Guía Software y Uso

ELCOche

María del Pilar Aguilera Manzanera

Enrique Calatayud Candelas

Guillermo González Martín

María Tejedor Rami

Índice de contenidos

1. Introducción

Guia del proyecto educativo ELCOche, diseñado para la asignatura Electrónica de consumo del Grado En Ingenieria De Tecnologias Y Servicios De Telecomunicación de la universidad Politécnica de Madrid. En la que se explica cómo realizar la programación del proyecto una vez completado el manual de montaje.

2. Arduinos

Para la realización de ELCOche se hace uso de dos Arduinos. Uno se encontrará en el coche y será el receptor. Se usa un Arduino Nano por su pequeño tamaño, ya que no se dispone de mucho espacio. Por otro lado, en la parte de control, se encuentra el transmisor. Se usa un Arduino Mega, ya que este dispone un mayor número de entradas y salidas digitales, a diferencia de otros modelos.
La definición y programación de los siguientes módulos no es una transcripción literal del código, hay cosas obviadas para dar sencillez a la lectura.

3. Programación del Transmisor

El transmisor se encuentra en la zona de control (volante, pedal y gafas) y se programa en un Arduino Mega.
Se divide el código según los diferentes bloques que lo componen:

a. Zona de configuración

Al principio del código se definen todas las constantes y los pines del Arduino a los que se van a conectar los módulos que se van a utilizar, para así poder ser modificadas con mayor facilidad en el caso de ser necesario.

b. Envío de datos por radio

Para hacer uso de las radios, se necesita importar la librería SoftwareSerial.h. Esta librería ha sido desarrollada para permitir comunicación serie por cualquier par de pines digitales. El Arduino tiene incorporados en los pines 0 y 1 este tipo de comunicación, pero se opta por usar la librería ya que si se usaran estos pines habría problemas al depurar (los mensajes mostrados en la consola del ordenador utilizan dichos pines)

Para crear una comunicación serie en dos pines se utiliza la siguiente sentencia:

```
SoftwareSerial radio(PIN_RADIO_TX, PIN_RADIO_RX);
```

Las radios elegidas funcionan a **57600** baudios. Esto se indica en el `setup()` de la siguiente forma:

```
radio.begin(RADIO_BAUDRATE);
```

Para el envío, se crea un *string* '*data*' con todos los datos de control separados mediante comas. Para la decodificación de estos datos en el receptor, se envían caracteres de control ('s' - start y 'e' - end).

```
radio.print("s" + data + "e");
```

c. Lectura de los potenciómetros

En el transmisor se realiza la lectura de dos potenciómetros. Uno controla la dirección del coche y se encuentra en el volante, y el otro controla la velocidad del coche y se encuentra en el pedal.

Se definen los pines en la zona de configuración y se declaran como entrada en el `setup()`:

```
pinMode(PIN_MOTOR, INPUT);
pinMode(PIN_DIR,   INPUT);
```

Se realiza una lectura analógica y se guarda el valor en una variable, haciendo previamente un mapeado para transformar los valores de lectura del potenciómetro (0 a 1023[1]) en los adecuados para el motor (0 a 255) y dirección (60 a 120).

Para calcular la dirección es necesario realizar un calibrado previamente, en el que se mira cual es el valor del potenciómetro cuando el volante está en su posición de reposo (CEN_WHEEL), el valor que tiene cuando está completamente girado hacia la izquierda (MIN_WHEEL) y hacia la derecha (MAX_WHEEL).

[1] El Arduino Mega posee conversores analógico-digitales de 10 bits (2^{10} = 1024) en sus 16 (A0 - A15) entradas analógicas

Se obtiene el mínimo de estos dos últimos valores, y se le suma y resta a la posición inicial, pasandoselos al map se pretende que se obtenga la dirección 90 cuando el volante esté en reposo, y que alcance 60 o 120 cuando esté completamente girado a cada uno de los sentidos.

También es necesario saber cual es valor máximo del potenciómetro al pisar completamente el pedal (MAX_PEDAL) y el mínimo cuando está en reposo (MIN_PEDAL).

