Sistemas con Procesadores Integrados

Proyecto. *Raspberry Pi*

Elena Álvarez Castro

Máster en Ingeniería Electrónica Industrial

Universidad de Granada

Curso 2019/2020

Contenido

1. Descripción General del Proyecto

Sistema de control remoto del robot ZUMO, mediante el joystick del Sense HAT para Raspberry Pi. Este sistema de control permite indicar cuatro direcciones (derecha, izquierda, delante y atrás), parar y variar la velocidad entre un valor mínimo y un valor máximo.

Monitorización web del estado de los sensores del robot ZUMO y de los sensores integrados en el Sense HAT.

Sensores Sense HAT	Sensores Robot ZUMO
Temperatura	Proximidad
Humedad	Acelerómetro
Presión Atmosférica	Magnetómetro
Brújula	Giróscopo
Temperatura de la CPU	Tensión de las Pilas

Tabla 1: Sensores Monitorizados.

Por tanto, se trata de un sistema de control remoto y monitorización utilizando protocolos de internet y presentación web.

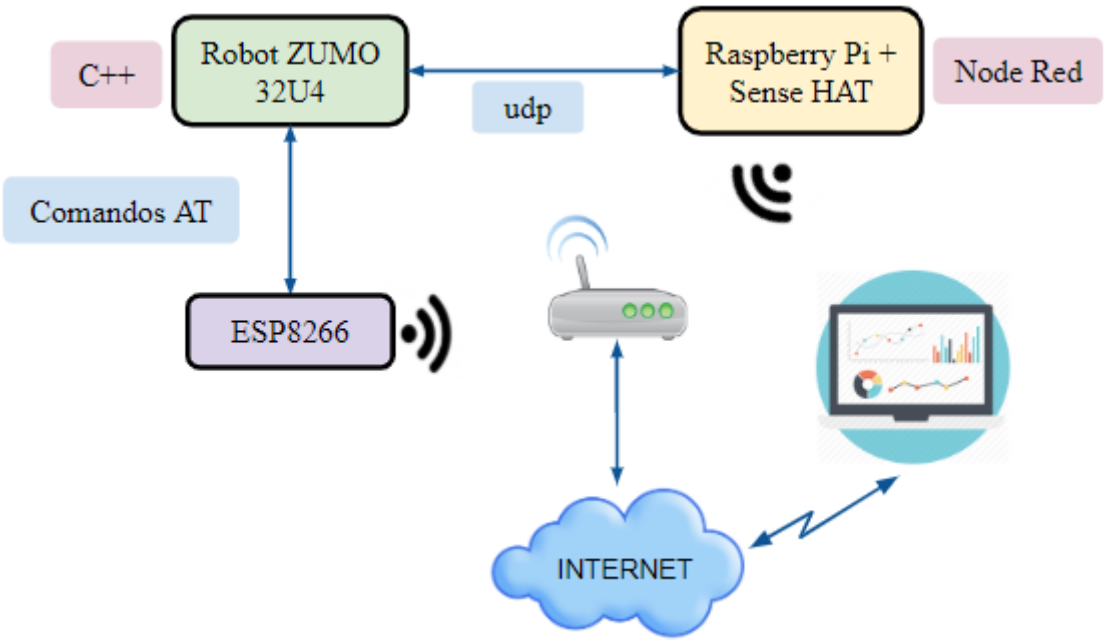


Ilustración 1: Esquema General del Sistema.

2. Lista de Materiales y herramientas utilizadas

- Raspberry Pi 3 b
- Sense Hat
- Robot ZUMO (Tarjeta compatible con Arduino)
- Módulo Wi-Fi ESP8266
- Entorno programación *Node Red*
- Entorno programación Arduino (c++)

3. Raspberry Pi + Sense HAT

El lenguaje de programación utilizado en Raspberry Pi es *node red*, un entorno de programación vía web con *java script* en el servidor. Está diseñado para internet de las cosas, y permite transmisión de mensajes basados en nodos *java script.*

Para este proyecto, se han instalado los paquetes de Sense HAT y *dashboard* en *node red*, con el fin de extraer información del Sense HAT y presentar el estado de los sensores en una interfaz de usuario.

El programa se divide en varios bloques:

Lectura del estado de los sensores del Sense HAT

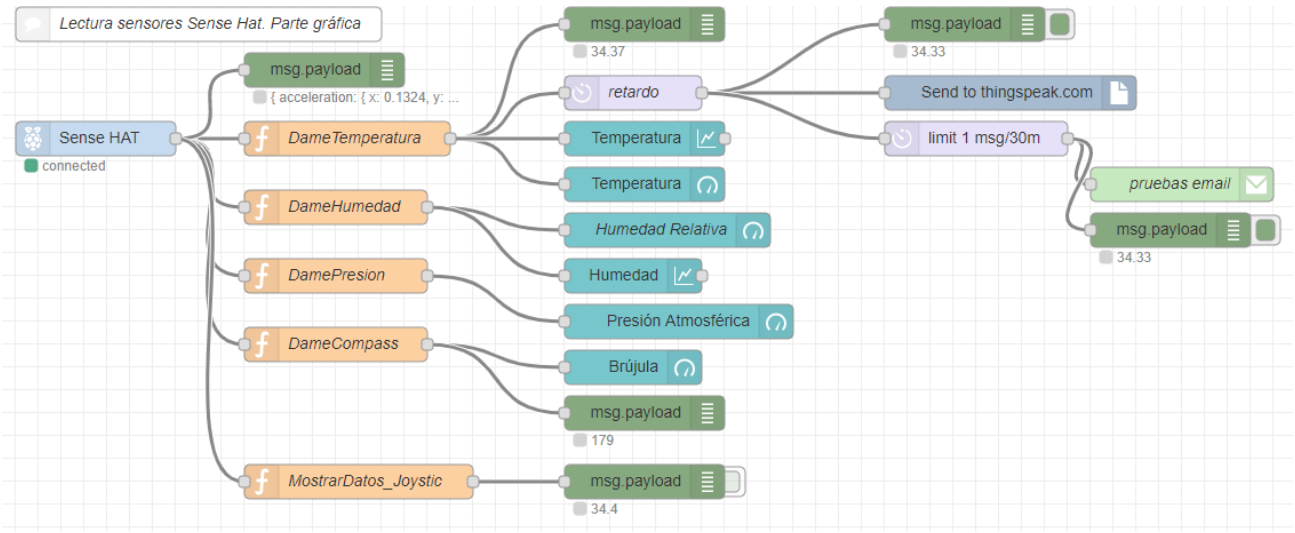


Ilustración 2: Node Red. Lectura sensores Sense HAT.

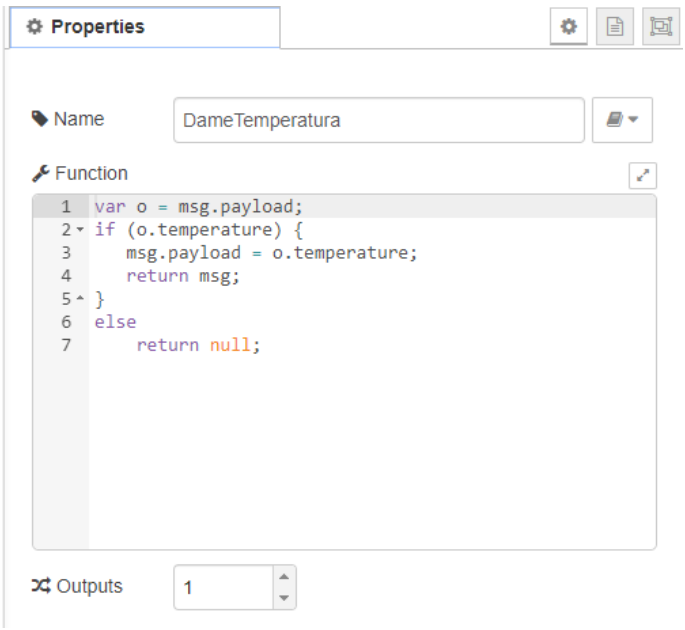


Ilustración 3. Node Red. función DameTemperatura para lectura de sensor de temperatura de Sense HAT. El valor de los demás sensores se obtiene de forma similar.

