,-20,0},
    {-40,40,-20,30},
    {-40,0,-20,0},
    {0,0,0,0},
};

const int num_dance3 = 8;       //Small broken
const int array_dance3[num_dance3][4] =
{
    {0,-40,0,0},
    {20,-30,20,20},
    {40,0,40,30},
    {0,0,0,40},
    {-20,-20,-20,30},
    {-40,-30,-40,0},

    {0,-40,0,0},
    {0,0,0,0},
};

const int num_dance4 = 20;      //In-situ ups and downs, increasing in angle
const int array_dance4[num_dance4][4] =
{
    {0,-20,0,20},
    {0,0,0,0},
    {0,-20,0,20},
    {0,0,0,0},
    {0,-20,0,20},
    {0,0,0,0},
    {0,-20,0,20},
    {0,0,0,0},

    {0,-50,0,50},
    {0,0,0,0},
    {0,-50,0,50},
    {0,0,0,0},
    {0,-50,0,50},
    {0,0,0,0},
    {0,-50,0,50},
    {0,0,0,0},

    {0,-40,0,40},
    {0,-50,0,50},
    {0,-60,0,60},
```

```
    {0,0,0,0,},
};

const int num_hold_up = 20;
const int array_hold_up[num_hold_up][4] =
{
    {0,20,0,-20},
    {0,0,0,0},
    {0,20,0,-20},
    {0,0,0,0},
    {0,20,0,-20},
    {0,0,0,0},
    {0,20,0,-20},
    {0,0,0,0},

    {0,50,0,-50},
    {0,0,0,0},
    {0,50,0,-50},
    {0,0,0,0},
    {0,50,0,-50},
    {0,0,0,0},
    {0,50,0,-50},
    {0,0,0,0},

    {0,40,0,-40},
    {0,50,0,-50},
    {0,60,0,-60},
    {0,0,0,0,},
};




const int num_dance5 = 17;
const int array_dance5[num_dance5][4] =
{
    {35,0,15,0},
    {35,30,15,30},
    {-35,30,15,30},
    {-20,0,15,0},
    {0,0,0,0},

    {0,-40,0,40},    //Inner flip foot
    {-30,-40,-20,40},
    {0,-40,0,40},
    {20,-40,30,40},


    {0,-40,0,40},    //In place to rise and fall
    {20,-40,-20,40},
    {20,-20,-20,20},
    {20,0,-20,0},
    {-20,-10,20,10},
    {-10,-30,10,30},

    {0,-40,0,40},
    {0,0,0,0},
};

const int num_dance6 = 32;
const int array_dance6[num_dance6][4] =
{
    {0,-40,0,-20},        //Outer Eight Steps - Advance
    {25,-40,18,-20},
    {25,0,18,0},
```

```
    {0,20,0,40},
    {-18,20,-25,40},
    {-18,0,-25,0},

    {0,-40,0,-20},        //Outer Eight Steps - Stepping In Place
    {25,-40,18,-20},
    {0,0,0,0},
    {0,20,0,40},
    {-18,20,-25,40},
    {0,0,0,0},

    {0,-40,0,-20},        //Outer Eight Steps - Backward
    {-25,-40,-18,-20},
    {-25,0,-18,0},
    {0,20,0,40},
    {18,20,25,40},
    {18,0,25,0},

    {0,-40,0,-20},        //Small step - forward
    {30,-40,30,-20},
    {30,0,30,0},
    {0,20,0,40},
    {-30,20,-30,40},
    {-30,0,-30,0},

    {0,-40,0,-20},        //Small step - back
    {-30,-40,-30,-20},
    {-30,0,-30,0},
    {0,20,0,40},
    {30,20,30,40},
    {30,0,30,0},