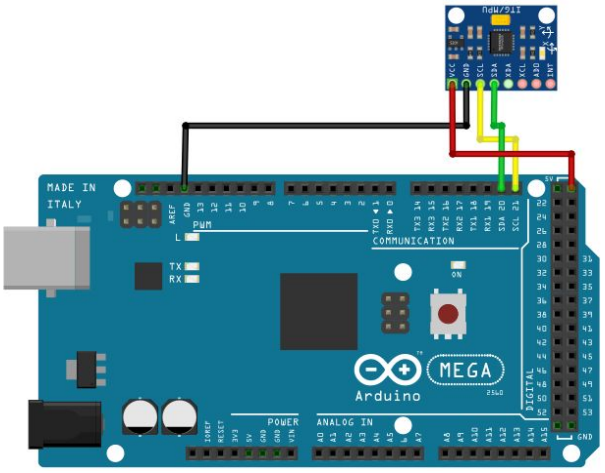
```
int max_turn = min(abs(MIN_WHEEL - CEN_WHEEL), abs(MAX_WHEEL -
CEN_WHEEL));

motor = map(analogRead(PIN_MOTOR), MIN_PEDAL, MAX_PEDAL, MIN_MOTOR,
MAX_MOTOR);

dir = map(analogRead(PIN_DIR), CEN_WHEEL - max_turn, CEN_WHEEL +
max_turn, MIN_DIR, MAX_DIR);
```

Además es recomendable asegurarse de que en caso de error del calibrado no se le pasa al motor o al servo un valor que no tenga sentido.

```
if (motor < 5) {
  motor = 0;
}

if (dir > MAX_DIR) {
  dir = MAX_DIR;
} else if (dir < MIN_DIR) {
  dir = MIN_DIR;
}
```

d. Lectura de datos del acelerómetro (MPU-6050)

En el transmisor se encuentran las gafas para controlar el gimbal en el coche. Esto se hace mediante un acelerómetro, el MPU-6050. Se conecta de la siguiente manera:



Para usarlo será necesario incluir la librería `<MPU6050.h>`.
Se define el acelerómetro:

```
MPU6050 mpu;
```

Se inicializa en el `setup()`:

```
while (!mpu.begin(MPU6050_SCALE_2000DPS, MPU6050_RANGE_2G)) {
  Serial.println("Could not find valid MPU6050 sensor, check wiring!");
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print(text_0);
  lcd.setCursor(0, 1);
  lcd.print("MPU ERROR!");
  lcd.setCursor(0, 2);
  lcd.print("Please, check wiring");
  delay(500);
}
```

En cada ciclo del `loop()` se leen los datos de interés del acelerómetro, en nuestro caso, el *yaw* (giro horizontal de la cabeza) y el *pitch* (giro vertical):

```
Vector norm = mpu.readNormalizeGyro();
yaw = yaw + norm.ZAxis * timeStep;
pitch = pitch + norm.YAxis * timeStep;
```

e. Botones y switches

El volante dispone de un panel de control con diferentes botones y *switches*. En la zona de configuración se define a qué pin se corresponde cada uno de ellos y en el `setup()` su modo de uso. Se definen como `INPUT_PULLUP` ya que cuando los botones y switches no están accionados no se pueden quedar al aire.

```
pinMode(PIN_LEVER_D,          INPUT_PULLUP);
pinMode(PIN_LEVER_R,          INPUT_PULLUP);
pinMode(PIN_SW_LIGHTS_ON,     INPUT_PULLUP);
pinMode(PIN_SW_LIGHTS_AUTO,   INPUT_PULLUP);
pinMode(PIN_SW_WARNINGS,      INPUT_PULLUP);
pinMode(PIN_SW_ULTRASONIC,    INPUT_PULLUP);
pinMode(PIN_BTN_STOP,         INPUT_PULLUP);
pinMode(PIN_BTN_CAL,          INPUT_PULLUP);
pinMode(PIN_BTN_CAL_MPU,      INPUT_PULLUP);
pinMode(PIN_BTN_CLAXON,       INPUT_PULLUP);
pinMode(PIN_SW_CAM,           INPUT_PULLUP);
```

Se realiza la lectura de cada uno de los pines con la función `readValue()` y guardamos su valor, se utiliza esta función para en vez de enviar 0 o 1, que es el valor que se obtiene mediante digitalRead se cambie a los valores definidos en DATA_LOW (2) y DATA_HIGH (3), para así en receptor detectar con mayor facilidad que ha habido un fallo al enviarse:

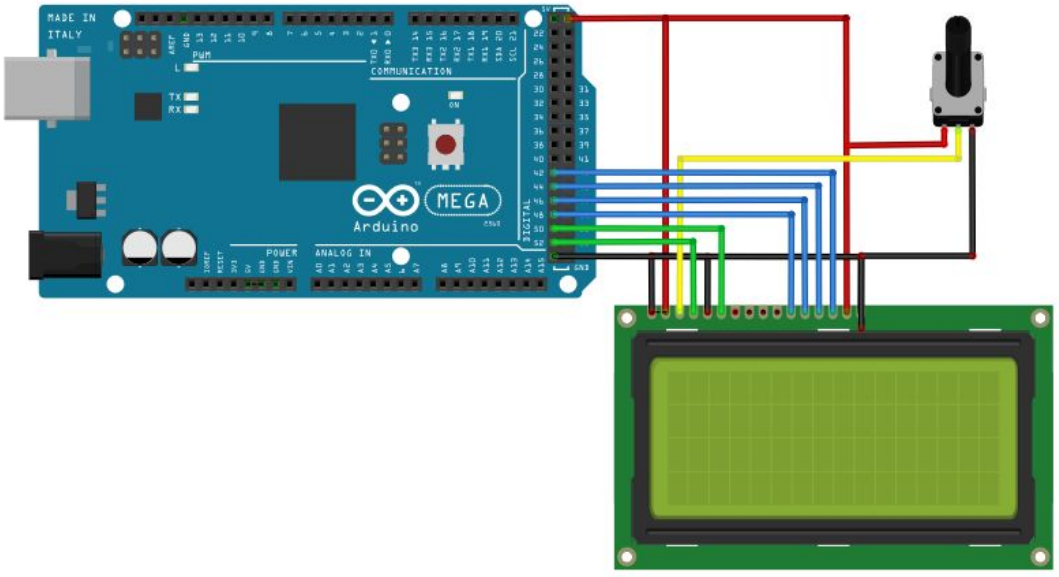
```
lever_d =         readValue(PIN_LEVER_D);
lever_r =         readValue(PIN_LEVER_R);
sw_lights_on =    readValue(PIN_SW_LIGHTS_ON);
sw_lights_auto =  readValue(PIN_SW_LIGHTS_AUTO);
sw_warnings =     readValue(PIN_SW_WARNINGS);
sw_ultrasonic =   readValue(PIN_SW_ULTRASONIC);
sw_cam =          readValue(PIN_SW_CAM);
btn_stop =        readValue(PIN_BTN_STOP);
btn_cal =         readValue(PIN_BTN_CAL);
btn_cal_mpu=      readValue(PIN_BTN_CAL_MPU);
btn_claxon =      readValue(PIN_BTN_CLAXON);


int readValue(int pin) {
  return digitalRead(pin) ? DATA_LOW : DATA_HIGH;
}
```

Dependiendo de la posición del *switch* del control del gimbal se enviarán los datos recogidos por el acelerómetro o por el joystick.

```
if (sw_cam == 2) {
  c_yaw =   map(analogRead(PIN_JOY_X), 0, 1023, 0, 180);
  c_pitch = map(analogRead(PIN_JOY_Y), 0, 1023, 0, 180);
} else {
  c_yaw =   map(yaw, -10, 10, 0, 180);
  c_pitch = map(pitch, -10, 10, 0, 180);
}
```

f. Pantalla LCD

El volante dispone de una pantalla LCD para mostrar mensajes, que se conecta de la siguiente forma:



Para programar la pantalla, se hace uso de la librería `<LiquidCrystal.h>` de Arduino.