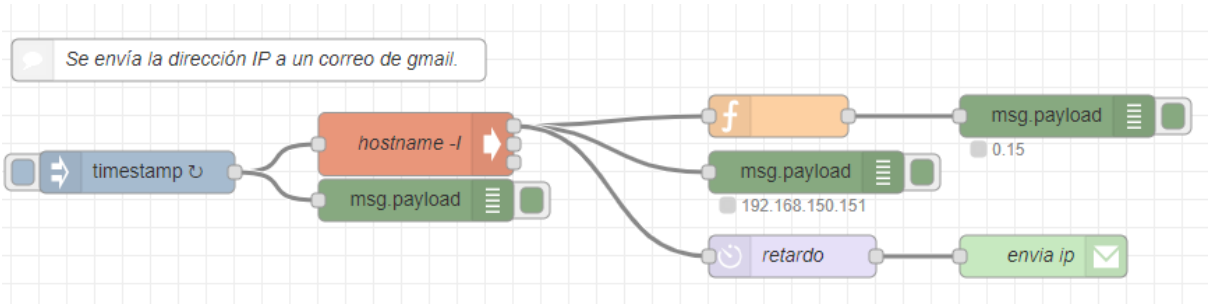


Ilustración 4: Node Red. Envío de la dirección IP asignada a una cuenta de correo electrónico.

El bloque *hostname* ejecuta un comando en el terminal de Raspberry, obteniendo la dirección IP asignada. A través de este bloque, también se puede leer la temperatura de la CPU.

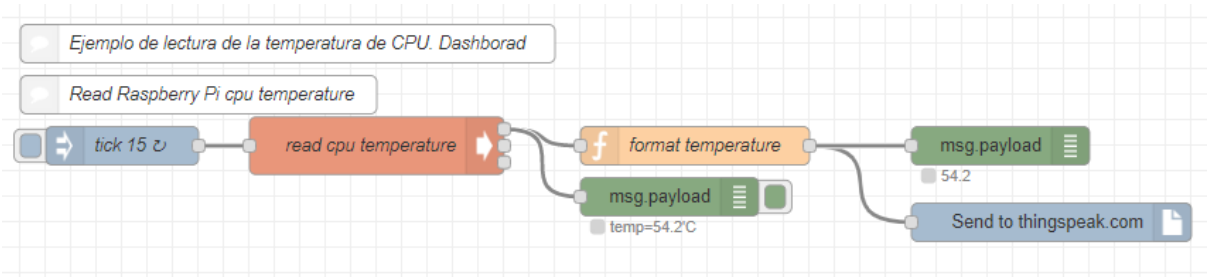


Ilustración 5: Node Red. Lectura y envío de la temperatura de la CPU de Raspberry.

El valor de la temperatura de la CPU, se envía a la plataforma *ThingSpeak*, junto con la temperatura captada por el sensor del Sense HAT.

Comunicación *udp* con el módulo ESP8266 conectado al Robot ZUMO

La comunicación se realiza mediante el protocolo *udp* (User Datagram Protocol), el cual permite enviar y recibir datagramas a través una red IP de forma directa.

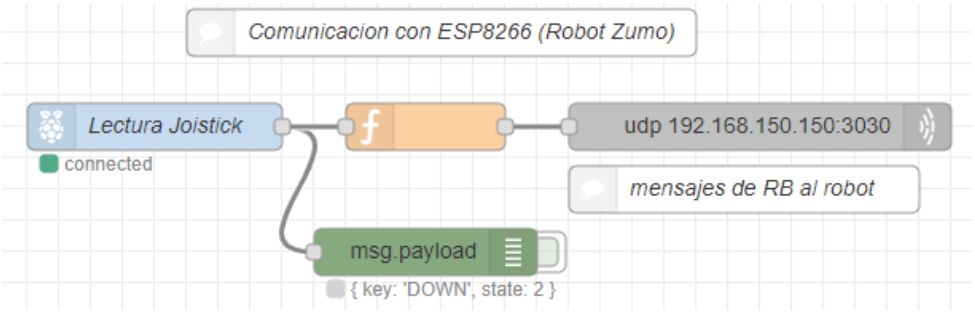


Ilustración 6: Node Red. Comunicación con ESP8266

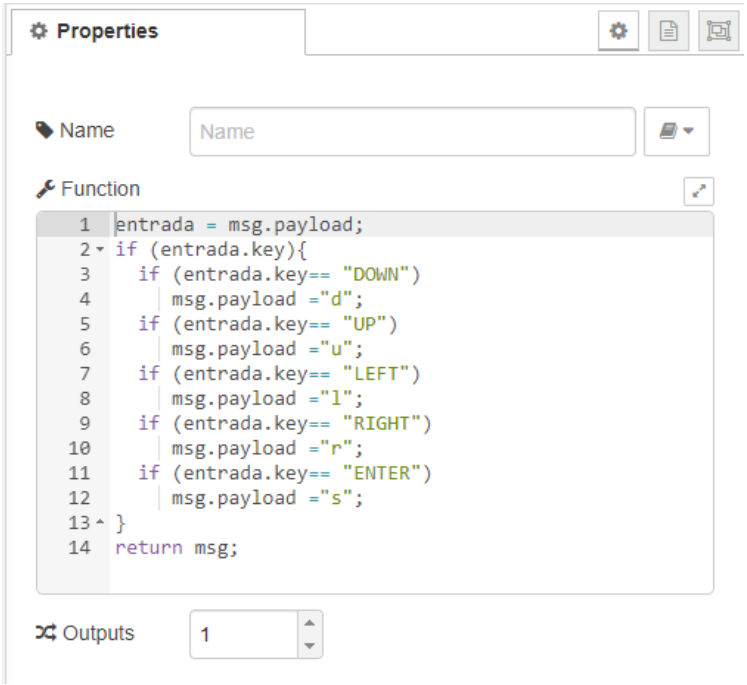


Ilustración 7: Node Red. Función que extrae las instrucciones del joystick del Sense HAT