    {15,0,15,0},
    {0,0,0,0},
};

const int num1 = 8;
const int array_forward[num1][4] =
{
    {0,-40,0,-20},        //Pace - advance
    {30,-40,30,-20},
    {30,0,30,0},
    {0,20,0,40},
    {-30,20,-30,40},
    {-30,0,-30,0},
    {-15,0,-15,0},
    {0,0,0,0},
};

const int num2 = 8;
const int array_turn[num2][4] =
{

    {0,-40,0,-20},        //Step-back
    {-30,-40,-30,-20},
    {-30,0,-30,0},
    {0,20,0,40},
    {30,20,30,40},
    {30,0,30,0},
    {15,0,15,0},
    {0,0,0,0},
};


const int num3 = 10;
const int array_turnright[num3][4] =
{      //Turn right
//    {0,1,0,15},
//    {0,10,0,35},
```

```
//    {0,15,0,50},
//    {-30,15,0,50},
//    {-30,5,0,20},
//    {-30,0,0,0},
//    {-15,0,0,0},
//    {0,0,0,0},


    {0,10,0,15},
    {0,10,0,35},
    {0,15,0,60},
    {-30,20,0,60},
    {-30,5,0,20},
    {-30,0,0,0},
    {-25,0,0,0},
    {-15,0,0,0},
    {0,0,0,0},
};



const int num4 = 10;
const int array_turnleft[num4][4] =
{
  //Turn left
//    {0,-15,0,-1},
//    {0,-35,0,-10},
//    {0,-50,0,-15},
//    {0,-50,30,-15},
//    {0,-20,30,-5},
//    {0,0,30,0},
//    {0,0,25,0},
//    {0,0,15,0},
//    {0,0,0,0},

    {0,-15,0,-10},
    {0,-35,0,-10},
    {0,-60,0,-15},
    {0,-60,30,-20},
    {0,-20,30,-5},
    {0,0,30,0},
    {0,0,25,0},
    {0,0,15,0},
    {0,0,0,0},


};
//#define INSTALL
//#define CALIBRATION
#define RUN

void Servo_Init()
{
    RU.attach(3);   // Connect the signal wire of the upper-right servo to pin 9
    RL.attach(5);   // Connect the signal wire of the lower-right servo to pin 10
    LU.attach(6);   // Connect the signal wire of the upper-left  servo to pin 11
    LL.attach(9);   // Connect the signal wire of the lower-left  servo to pin 12
}

void Adjust()                         // Avoid the servo's fast spinning in initialization
{                                   // RU,LU goes from array_cal[0] - 5 ,array_cal[2] + 5 degrees to array_cal[0],array_cal[2]
degrees
    for(RU_Degree = array_cal[0] - 5; RU_Degree <= array_cal[0]; RU_Degree += 1) {
        RU.write(RU_Degree);              // in steps of 1 degree
        LU.write(LU_Degree--);            // tell servo to go to RU_Degreeition, LU_Degreeition in variable 'RU_Degree',
'LU_Degree'
        delay(15);                        // waits 15ms for the servo to reach the RU_Degreeition
    }
}
```

```
void Slide_2_Left(int times)
{
   for(int time1 = 0; time1 < times; time1++) {
      for(int z=0; z<5; z++) {
         RU.slowmove (array_cal[0] + array_dance1[z][0] , vel_Dance1);
         RL.slowmove (array_cal[1] + array_dance1[z][1] , vel_Dance1);
         LU.slowmove (array_cal[2] + array_dance1[z][2] , vel_Dance1);
         LL.slowmove (array_cal[3] + array_dance1[z][3] , vel_Dance1);
         delay(delay_Dance1);
      }
   }
}

void Slide_2_Right(int times)
{
   for(int time1 = 0; time1 < times; time1++) {
      for(int z=5; z<10; z++) {
         RU.slowmove (array_cal[0] + array_dance1[z][0] , vel_Dance1);
         RL.slowmove (array_cal[1] + array_dance1[z][1] , vel_Dance1);
         LU.slowmove (array_cal[2] + array_dance1[z][2] , vel_Dance1);
         LL.slowmove (array_cal[3] + array_dance1[z][3] , vel_Dance1);
         delay(delay_Dance1);
      }
   }
}

void Left_Foot_Support()
{
   for(int z=0; z<16; z++) { //z<12
      if ( z > 5 && z < 14) {   //z(1,10)
         vel_Dance2 = 50;
         delay_Dance2 = 200;
      }
      else {
      vel_Dance2 = 25;
      delay_Dance2 = 750;
      }