Se definen los pines de la pantalla LCD en la zona de configuración y se inicializa en el `setup()`:

```
LiquidCrystal lcd(E, RS, D4, D5, D6, D7);

lcd.begin(COLS, ROWS);
```

Para escribir en la pantalla se utilizan las siguientes funciones de la librería.

```
lcd.clear();
lcd.setCursor(0, 0);
lcd.print(text_0);
lcd.setCursor(0, 1);
lcd.print(text_1);
```

4. Programación del Receptor

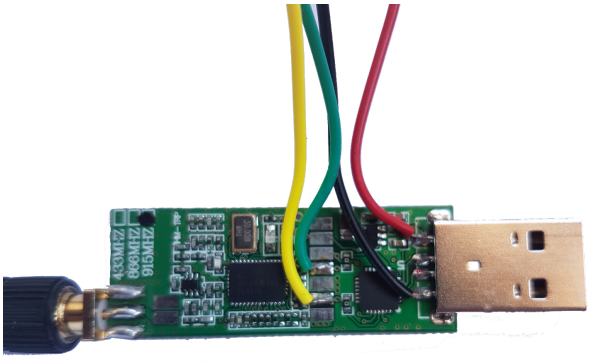
El receptor se encuentra en el coche y se programa en un Arduino Nano. Se divide el código según los diferentes bloques que lo componen:

a. Zona de configuración

Tiene la misma función que la del transmisor.

b. Recepción de datos por radio

Una de las dos radios está diseñada para ser utilizada mediante un USB, por lo tanto para poder conectarla al arduino es necesario soldar en su placa los cables necesarios para tener TX, RX, VCC y GND.



Para programarlas se hace uso de la librería <SoftwareSerial.h>. Esta librería ha sido desarrollada para permitir comunicación serie por cualquier par de pines digitales. El Arduino tiene incorporados en los pines 0 y 1 este tipo de comunicación, pero se opta por usar la librería ya que si se usaran estos pines habría problemas al depurar (los mensajes mostrados en la consola del ordenador utilizan dichos pines)
Para crear una comunicación serie en dos pines se utiliza la siguiente sentencia:

```
SoftwareSerial radioSerial(PIN_RADIO_TX, PIN_RADIO_RX);
```

Las radios elegidas funcionan a **57600** baudios por tanto se fija su uso a estos baudios en el setup():

```
radioSerial.begin(RADIO_BAUDRATE);
```

Se comprueba que haya datos disponibles para proceder a su lectura. Para decodificarlos se buscan los caracteres 's' y 'e', que marcan el inicio y el fin de un bloque, respectivamente.

```
while (radioSerial.available()) {
  incomingByte = radioSerial.read();
  delay(1);
  if (start == true) {
    if (incomingByte != 'e') {
      readBuffer += char(incomingByte);
    } else {
      start = false;
    }
  } else if (incomingByte == 's') {
    start = true;
    stringComplete = true;
  }
}
```

Una vez completado el *string*, se separan y se almacenan los datos en un array de números enteros, mediante la función string_data(). Haciendo uso de check_values(), se comprueba que los valores recibidos están dentro de unos intervalos, para detectar errores de envío, si son correctos se asignan a las variables correspondientes, cambiando previamente los que en el receptor fueron modificados por DATA_LOW (2) y DATA_HIGH (3), por 0 y 1, respectivamente.