Lectura del estado de los sensores del Robot ZUMO

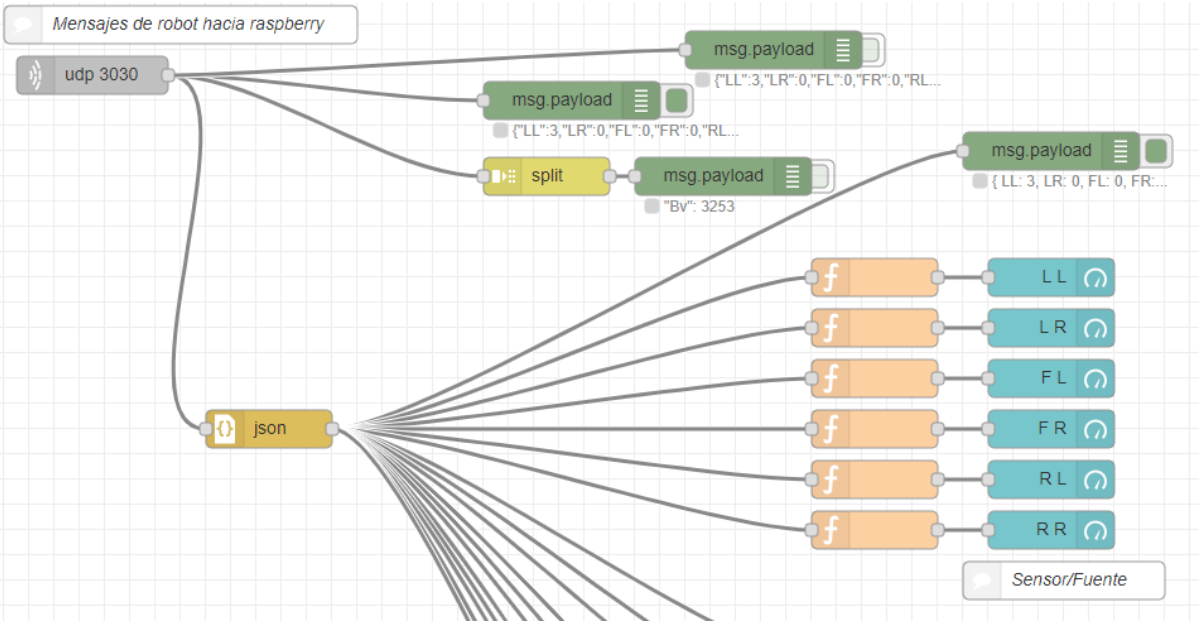


Ilustración 8: Node Red. Lectura de los sensores del Robot ZUMO. 1

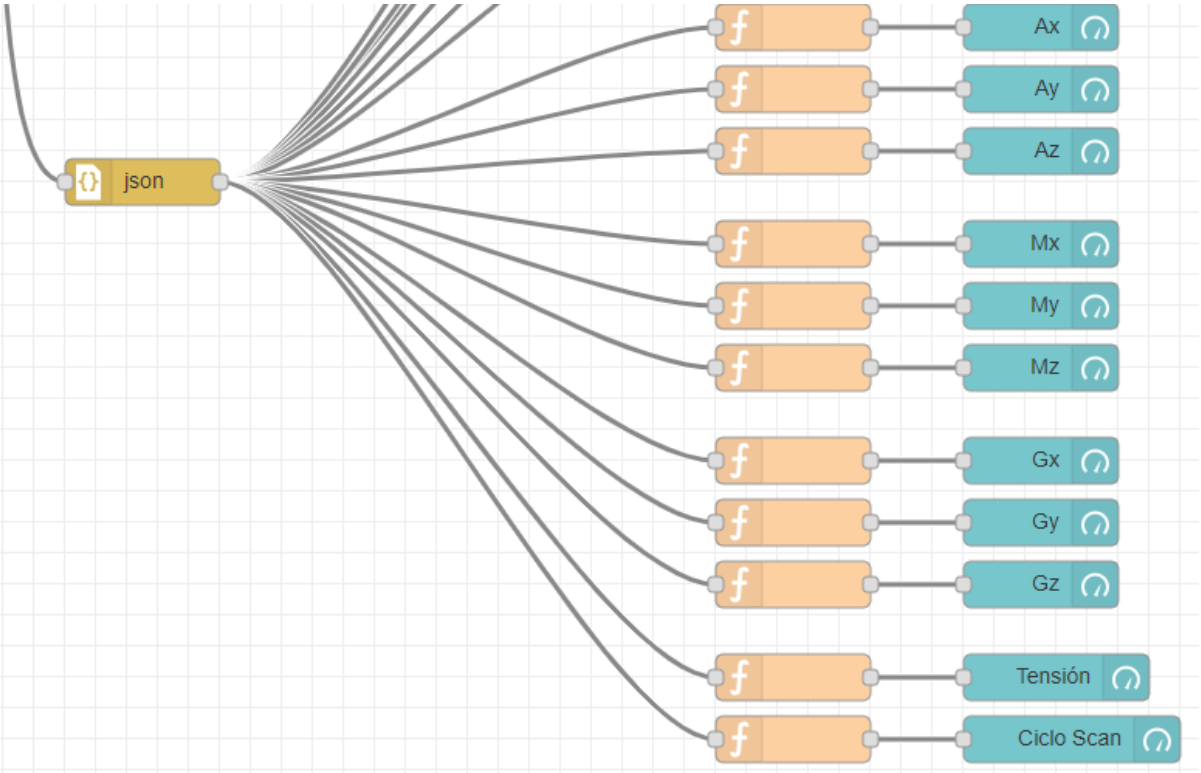


Ilustración 9:Node Red. Lectura de los sensores del Robot ZUMO. 2

El mensaje que Raspberry recibe del robot ZUMO está en formato *json*, por lo que es muy simple separar nombres y valores de variables.

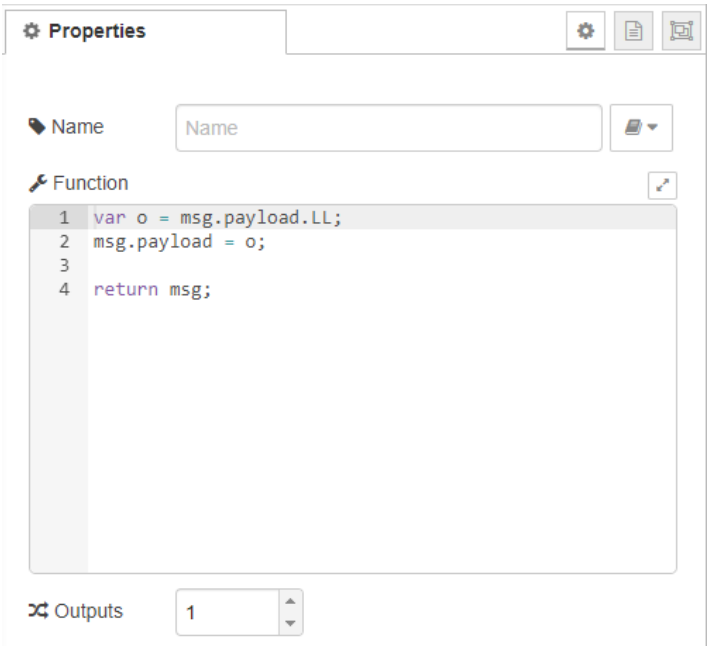


Ilustración 10. Node Red. Función para obtener el valor del sensor izquierdo de proximidad. El valor de los demás sensores de obtiene de forma similar.

El valor de los sensores de proximidad, se muestran en la matriz de leds del sense HAT, con el objetivo de realizar un control remoto más seguro.

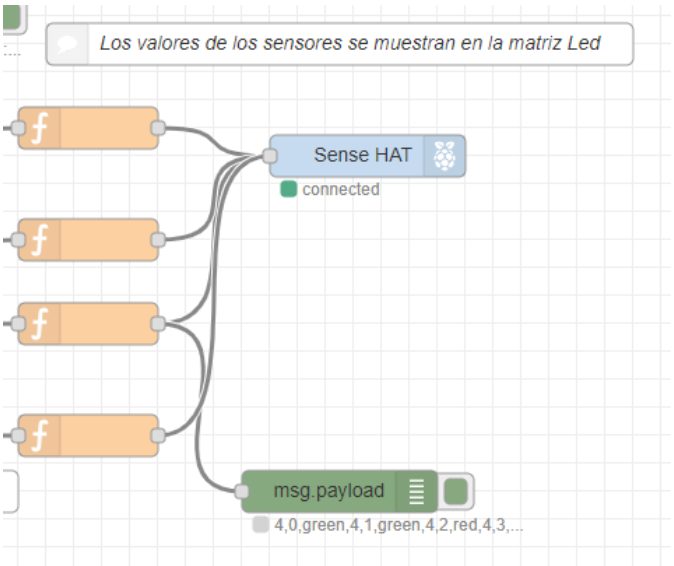


Ilustración 11: Node Red. Mostrar valor de los sensores de proximidad en la matriz de Leds.