      RU.slowmove (array_cal[0] + array_dance2[z][0] , vel_Dance2);
      RL.slowmove (array_cal[1] + array_dance2[z][1] , vel_Dance2);
      LU.slowmove (array_cal[2] + array_dance2[z][2] , vel_Dance2);
      LL.slowmove (array_cal[3] + array_dance2[z][3] , vel_Dance2);
      delay(delay_Dance2);
   }
}

void Right_Foot_Support()
{
   for(int z=16; z<32; z++) { //z<24
      if ( z > 21 && z < 30) {   //z(13,22)
         vel_Dance2 = 50;
         delay_Dance2 = 200;
      }
      else {
      vel_Dance2 = 25;
      delay_Dance2 = 750;
      }

      RU.slowmove (array_cal[0] + array_dance2[z][0] , vel_Dance2);
      RL.slowmove (array_cal[1] + array_dance2[z][1] , vel_Dance2);
      LU.slowmove (array_cal[2] + array_dance2[z][2] , vel_Dance2);
      LL.slowmove (array_cal[3] + array_dance2[z][3] , vel_Dance2);
      delay(delay_Dance2);
   }
}
```

```
void Dancing1_2()
{
   Slide_2_Left(2);
   Left_Foot_Support();

   Slide_2_Right(2);
   Right_Foot_Support();
}

void Dancing3(int Times = 1, int Vel = 40, int Delay = 250, int low = 0, int high = 0)
{
   for(int time3 = 0; time3 < Times; time3++) {
      for(int z=0; z<6; z++) {
         if ( time3 > 1 && time3 < 4) {
         vel_Dance3 = Vel;
         delay_Dance3 = Delay;
         }
         else {
         vel_Dance3 = 40;
         delay_Dance3 = 200;
         }

         RU.slowmove (array_cal[0] + array_dance3[z][0] , vel_Dance3);
         RL.slowmove (array_cal[1] + array_dance3[z][1] , vel_Dance3);
         LU.slowmove (array_cal[2] + array_dance3[z][2] , vel_Dance3);
         LL.slowmove (array_cal[3] + array_dance3[z][3] , vel_Dance3);
         delay(delay_Dance3);
      }
   }
   for(int z=6; z<8; z++) {
         RU.slowmove (array_cal[0] + array_dance3[z][0] , vel_Dance3);
         RL.slowmove (array_cal[1] + array_dance3[z][1] , vel_Dance3);
         LU.slowmove (array_cal[2] + array_dance3[z][2] , vel_Dance3);
         LL.slowmove (array_cal[3] + array_dance3[z][3] , vel_Dance3);
         delay(delay_Dance3);
      }
}


void hold_up()
{    int vel_hold_up = 10;
     int delay_hold_up= 1500;
  for(int z=16; z<20; z++)
  {
       RU.slowmove (array_cal[0] + array_hold_up[z][0] , vel_hold_up);
       RL.slowmove (array_cal[1] + array_hold_up[z][1] , vel_hold_up);
       LU.slowmove (array_cal[2] + array_hold_up[z][2] , vel_hold_up);
       LL.slowmove (array_cal[3] + array_hold_up[z][3] , vel_hold_up);
       delay(delay_hold_up);
   }

}


void large_hold_up()
{  int vel_hold_up = 40;
   int delay_hold_up = 200;
  for(int x = 0; x < 4; x++) {
  for(int z=6; z<16; z++)
  {
       RU.slowmove (array_cal[0] + array_hold_up[z][0] , vel_hold_up);
       RL.slowmove (array_cal[1] + array_hold_up[z][1] , vel_hold_up);
       LU.slowmove (array_cal[2] + array_hold_up[z][2] , vel_hold_up);
       LL.slowmove (array_cal[3] + array_hold_up[z][3] , vel_hold_up);
       delay(delay_hold_up);
   }
  }
}
```