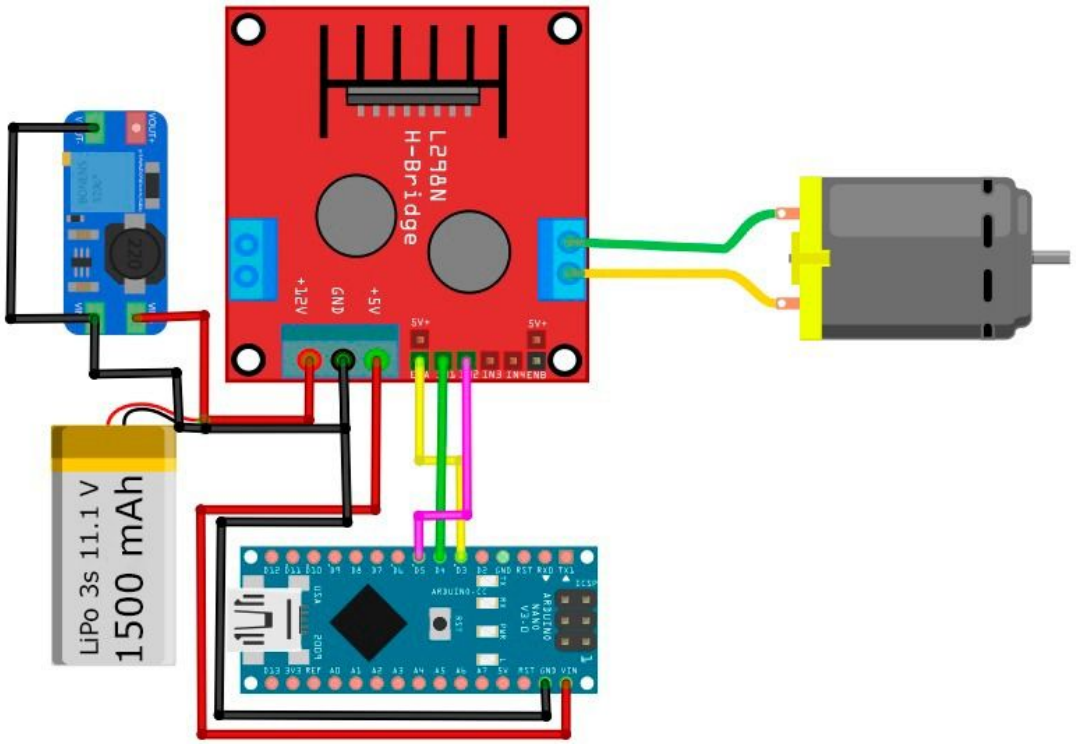
```
void string_data(String sdata){
  for (int i = 0; i < DATA_LENGTH; i++) {
    int index = sdata.indexOf(separator);
    data[i] = sdata.substring(0, index).toInt();
    sdata = sdata.substring(index + 1);
  }
}


bool check_values(int data[]) {
  if (data[0] >= MIN_MOTOR && data[0] <= MAX_MOTOR &&
      data[1] >= MIN_DIR && data[1] <= MAX_DIR &&
      data[2] >= MIN_YAW && data[2] <= MAX_YAW &&
      data[3] >= MIN_PITCH && data[3] <= MAX_PITCH) {
    // checking flags
    for (int i = 4; i < DATA_LENGTH-1; i++) {
      if (data[i] != DATA_LOW && data[i] != DATA_HIGH) {
        return false;
      }
    }
    return true;
  }
  return false;
}
```

En el caso de que el receptor dejase de recibir datos durante más de 1 segundo, se para el motor para evitar posibles accidentes.

```
unsigned long currentMillis = millis();
if ((unsigned long)(currentMillis - prevMillisSignal) >=
intervalNoSignal) {
  while (!radioSerial.available()) {
    analogWrite(PIN_ME, 0);
  }
}
```

c. Movimiento motor

Para mover el motor de corriente continua es necesario hacer uso del controlador L298N, que además tiene una salida de 5V utilizada para alimentar el arduino. Se alimenta todo con una batería de 12V, a la que también se conecta un regulador DC-DC con el que se obtienen 5V para alimentar los servos y la radio, ya que si se conectaran directamente al arduino no les llegaría suficiente corriente.



Tiene tres pines de entrada que se inicializan en el `setup()`:

```
pinMode(PIN_ME, OUTPUT);
pinMode(PIN_M1, OUTPUT);
pinMode(PIN_M2, OUTPUT);
```

Mediante la función analogWrite se controla la velocidad del motor y en función de la posición del switch se hará que el coche vaya marcha adelante (*lever_d*), marcha atrás(*lever_r*) o que se quede parado.

```
analogWrite(PIN_ME, motor);

if (lever_d) {
  digitalWrite(PIN_M1, HIGH);
  digitalWrite(PIN_M2, LOW);
} else if (lever_r) {
  digitalWrite(PIN_M1, LOW);
  digitalWrite(PIN_M2, HIGH);
} else {
  digitalWrite(PIN_M1, LOW);
  digitalWrite(PIN_M2, LOW);
}
```

d. Movimiento dirección

Para el control del servo de dirección, se necesita la librería `<Servo.h>` de Arduino.