Entorno de Monitorización del Sistema

El entorno web para la monitorización del sistema, se realiza mediante los bloques de *dashboard* para *Node Red*. Se presentan tres páginas, correspondientes a los datos de los sensores del sense HAT, del robot ZUMO y un enlace a la plataforma *ThinkSpeak*, dónde se visualiza la temperatura de la CPU de Raspberry y la temperatura captada por el sensor integrado en el sense HAT.

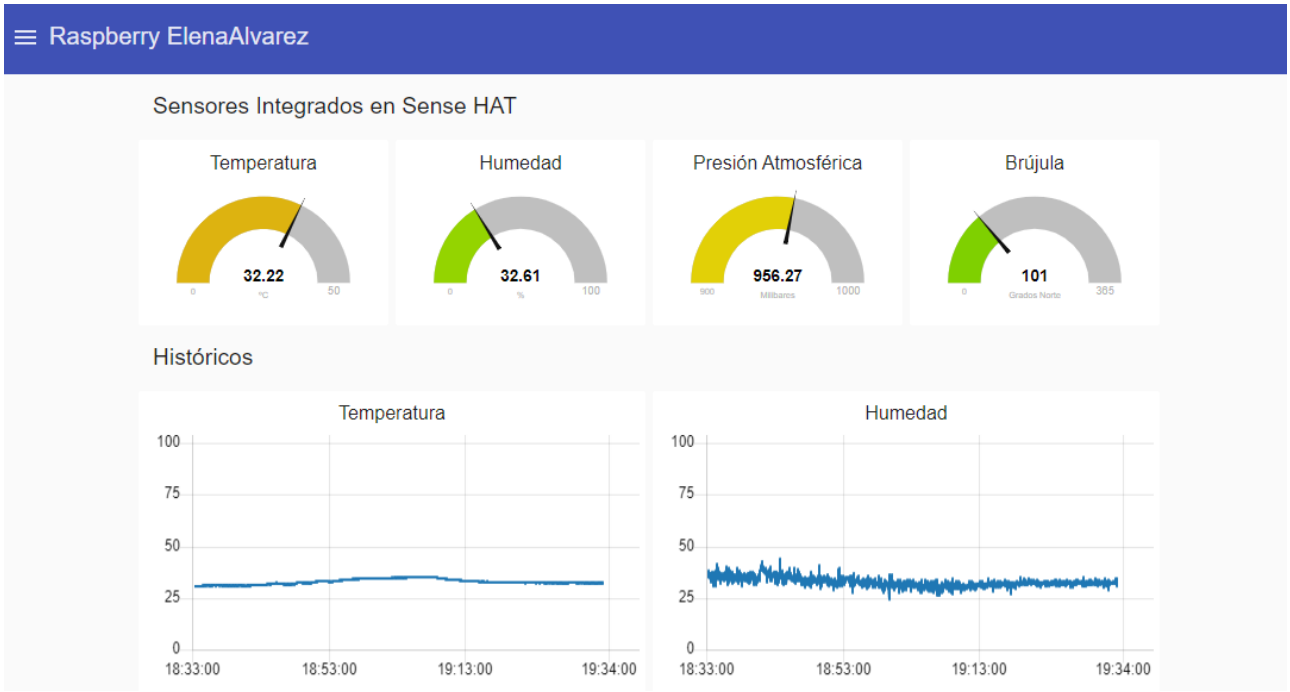


Ilustración 12: Entorno Web de monitorización. 1

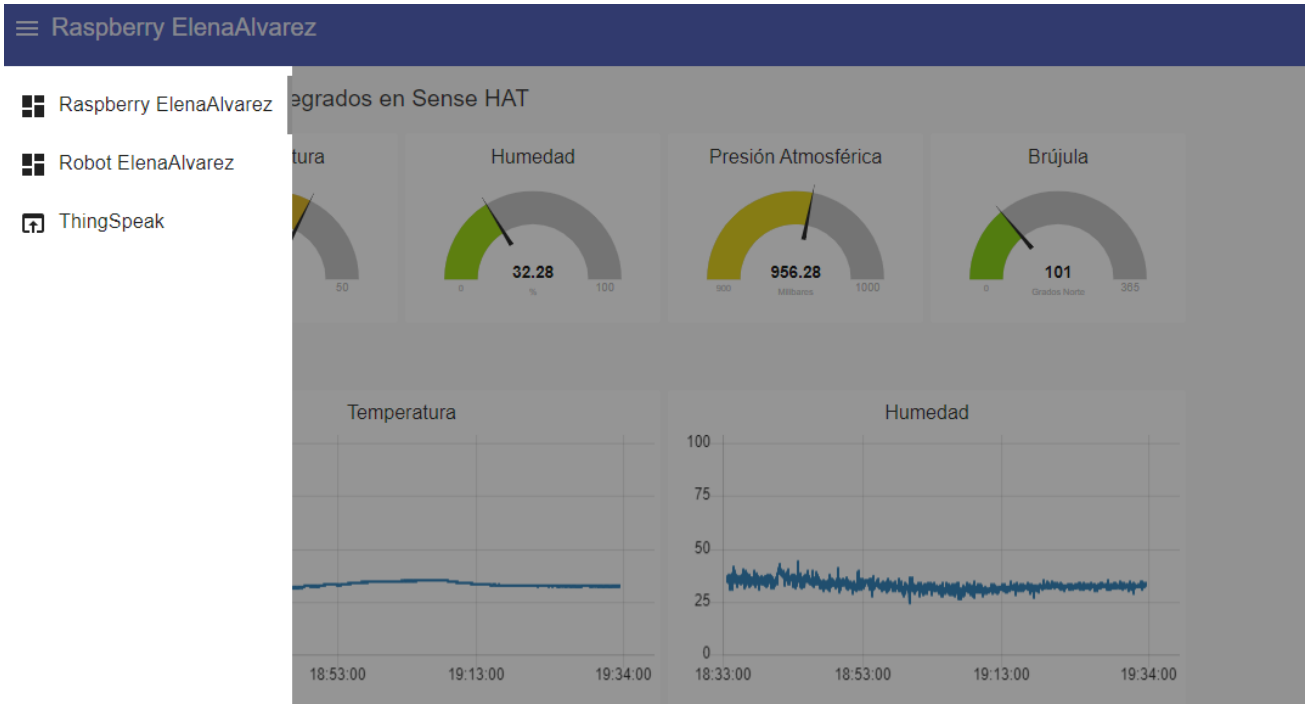


Ilustración 13: Entorno Web de Monitorización. 2

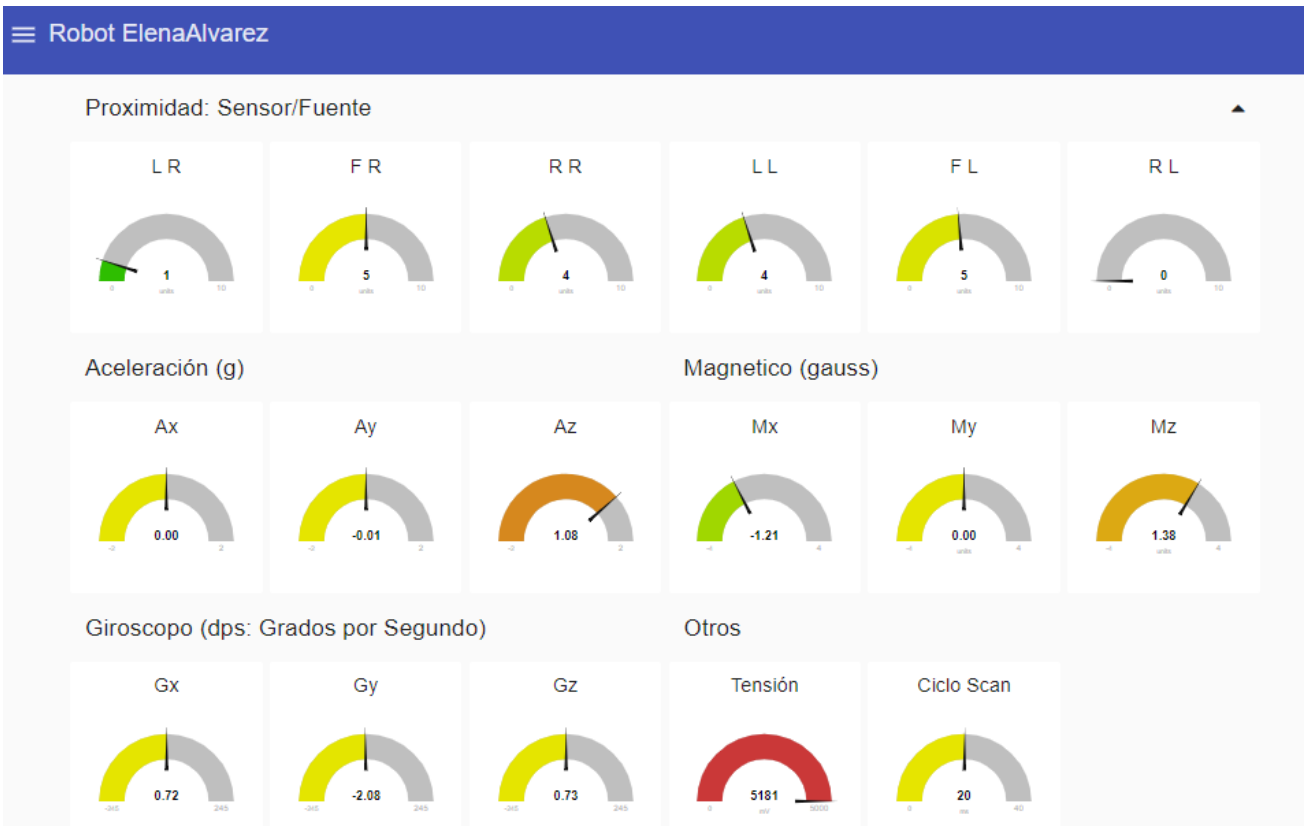


Ilustración 14: Entorno Web de monitorización. Sensores Robot ZUMO. 3

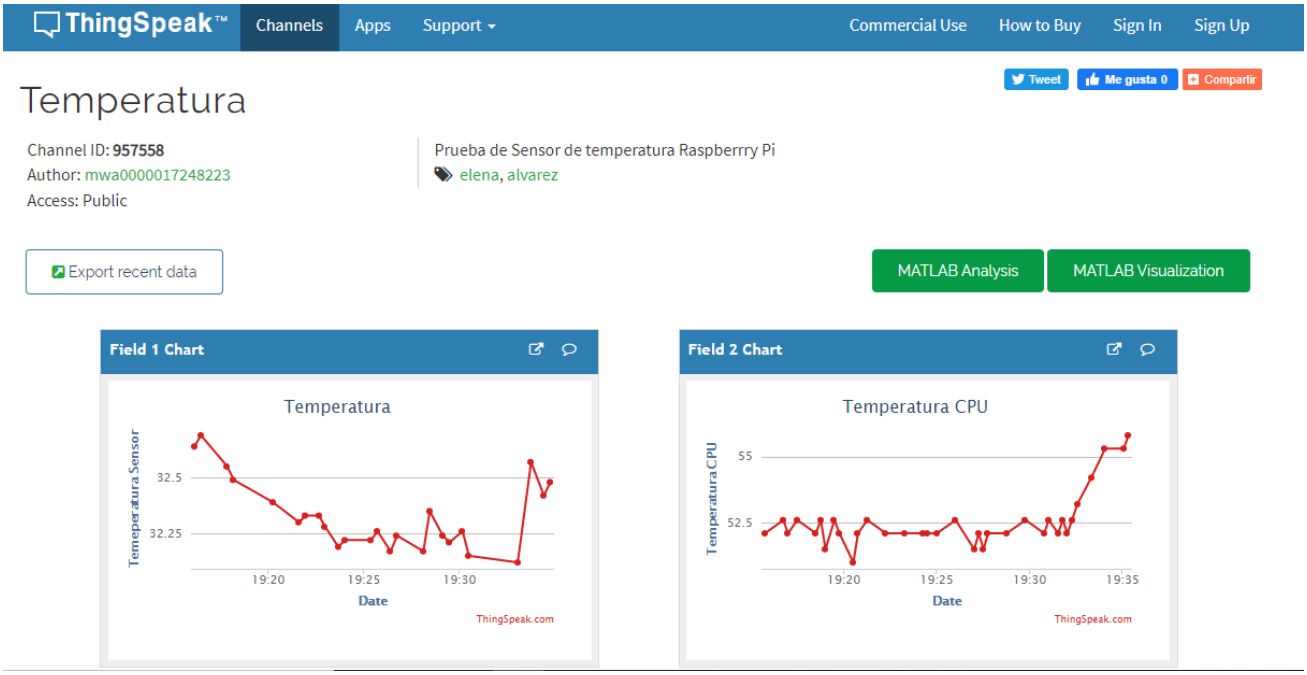