```
void large_Shrink()
{ vel_Dance4 = 40;
  delay_Dance4 = 200;
  for(int x = 0; x < 4; x++) {
  for(int z=6; z<16; z++)
  {
       RU.slowmove (array_cal[0] + array_dance4[z][0] , vel_Dance4);
       RL.slowmove (array_cal[1] + array_dance4[z][1] , vel_Dance4);
       LU.slowmove (array_cal[2] + array_dance4[z][2] , vel_Dance4);
       LL.slowmove (array_cal[3] + array_dance4[z][3] , vel_Dance4);
       delay(delay_Dance4);
    }
  }
}

void Shrink()
{ vel_Dance4 = 10;
  delay_Dance4 = 1500;

  for(int z=16; z<20; z++)
  {
       RU.slowmove (array_cal[0] + array_dance4[z][0] , vel_Dance4);
       RL.slowmove (array_cal[1] + array_dance4[z][1] , vel_Dance4);
       LU.slowmove (array_cal[2] + array_dance4[z][2] , vel_Dance4);
       LL.slowmove (array_cal[3] + array_dance4[z][3] , vel_Dance4);
       delay(delay_Dance4);

  }
}


void small_Shrink()
{ vel_Dance4 = 10;
  delay_Dance4 = 1500;
  for(int z=8; z<16; z++)
  {
       RU.slowmove (array_cal[0] + array_dance4[z][0] , vel_Dance4);
       RL.slowmove (array_cal[1] + array_dance4[z][1] , vel_Dance4);
       LU.slowmove (array_cal[2] + array_dance4[z][2] , vel_Dance4);
       LL.slowmove (array_cal[3] + array_dance4[z][3] , vel_Dance4);
       delay(delay_Dance4);
    }

}

void Dancing4()
{
    for(int z=0; z<num_dance4; z++) {
       if ( z > 17) {
          vel_Dance4 = 10;
          delay_Dance4 = 1500;
          }
       else {
          vel_Dance4 = 40;
          delay_Dance4 = 400;
          }

       RU.slowmove (array_cal[0] + array_dance4[z][0] , vel_Dance4);
       RL.slowmove (array_cal[1] + array_dance4[z][1] , vel_Dance4);
       LU.slowmove (array_cal[2] + array_dance4[z][2] , vel_Dance4);
       LL.slowmove (array_cal[3] + array_dance4[z][3] , vel_Dance4);
       delay(delay_Dance4);
    }
}
```

```
void Turn_inside()
{    for(int x = 0; x < 2; x++) {
       for(int z=5; z<9; z++) {
          RU.slowmove (array_cal[0] + array_dance5[z][0] , 30);
          RL.slowmove (array_cal[1] + array_dance5[z][1] , 30);
          LU.slowmove (array_cal[2] + array_dance5[z][2] , 30);
          LL.slowmove (array_cal[3] + array_dance5[z][3] , 30);
          delay(550);
       }
    }
     for(int z=15; z<17; z++) {
          RU.slowmove (array_cal[0] + array_dance5[z][0] , 30);
          RL.slowmove (array_cal[1] + array_dance5[z][1] , 30);
          LU.slowmove (array_cal[2] + array_dance5[z][2] , 30);
          LL.slowmove (array_cal[3] + array_dance5[z][3] , 30);
          delay(550);
       }