Se inicializa el servo:

```
Servo servo_dir;
servo_dir.detach();
servo_dir.attach(PIN_DIR);
```

Se realiza el `detach()` y `attach()` cada vez para eliminar las interrupciones que se producen en los servos. Se escribe el valor enviado por el transmisor para el movimiento del servo:

```
servo_dir.write(dir);
```

e. Movimiento gimbal

Para evitar problemas de interrupciones, los dos servos que controlan la cámara no se programan con la librería servo. Se mueven mediante el siguiente método, al que se le pasa el pin y el ángulo del servo que se quiere move:

```
void moverServo(int pin, int angulo){
   float pausa;
   pausa = angulo * 2000.0/180.0 + 700;
   digitalWrite(pin, HIGH);
```

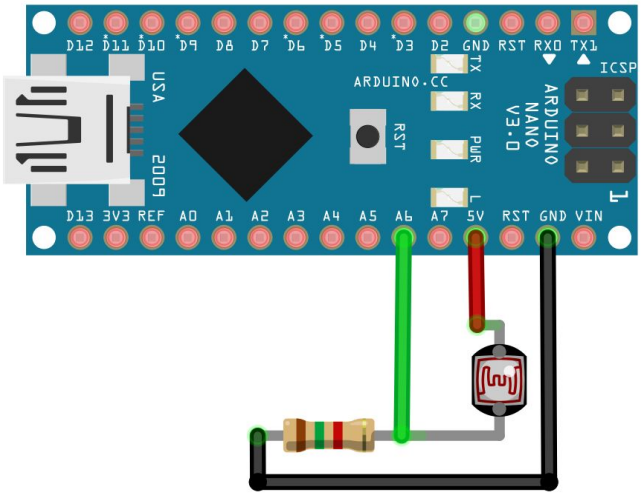
```
    delayMicroseconds(pausa);
    digitalWrite(pin, LOW);
    delayMicroseconds(25000-pausa);
}

moverServo(PIN_CAM_YAW, c_yaw);
moverServo(PIN_CAM_PITCH, c_pitch);
```

f. Botón de emergencia

Si el botón de emergencia es pulsado el coche se queda parado durante 10 segundos, poniendo un delay al programa. Esta función puede ser modificada para realizar otras funciones como que el coche pare hasta que se reinicie o que se produzca un pitido del claxon continuo.

g. Luces del coche

En el coche se hace uso de leds de 5 mm para diferentes funciones: luces de cruce (delanteras y traseras), luces de emergencia (podrían ser usadas como intermitentes) y luces de marcha atrás. Además, tiene un sensor de luz, que se conecta con una resistencia (alrededor de 1.5 kΩ) realizando un divisor de tensión. Por lo tanto, la entrada analógica A6 leerá un valor idealmente entre 0 (ausencia de luz, resistencia LDR = infinito) y 1023 (mucha luz, resistencia LDR = 0).



Se definen los pines en la zona de configuración y en el `setup()` se declaran como salida.

```
pinMode(PIN_LIGHT_F1, OUTPUT); // Front left headlights
pinMode(PIN_LIGHT_F2, OUTPUT); // Front right headlights
pinMode(PIN_LIGHT_B1, OUTPUT); // Back headlights
pinMode(PIN_LIGHT_R1, OUTPUT); // Reverse headlights
pinMode(PIN_LIGHT_W1, OUTPUT); // Front left warning lights
pinMode(PIN_LIGHT_W2, OUTPUT); // Front right warning lights
pinMode(PIN_LDR, INPUT);       // Sensor de luz
```