Ilustración 15: Entorno Web de monitorización. Plataforma ThinkSpeak. 4

Montaje Portátil del mando de control

Mediante un *power bank* para alimentar a Raspberry, se consigue un mando de control portátil del sistema.

En la matriz de leds, aparecen los valores de los sensores de proximidad frontales, de derecha y de izquierda. Sus valores se van actualizando conforme detecta obstáculos.

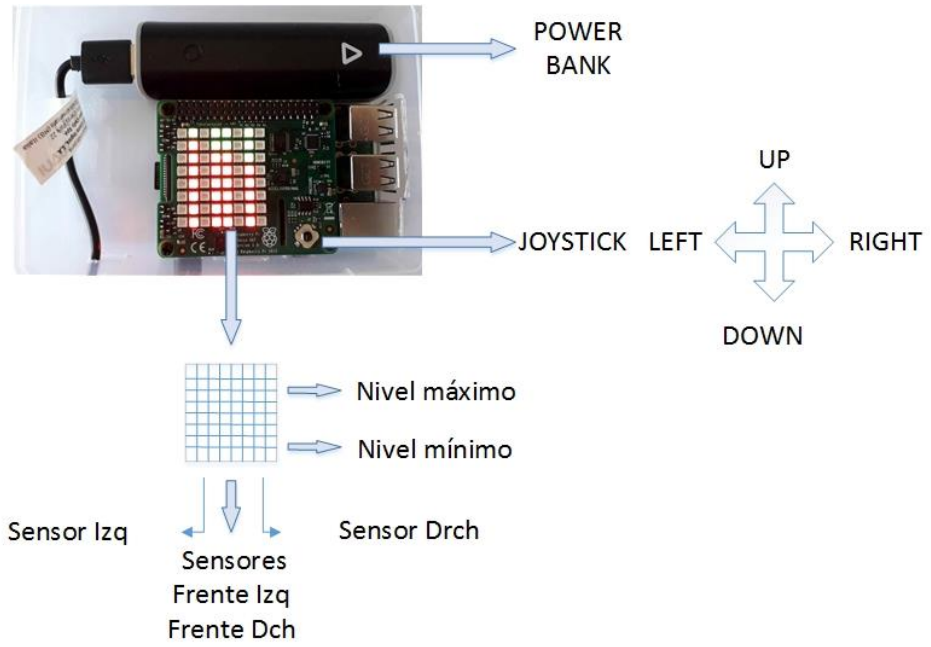


Ilustración 16: Mando para control remoto de robot ZUMO.

4. Robot ZUMO + Módulo wifi ESP8266

Lee los sensores del robot, actúa conforme órdenes recibidas del *joystick* del Sense HAT, controla la presencia de obstáculos para no chocar y envía los datos de monitorización en cada ciclo del programa.