  }


void In_place()
{   for(int x = 0; x < 3; x++) {
       for(int z=9; z<15; z++) {
          RU.slowmove (array_cal[0] + array_dance5[z][0] , vel_Dance5);
          RL.slowmove (array_cal[1] + array_dance5[z][1] , vel_Dance5);
          LU.slowmove (array_cal[2] + array_dance5[z][2] , vel_Dance5);
          LL.slowmove (array_cal[3] + array_dance5[z][3] , vel_Dance5);
          delay(300);
       }
    }

     for(int z=15; z<17; z++) {
          RU.slowmove (array_cal[0] + array_dance5[z][0] , 30);
          RL.slowmove (array_cal[1] + array_dance5[z][1] , 30);
          LU.slowmove (array_cal[2] + array_dance5[z][2] , 30);
          LL.slowmove (array_cal[3] + array_dance5[z][3] , 30);
          delay(550);
       }
  }




void Dancing5()
{
       for(int z=0; z<5; z++) {
          RU.slowmove (array_cal[0] + array_dance5[z][0] , vel_Dance5);
          RL.slowmove (array_cal[1] + array_dance5[z][1] , vel_Dance5);
          LU.slowmove (array_cal[2] + array_dance5[z][2] , vel_Dance5);
          LL.slowmove (array_cal[3] + array_dance5[z][3] , vel_Dance5);
          delay(delay_Dance5);

    }

    for(int x = 0; x < 2; x++) {
       for(int z=5; z<9; z++) {
          RU.slowmove (array_cal[0] + array_dance5[z][0] , 30);
          RL.slowmove (array_cal[1] + array_dance5[z][1] , 30);
          LU.slowmove (array_cal[2] + array_dance5[z][2] , 30);
          LL.slowmove (array_cal[3] + array_dance5[z][3] , 30);
          delay(550);
       }
    }

    for(int x = 0; x < 3; x++) {
       for(int z=9; z<15; z++) {
          RU.slowmove (array_cal[0] + array_dance5[z][0] , vel_Dance5);
```

```
        RL.slowmove (array_cal[1] + array_dance5[z][1] , vel_Dance5);
        LU.slowmove (array_cal[2] + array_dance5[z][2] , vel_Dance5);
        LL.slowmove (array_cal[3] + array_dance5[z][3] , vel_Dance5);
        delay(300);
      }
   }
   for(int z=15; z<17; z++) {
        RU.slowmove (array_cal[0] + array_dance5[z][0] , 10);
        RL.slowmove (array_cal[1] + array_dance5[z][1] , 10);
        LU.slowmove (array_cal[2] + array_dance5[z][2] , 10);
        LL.slowmove (array_cal[3] + array_dance5[z][3] , 10);
        delay(1500);
      }
}




void Walking_in_place()
{ for(int x = 0; x < 3; x++) {
      for(int z=6; z<12; z++) {
        RU.slowmove (array_cal[0] + array_dance6[z][0] , 40);
        RL.slowmove (array_cal[1] + array_dance6[z][1] , 40);
        LU.slowmove (array_cal[2] + array_dance6[z][2] , 40);
        LL.slowmove (array_cal[3] + array_dance6[z][3] , 40);
        delay(400);
      }
  }

}

void Dancing6()
{
   const int array_cal_0 = array_cal[0] + 10 , array_cal_2 = array_cal[2] - 10;

   for(int x = 0; x < 3; x++) {
      for(int z=0; z<6; z++) {
        RU.slowmove (array_cal_0 + array_dance6[z][0] , vel_Dance6);
        RL.slowmove (array_cal[1] + array_dance6[z][1] , vel_Dance6);
        LU.slowmove (array_cal_2 + array_dance6[z][2] , vel_Dance6);
        LL.slowmove (array_cal[3] + array_dance6[z][3] , vel_Dance6);
        delay(delay_Dance6);
      }
   }

   for(int x = 0; x < 3; x++) {
      for(int z=6; z<12; z++) {
        RU.slowmove (array_cal_0 + array_dance6[z][0] , 40);
        RL.slowmove (array_cal[1] + array_dance6[z][1] , 40);
        LU.slowmove (array_cal_2 + array_dance6[z][2] , 40);
        LL.slowmove (array_cal[3] + array_dance6[z][3] , 40);
        delay(400);
      }
   }