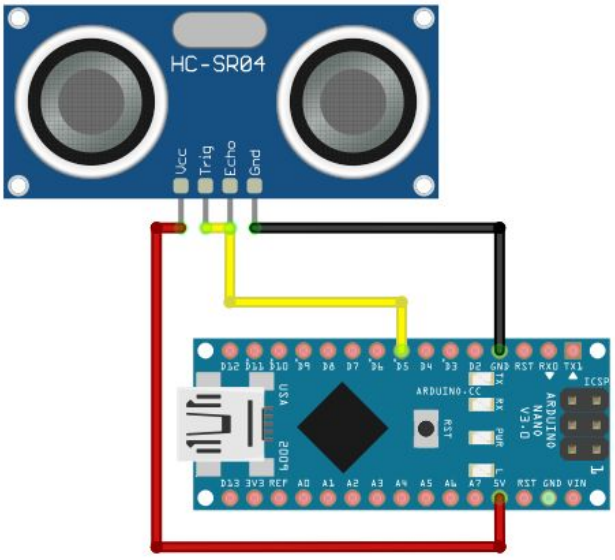
Según los valores recibidos se procede al encendido o apagado de las luces, para lo cual se hace uso de la función `checkControls()`, que enciende las luces delanteras y las traseras si el switch está activo en la posición *lights_on*, o en la posición *lights_auto* si la luz que percibe el sensor es menor a un umbral. También hace parpadear las luces de warnings mediante el método `blink_lights()`.
Además al mover el motor marcha atrás se activan las luces LIGHT_R1 , y cuando se pulsa el botón de emergencia las de warnings.

```
void checkControls() {
  if ((sw_lights_auto && checkLightIntensity()) || sw_lights_on) {
    digitalWrite(PIN_LIGHT_F1, HIGH); // Front left headlights
    digitalWrite(PIN_LIGHT_F2, HIGH); // Front right headlights
    digitalWrite(PIN_LIGHT_B1, HIGH); // Back headlights
  } else {
    digitalWrite(PIN_LIGHT_F1, LOW);  // Front left headlights
    digitalWrite(PIN_LIGHT_F2, LOW);  // Front right headlights
    digitalWrite(PIN_LIGHT_B1, LOW);  // Back headlights
  }

  if (sw_warnings) {
    blink_lights();                   // Function that blink lights
  } else {
    digitalWrite(PIN_LIGHT_W1, LOW);  // Front left warning lights
    digitalWrite(PIN_LIGHT_W2, LOW);  // Front right warning lights
  }
}

void blink_lights() {
  unsigned long currentMillis = millis();
```

```
  if ((unsigned long)(currentMillis - prevMillisBlink) >= intervalWarn){
    state_warning = !state_warning;
    digitalWrite(PIN_LIGHT_W1, state_warning);
    digitalWrite(PIN_LIGHT_W2, state_warning);
    prevMillisBlink = millis();
  }
}
```

h. Sensor ultrasonido

El coche dispone de un sensor de ultrasonidos que es programado mediante la librería `<NewPing.h>`, y se conecta de la siguiente forma:



En la zona de configuración se inicializa:

```
NewPing sonar(PIN_ULTRASONIC, PIN_ULTRASONIC, MAX_DISTANCE);
```

Si se detecta un obstáculo a una distancia que esté entre MAX_DIST_BRAKE y MIN_DIST_BRAKE, no permite que el coche vaya marcha adelante. Se ha definido una distancia mínima para evitar que cuando el sensor falle y detecte una distancia 0 se pare el coche.

```
dist = sonar.ping_cm();
```

```
if (lever_d && dist < MAX_DIST_BRAKE && dist > MIN_DIST_BRAKE) {
  Serial.println("!!!------------------------!!!");
  Serial.println("!!!----EMERGENCY BRAKE!----!!!");
  Serial.println("!!!------------------------!!!");
  analogWrite(PIN_ME, 0);
}
```

i. Claxon

Para hacer uso del *buzzer* se utilizan las funciones `tone()` y `noTone()`. Si no se modifica la librería para controlar el sensor de ultrasonidos, se genera un conflicto, por lo que las soluciones son modificar la librería (Cambiar el valor de `TIMER_ENABLED = false` en `<NewPing.h>`) o usar otra librería para controlar el *buzzer*.

Se declara el pin del *buzzer* y la frecuencia del sonido en la zona de configuración y se declara como salida en el `setup()`. Se activar el zumbador en caso de que el botón se haya pulsado.

```
if (btn_claxon) {
  tone(PIN_CLAXON, FREC_CLAXON);
} else {
  noTone(PIN_CLAXON);
}
```

5. Cámara del coche

Se utiliza un sistema de comunicaciones independiente. Se trata de una cámara FPV, que tiene su propio transmisor y receptor. El receptor (EACHINE R051 está diseñado para que simplemente conectándolo a un móvil haciendo uso de la aplicación EACHINE, disponible en Android e iOS, se pueda ver lo que la cámara está emitiendo. El emisor (TS832S) se conecta directamente a la cámara y a una fuente de 12 VDC (se conecta directamente a la batería del coche).
El sistema funciona a una frecuencia de 5.8 GHz.