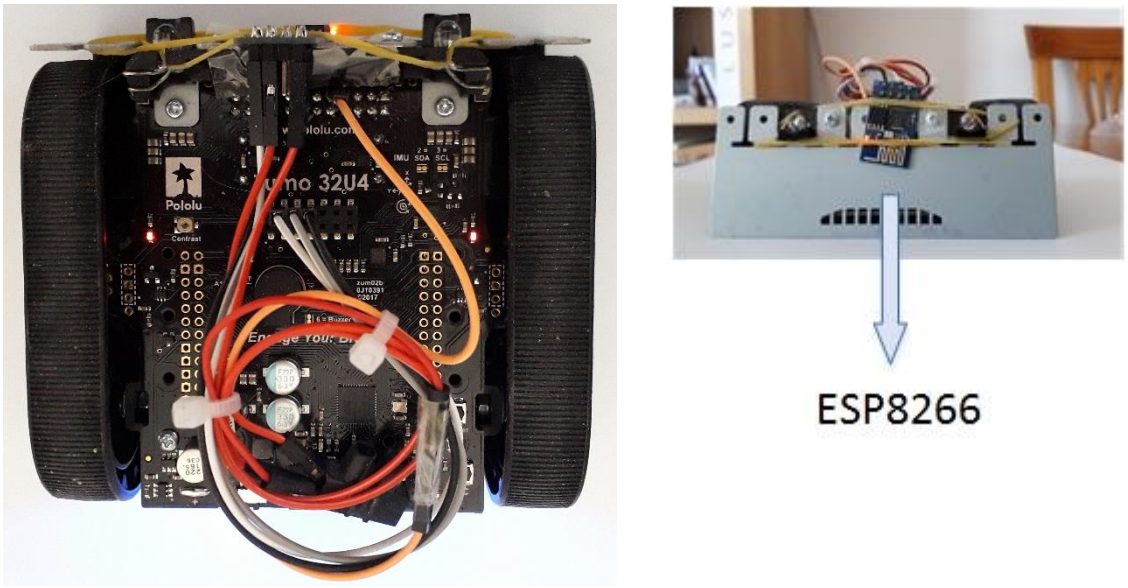


Ilustración 17: Robot ZUMO 32U4 con módulo wifi ESP8266

Diagrama Programa C++

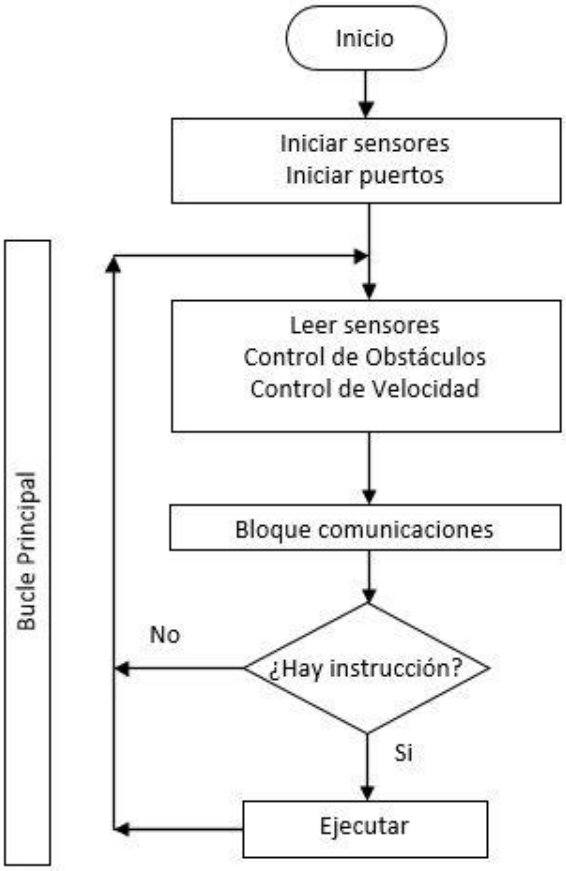
El programa principal para el Robot ZUMO, sigue el esquema mostrado en la *ilustración 18*. El programa completo, se adjunta en este documento en el Anexo 1.

En el bloque de comunicaciones, se realiza un control de estado para establecer la comunicación. Una vez establecida, los dos últimos estados son los encargados de enviar los datos que proceden de los sensores del robot a la Raspberry, en formato *json*. Si ya está establecida la conexión, el programa se salta los estados iniciales y realiza el envío de datos directamente. El esquema del funcionamiento del bloque de comunicaciones se muestra en *la ilustración 19*.

Sistemas de seguridad

Haciendo uso de los sensores integrados en el robot ZUMO, se implantan mecanismos de seguridad para el control remoto.

- Sensores de proximidad. Si el robot encuentra un obstáculo, se detiene de forma automática y deja de obedecer al sistema de control. En este caso sólo es posible enviar la instrucción retroceder. Una vez que los sensores de proximidad del robot vuelvan a los valores normales, el control total vuelve a ser posible.
- Activación de un *buzzer* cuando se llega a la velocidad máxima.