   for(int x = 0; x < 3; x++) {
      for(int z=12; z<18; z++) {
        RU.slowmove (array_cal_0 + array_dance6[z][0] , vel_Dance6);
        RL.slowmove (array_cal[1] + array_dance6[z][1] , vel_Dance6);
        LU.slowmove (array_cal_2 + array_dance6[z][2] , vel_Dance6);
        LL.slowmove (array_cal[3] + array_dance6[z][3] , vel_Dance6);
        delay(delay_Dance6);
      }
   }
   for(int x = 0; x < 3; x++) {
      for(int z=18; z<24; z++) {
        RU.slowmove (array_cal[0] + array_dance6[z][0] , vel_Dance6);
        RL.slowmove (array_cal[1] + array_dance6[z][1] , vel_Dance6);
```

```
        LU.slowmove (array_cal[2] + array_dance6[z][2] , vel_Dance6);
        LL.slowmove (array_cal[3] + array_dance6[z][3] , vel_Dance6);
        delay(delay_Dance6);
      }
   }

   for(int x = 0; x < 3; x++) {
      for(int z=24; z<30; z++) {
        RU.slowmove (array_cal[0] + array_dance6[z][0] , vel_Dance6);
        RL.slowmove (array_cal[1] + array_dance6[z][1] , vel_Dance6);
        LU.slowmove (array_cal[2] + array_dance6[z][2] , vel_Dance6);
        LL.slowmove (array_cal[3] + array_dance6[z][3] , vel_Dance6);
        delay(delay_Dance6);
      }
   }
   for(int z=30; z<32; z++) {
        RU.slowmove (array_cal[0] + array_dance5[z][0] , 10);
        RL.slowmove (array_cal[1] + array_dance5[z][1] , 10);
        LU.slowmove (array_cal[2] + array_dance5[z][2] , 10);
        LL.slowmove (array_cal[3] + array_dance5[z][3] , 10);
        delay(1500);
      }
}

void Forward()
{
   for(int x=0; x<num1; x++) {
      RU.slowmove (array_cal[0] + array_forward[x][0] , vel);
      RL.slowmove (array_cal[1] + array_forward[x][1] , vel);
      LU.slowmove (array_cal[2] + array_forward[x][2] , vel);
      LL.slowmove (array_cal[3] + array_forward[x][3] , vel);
      delay(delay_Forward);
   }
}

void Backward()
{
      for(int y=0; y<num2; y++) {
        RU.slowmove (array_cal[0] + array_turn[y][0] , vel_Back);
        RL.slowmove (array_cal[1] + array_turn[y][1] , vel_Back);
        LU.slowmove (array_cal[2] + array_turn[y][2] , vel_Back);
        LL.slowmove (array_cal[3] + array_turn[y][3] , vel_Back);
        delay(delay_Back);
      }

}
void turn_left()
{
  for(int x=0; x<5; x++)
  {
    for(int z=0; z<num4; z++)
      {
        RU.slowmove (array_cal[0] + array_turnleft[z][0] , vel);
        RL.slowmove (array_cal[1] + array_turnleft[z][1] , vel);
        LU.slowmove (array_cal[2] + array_turnleft[z][2] , vel);
        LL.slowmove (array_cal[3] + array_turnleft[z][3] , vel);
        delay(delay_Left);
      }
  }
}


void turn_right()
{
  for(int x=0; x<5; x++)
  {
    for(int z=0; z<num3; z++)
      {
        RU.slowmove (array_cal[0] + array_turnright[z][0] ,vel);
```

```
          RL.slowmove (array_cal[1] + array_turnright[z][1] ,vel);
          LU.slowmove (array_cal[2] + array_turnright[z][2] ,vel);
          LL.slowmove (array_cal[3] + array_turnright[z][3] ,vel);
          delay(delay_Right);
      }
  }
}

void setup()
{
   Serial.begin(9600); //  set the baud rate to 9600
#ifdef INSTALL
   Servo_Init();

   RU.slowmove (90 , vel);
   RL.slowmove (90 , vel);
   LU.slowmove (90 , vel);
   LL.slowmove (90 , vel);
   while(1);
#endif

#ifdef CALIBRATION
   Servo_Init();
   Adjust();

   RL.slowmove (array_cal[1] , vel);
   LL.slowmove (array_cal[3] , vel);
   delay(2000);
   while(1);
#endif

#ifdef RUN
   Servo_Init();
   Adjust();

   RL.slowmove (array_cal[1] , vel);
   LL.slowmove (array_cal[3] , vel);
   delay(2000);
#endif
}

void loop()
{
//    Dancing1_2();
//    delay(500);
//    Dancing3(5,20,400);
//    delay(500);
//    Dancing4();
//    delay(500);
//    Dancing5();
//    delay(500);
//    Dancing6();
//    delay(500);
//
// while (Serial.available() > 0)
//    {
//        val+= char(Serial.read());
//        delay(2);
//    }

//    if(Serial.available())   // if receive the data
//  {
//    val=Serial.read();    // read the received data
//    Serial.println(val);
//

while (Serial.available())
  {
   incomingByte = Serial.read();            //One byte is read one byte, and the next sentence is read into the string array to
```

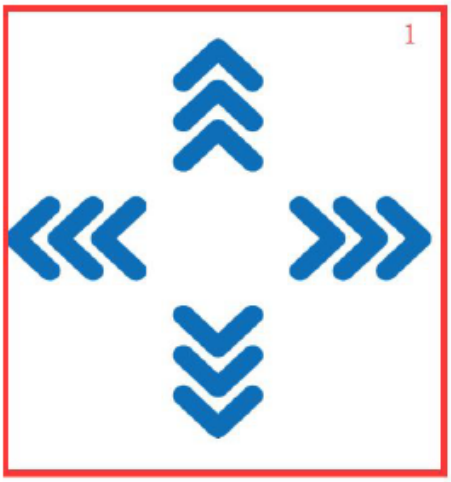
```
form a completed packet.
    if (incomingByte == '%')
    {
      num_reveice = 0;
      startBit = true;
    }
    if (startBit == true)
    {
      num_reveice++;
      inputString += (char) incomingByte;     // Full-duplex serial port can be added without delay, half-duplex is added
    }
    if (startBit == true && incomingByte == '#')
    {
      newLineReceived = true;
      startBit = false;
    }

    if(num_reveice >= 20)
    {
      num_reveice = 0;
      startBit = false;
      newLineReceived = false;
      inputString = "";
    }
  }



if(newLineReceived)
{
    switch(inputString[1])
    { case 'A':  hold_up();;  break;
      case 'B':  Shrink();  break;
      case 'C':  Slide_2_Right(3);  break;
      case 'D':  In_place();  break;
      case 'E':  Turn_inside();  break;   // execute the corresponding function when receive the value
      case 'F':  Dancing3(5,20,400);  break;//
      case 'G':  Left_Foot_Support();  break;
      case 'H':  Right_Foot_Support();  break;
      case '4':  Forward();;  break;
      case '5':  turn_left();  break;
      case '6':  turn_right();  break;
      case '7':  Backward();  break;
      default:break;
    }

      inputString = "";   // clear the string
      newLineReceived = false;
}

}
```

Si se ha cargado correctamente en la placa Nano, vuelva a conectar el Bluetooth. Repita los pasos anteriores con respecto a la conexión de la aplicación al Bluetooth módulo Ahora que la aplicación se ha conectado correctamente a Bluetooth, podemos comenzar a controlar el robot a través de la aplicación.

Los botones del cuadro rojo 1 se utilizan para controlar que el robot avance, retroceda, gire a la izquierda y gire a la derecha.

Hay un total de 8 botones en el cuadro rojo 2, y el patrón de cada botón corresponde a una acción de baile del robot.

LAFVIN_Robot



Scan device

LAFVIN

Connected