lustración 18: Esquema principal del programa utilizado

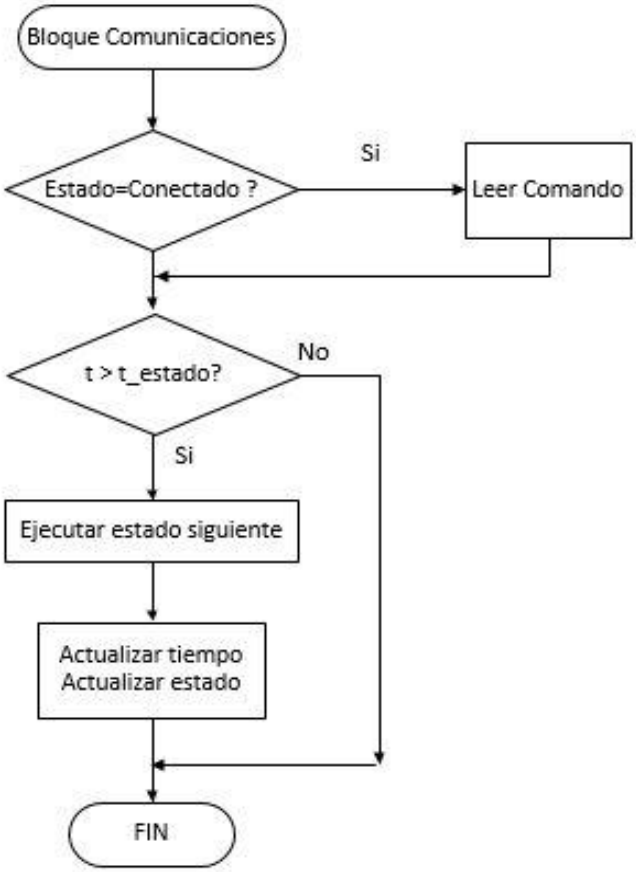


Ilustración 19: Control de estado Bloque de Comunicaciones

Anexo 1. Programa Arduino

```
#include <Arduino.h>
//#include "comandosRemotos.h"
#include <Zumo32U4Motors.h>
#include <Zumo32U4Buzzer.h>
#include<Zumo32U4ProximitySensors.h>
#include<Zumo32U4LineSensors.h>
// Para gyro y compas
#include <Wire.h>
#include <Zumo32U4.h>

LSM303 compass;
L3G gyro;
#define null (char) 0
//  Arduino pin 2 (RX) to ESP8266 TX
//  Arduino pin 3 to voltage divider then to ESP8266 RX
//  Connect GND from the Arduiono to GND on the ESP8266
//  Pull ESP8266 CH_PD HIGH

// When a command is entered in to the serial monitor on the computer
// the Arduino will relay it to the ESP8266

char ComunicaDatosESP8266();
charPresentaRespuestaModuloWifi();
voidInicializaSensoresLineaProximidad();
void InicializaGyroCompass();
voidActualizaGyroCompass();
voidActualizaTensionBateria();
int LEDPIN = 13;
Zumo32U4Motors motors;
Zumo32U4Buzzer buzzer;
Zumo32U4LineSensors      lineSensors;
Zumo32U4ProximitySensors proxSensors;

bool proxLeftActive;
bool proxFrontActive;
bool proxRightActive;

const uint8_t remoteUp    = 'u';
const uint8_t remoteLeft  = 'l';
const uint8_t remoteRight = 'r';
const uint8_t remoteDown  = 'd';
const uint8_t remoteStop  = 's';

int16_t leftSpeed = (int16_t) 0;
int16_t rightpeed = (int16_t) 0;

char reportGyroCompas[120];
//Caracteres enviados con el valor de los sensores
charcadenaMonitorizacion[512];
```

```
void setup()
{
   pinMode(LEDPIN, OUTPUT);

   Serial.begin(9600);     // communication with the host computer
   //while (!Serial);
   digitalWrite(LED_BUILTIN, LOW);

   // Start the software serial for communication with the ESP8266
   // ComunicaDatosESP8266(); se hace en el loop
   while (!Serial1);

  InicializaSensoresLineaProximidad();
   InicializaGyroCompass();

   //Para asegurarnos el fin de la cadena de Monitorizacion.
   for (int i =0; i<sizeof(cadenaMonitorizacion);i++)
      cadenaMonitorizacion[i] = NULL;

   /* De pruebas anteriores
   Serial.println("Abro puerto udp");
  Serial.println("AT+CIPSTART=\"UDP\",\"192.168.1.139\",3030,3030,0");

  Serial1.println("AT+CIPSTART=\"UDP\",\"192.168.1.139\",3030,3030,0");
   delay(1000);
   if (Serial1.available())
      Serial.write(Serial1.read());
   */
}

void loop()
{
   char comando;

   for (int i =0; i<sizeof(cadenaMonitorizacion);i++)
      cadenaMonitorizacion[i] = NULL;
   ActualizaSensoresProximidad();
   ActualizaGyroCompass();
   ActualizaTensionBateria();
   comando = ComunicaDatosESP8266();
   if (comando){
     Serial.print("Ejecutar Comando:");
     Serial.println(comando);
     processRemoteCommand(comando);
    }
   // listen for user input and send it to the ESP8266
   if (Serial.available())
      Serial1.write(Serial.read());
}
```

```
/*
 * Del programa de Irene
 */
charComunicaDatosESP8266(){

  static unsigned long ultimoTiempo   = 0;
  static unsigned long waitProximoComando = 1000; //mil milisegundos entre
comandos
  static int           estadoConexion = 0;
  unsigned long        actualTiempo;
  int                  longitudDatos;
  char                 contenedor[20];
  char                 contenedor1[20];
  char                 respuesta = 0;
  int                  longitudDatosEnviar;
  static char          comandoLongitud[16];

  if (estadoConexion > 0)
    respuesta = PresentaRespuestaModuloWifi();

  actualTiempo = millis();
  if (actualTiempo < (ultimoTiempo + waitProximoComando)){
    //Serial.println("Retorna");
    return (respuesta); //Ojo !!! si no ha pasado el tiempo necesiario, no
hacemos nada
  }
  else
    ultimoTiempo = actualTiempo; // Y enviamos comandos ...

  switch (estadoConexion){
    case 0:
      Serial1.begin(115200);
      waitProximoComando = 1000;
      estadoConexion ++;
      break;
    case 1:
      Serial1.println("AT+CWMODE=3"); //Tiene tres modos (1,2,3), elegimos este.
      waitProximoComando = 1000;
      estadoConexion ++;
      break;
    case 2:
      Serial1.println("AT+RST");
      waitProximoComando = 1000;
      estadoConexion ++;
      break;
    case 3:
      Serial1.println
      // ("AT+CWJAP=\"Andared\",
```

```
\"llevalatararaunvestidoblancollenodecascabeles\"");
      ("AT+CWJAP=\"Pololu\",\"Zumo32U4\"");
      waitProximoComando = 8000;
      estadoConexion ++;
      break;
    case 4:
      Serial1.println("AT+CIFSR"); //Nos da la IP local
      waitProximoComando = 1000;
      estadoConexion ++;
      break;
    case 5: //conecta con la raspberry
     Serial1.println("AT+CIPSTART=\"UDP\",\"192.168.150.151\",3030,3030,0");
      waitProximoComando = 500;
      estadoConexion ++;
      digitalWrite(LED_BUILTIN, HIGH);
      buzzer.playFrequency(440, 500, 10);
      break;
    case 6: //envia la longitud de datos
      //longitudDatos= getStr.length();
      //comandoLongitud += String(longitudDatos);
      //Serial.println(comandoLongitud);
     longitudDatosEnviar=strlen(cadenaMonitorizacion);
     sprintf(comandoLongitud, "AT+CIPSEND=%d",longitudDatosEnviar);
      //Serial1.println("AT+CIPSEND=4");
      Serial1.println(comandoLongitud);
      waitProximoComando = 100;
      estadoConexion ++;
      break;
    case 7: //envia los datos
      //Serial.println(getStr);
      Serial.println(cadenaMonitorizacion);
      Serial1.println(cadenaMonitorizacion);
      digitalWrite(LED_BUILTIN, !digitalRead(LED_BUILTIN));
      //delay(1000);
      //estadoConexion = 5; //volvemos a conectar dentro de ...
      waitProximoComando = 100;
      estadoConexion = 6;
      break;
  }
  return (respuesta);

}

charPresentaRespuestaModuloWifi(){
  //presenta en el puerto serie la respuesta del modulo wifi
  static char lecturaComando[8]=
        {null, null, null, null, null, null, null, null};
  static int indiceComando = 0;
  while (Serial1.available()){
```

```
byte b = Serial1.read();
Serial.write(b);
// los datos de lectura se leen en la forma +IPD,1:d
// +IPD,numero:dato
// donde +IPD, Es el encabezado que introduce el ESP8266
// numero es el numero de caracteres recibidos
// dato es el dato en si.

// Aqui tenemos un control de los "comandos recibidos", en principio
// de un solo caracter.
switch (indiceComando){
 case 0:
    if (b=='+'){
    lecturaComando[indiceComando]=b;
     indiceComando++;
    }
    break;
 case 1:
    if (b=='I'){
    lecturaComando[indiceComando]=b;
     indiceComando++;
    }
    else
     indiceComando = 0;
    break;
 case 2:
    if (b=='P'){
    lecturaComando[indiceComando]=b;
     indiceComando++;
    }
    else
     indiceComando = 0;
    break;
 case 3:
    if (b=='D'){
    lecturaComando[indiceComando]=b;
     indiceComando++;
    }
    else
     indiceComando = 0;
    break;
 case 4:
    if (b==','){
    lecturaComando[indiceComando]=b;
     indiceComando++;
    }
    else
     indiceComando = 0;
    break;
```

```
     case 5:
        if (b=='1'){
         lecturaComando[indiceComando]=b;
          indiceComando++;
         }
        else
          indiceComando = 0;
        break;
     case 6:
        if (b==':'){
         lecturaComando[indiceComando]=b;
          indiceComando++;
         }
        else
          indiceComando = 0;
        break;
     case 7: //Aqui esta el comando
        lecturaComando[indiceComando]=b;
        indiceComando++;
        break;
    }//switch
    if (indiceComando == 8){
     //acabamos de leer un comando
     indiceComando = 0;
     return lecturaComando[7];
    }
  }//Serial available
  return 0;
}

// Start running the new command.
void processRemoteCommand(uint8_t command){

  #define INCREMENTO_VELOCIDAD 10

  switch(command)
  {
  case remoteUp:
    leftSpeed += INCREMENTO_VELOCIDAD;
    rightpeed += INCREMENTO_VELOCIDAD;
   motors.setSpeeds(leftSpeed, rightpeed);
    break;
  case remoteDown:
    leftSpeed -= INCREMENTO_VELOCIDAD;
    rightpeed -= INCREMENTO_VELOCIDAD;
    break;
  case remoteLeft:
    leftSpeed -=  INCREMENTO_VELOCIDAD;
    rightpeed +=  INCREMENTO_VELOCIDAD;
```

```
    break;
  case remoteRight:
    leftSpeed +=  INCREMENTO_VELOCIDAD;
    rightpeed -=  INCREMENTO_VELOCIDAD;
    break;
  case remoteStop:
    leftSpeed = 0;
    rightpeed = 0;
    break;
/*
  case remote1:
    buzzer.playNote(NOTE_C(4), 200, 15);
    break;

  case remote2:
    buzzer.playNote(NOTE_D(4), 200, 15);
    break;

  case remote3:
    buzzer.playNote(NOTE_E(4), 200, 15);
    break;

  case remote4:
    break;

  case remote5:
    break;

  case remote6:
    break;

  case remote7:
    break;

  case remote8:
    break;

  case remote9:
    break;
*/
  }
  //Control de limites de velocidad
  if (leftSpeed > 0) if (leftSpeed > 400) leftSpeed = 400;
  if (rightpeed > 0) if (rightpeed > 400) rightpeed = 400;
  if (leftSpeed < 0) if (leftSpeed <-400) leftSpeed = -400;
  if (rightpeed < 0) if (rightpeed <-400) rightpeed = -400;


 motors.setSpeeds(leftSpeed, rightpeed);
```

```
  if (leftSpeed > 300 | rightpeed > 300
     |leftSpeed < -300 | rightpeed < -300)
      buzzer.playFrequency(440, 500, 10);
}

void InicializaSensoresLineaProximidad () {
/**
The assembled versions of the Zumo 32U4 robot ship with jumpers selecting the
left (LFT) and right (RGT) proximity sensors instead of down-facing DN2 and DN4,
so these versions are configured for three down-facing sensors and all three
proximity sensors by default.
Veasehttps://www.pololu.com/docs/0J63/3.5
**/
 /* Configuration 1:
   * - 3 line sensors (1, 3, 5)
   * - 3 proximity sensors (left, front, right)
   *
   * For this configuration to work, jumpers on the front sensor
   * array must be installed in order to connect pin 4 to RGT and
   * connect pin 20 to LFT.  This is a good configuration for a
   * sumo robot. */
 //lineSensors.initThreeSensors();
 proxSensors.initThreeSensors();
}
voidActualizaSensoresProximidad(){
  static uint16_t lastSampleTime = 0;

  if ((uint16_t)(millis() - lastSampleTime) >= 0){
    lastSampleTime = millis();
    // Send IR pulses and read the proximity sensors.
    proxSensors.read();

  //Control de Proximidad
  //Si detectamos objeto demasiado cerca. paramos
  if((proxSensors.countsFrontWithLeftLeds()  > 5)&(leftSpeed > 0)){
         leftSpeed = 0;
        motors.setSpeeds(leftSpeed, rightpeed);
  }
  if((proxSensors.countsFrontWithRightLeds() > 5)&(rightpeed > 0)){
         rightpeed = 0;
        motors.setSpeeds(leftSpeed, rightpeed);
  }

// Just read the proximity sensors without sending pulses.
    proxLeftActive  = proxSensors.readBasicLeft();
    proxFrontActive = proxSensors.readBasicFront();
    proxRightActive = proxSensors.readBasicRight();
     // Send the results to the
    PrintReadingsToNetwork();
```

```
  }
}
voidPrintReadingsToNetwork(){
  //static char sensoresProximidad[160];

  uint16_t lineSensorValues[5] = { 0, 0, 0, 0, 0 };
 sprintf(cadenaMonitorizacion,
  //formato JSON

"{\"LL\":%d,\"LR\":%d,\"FL\":%d,\"FR\":%d,\"RL\":%d,\"RR\":%d,\"LA\":%d,\"FA\":%d,
\"RA\":%d,\"LS0\":%4d,\"LS1\":%4d,\"LS2\":%4d,\"LS3\":%4d,\"LS4\":%4d,",
    //Sensor/Emisor
   proxSensors.countsLeftWithLeftLeds(),
   proxSensors.countsLeftWithRightLeds(),
   proxSensors.countsFrontWithLeftLeds(),
   proxSensors.countsFrontWithRightLeds(),
   proxSensors.countsRightWithLeftLeds(),
   proxSensors.countsRightWithRightLeds(),
    proxLeftActive,
    proxFrontActive,
    proxRightActive,
    lineSensorValues[0],
    lineSensorValues[1],
    lineSensorValues[2],
    lineSensorValues[3],
    lineSensorValues[4]
  );
  return;
}

/*
 * Del ejemplo inertialsensors
 */
voidInicializaGyroCompass()
{
  Wire.begin();

  if (!compass.init())
  {
    // Failed to detect the compass.
    ledRed(1);
    while(1)
    {
     Serial.println(F("Failed to detect the compass."));
      delay(100);
    }
  }

 compass.enableDefault();
```

```
  if (!gyro.init())
  {
    // Failed to detect the gyro.
    ledRed(1);
    while(1)
    {
     Serial.println(F("Failed to detect gyro."));
      delay(100);
    }
  }
 gyro.enableDefault();
}
voidActualizaGyroCompass()
{
  static int  primeraVez=1;
  int longitudPrevia;
  long minimo;
  long maximo;



  compass.read();
  gyro.read();

//"{\"LL\":%d,\"LR\":%d,\"FL\":%d,\"FR\":%d,\"RL\":%d,\"RR\":%d,\"LA\":%d,\"FA\":%
d,\"RA\":%d,\"LS0\":%4d,\"LS1\":%4d,\"LS2\":%4d,\"LS3\":%4d,\"LS4\":%4d}\n",

// snprintf_P(reportGyroCompas, sizeof(reportGyroCompas),
//
PSTR("{\"Ax\":%6d,\"Ay\":%6d,\"Az\":%6d,\"Mx\":%6d,\"My\":%6d,\"Mz\":%6d,\"Gx\":%6
d,\"Gy\":%6d,\"Gz\"%6d"}\n"),

/*
Serial.print("Longitud Cadena Monitorizacion1=");
longitudPrevia=strlen(cadenaMonitorizacion);
Serial.println(longitudPrevia);
*/
sprintf(cadenaMonitorizacion+strlen(cadenaMonitorizacion),
"\"Ax\":%6d,\"Ay\":%6d,\"Az\":%6d,\"Mx\":%6d,\"My\":%6d,\"Mz\":%6d,\"Gx\":%6d,\"Gy
\":%6d,\"Gz\":%6d,",
    compass.a.x, compass.a.y, compass.a.z,
    compass.m.x, compass.m.y, compass.m.z,
    gyro.g.x, gyro.g.y, gyro.g.z);

/*
Serial.print("Longitud Cadena Monitorizacion2=");
Serial.println(strlen(cadenaMonitorizacion));
```

```
minimo=compass.m_min.x;
maximo=compass.m_max.x;
Serial.println("m_max=%6d;m_min=%6d",compass.m_min.x,compass.m_min.x);
*/
  delay(1);
}
voidActualizaTensionBateria(){
  uint16_t   tensionBateria;

   tensionBateria = readBatteryMillivolts ();
 sprintf(cadenaMonitorizacion+strlen(cadenaMonitorizacion),
          "\"Bv\":%6d}\n",
          tensionBateria);

}
```