STRD – Detección de distracciones al volante

Javier Alonso Silva
Alfonso Díez Ramírez
Sara Moreno Prieto
Mihai Octavian Stănescu

2021

Resumen

Se desarrolla un sistema de detección de distracciones al volante el cual se espera ayude a evitar los posibles accidentes derivados de la casuística anterior.

El desarrollo consiste en una evaluación de los requisitos, modelado del sistema mediante diagramas SysML hasta una implementación final en dos nodos diferenciados los cuales se comunican entre sí usando la tecnología CANBus.

El primer nodo (*nodo 1*) tendrá una carga balanceada entre la lectura de dispositivos así como la intervención en elementos físicos del vehículo, como son los frenos; y a su vez será el encargado de una transmisión continua de mensajes hacia el segundo nodo. El *nodo 2* leerá información sobre el estado psico-físico del conductor y, junto con la información recibida del *nodo 1*, alertará al mismo sobre distintos factores que se han visto peligrosos para que pueda reconducir su comportamiento. Finalmente, se ofrece al conductor un método para evitar ser distraído por el propio sistema pudiendo decidir entre tres niveles de avisos: completo, parcial e inactivo.

Índice

1. Introducción

Una de las mayores causas de accidentes son las distracciones de los conductores al volante, o bien por el uso de dispositivos electrónicos, somnolencia u otras acciones que llevan a la persona a no prestar atención a la carretera y su entorno.

A raíz de ese problema, los mecanismos de regulación internacionales han invertido tiempo, dinero y desarrollo en los sistemas ADAS (*Advanced Driving Assistance Systems*), con el fin de mitigar las situaciones anteriores y realizar una prevención activa sobre los accidentes de tráfico. Sin embargo, dichos sistemas no cuentan con una penetración significativa en el mercado, por lo que interesa agilizar su implantación y que pasen a ser un elemento de seguridad "por defecto" en los nuevos vehículos.

En este contexto, se ha pedido realizar una implementación distribuida, que cumpla con unos requisitos de tiempo real, en dos nodos que interactúan entre sí para actuar como un organismo conjunto sobre un vehículo como sistema ADAS.

El sistema a desarrollar contará con múltiples sensores:

- Giroscopio, para detectar en los ejes X y Y la inclinación de la cabeza del conductor y predecir una posible somnolencia.
- Giro del volante, para detectar si el conductor está pegando volantazos o está realizando "mini–correcciones", características de un estado de somnolencia o de atender al móvil.
- Agarre del volante, donde se indicará si el conductor está agarrando el volante o no.
- Velocímetro, con un rango de valores comprendido entre los $[0, 200]$ $^{km}/_{h}$. Se usará para comprobar que se cumple la distancia de seguridad.
- Sensor de distancia, capaz de realizar lecturas en el rango $[5, 200]$ m y que le indicará al conductor si está cumpliendo o no la distancia de seguridad, según la velocidad a la que circule.

y múltiples actuadores:

- Luces de aviso, las cuales se usarán para emitir señales luminosas al conductor indicando cierto nivel de riesgo que se está produciendo.
- *Display*, usado para visualizar los datos que obtiene el sistema.
- Alarma sonora, emitiendo un sonido con 3 niveles de intensidad.
- Luz de aviso/freno automático, donde ante un peligro de colisión inminente el sistema podrá activar el freno con hasta 3 niveles de intensidad.

Cada uno de los sensores/actuadores estarán controlados y monitorizados por una o varias tareas las cuales registran los datos en objectos protegidos. Dichas tareas vienen definidas con sus periodos y *deadlines* en el cuadro 1:

Tareas/objetos protegidos	**Tipo**	T_i	D_i	**WCET**	**Síntomas 1**	**Síntomas 2**	**Modo**
Inclinación cabeza	C	600	400	?	x_1		
Detección de volantazos	C	400	400	?	x_1		
Cálculo distancia	C	300	300	?		y_1	
Relax al volante	C	500	200	?	x_1		
Emergencias	C	300	300	?	x_2	y_2	z_2
Mostrar información	C	2000	2000	?	x_2	y_2	
Detección pulsador	S	-	100	?			z_1
Síntomas 1	P	-	-	x_1, x_2			
Síntomas 2	P	-	-	y_1, y_2			
Modo	P	-	-	z_1, z_2			

Cuadro 1: Listado de tareas y objetos protegidos junto con sus tiempos.

Como hay multitud de tareas y se cuenta con dos nodos, el sistema a implementar irá distribuído entre ambos y viene representado por la figura 1:

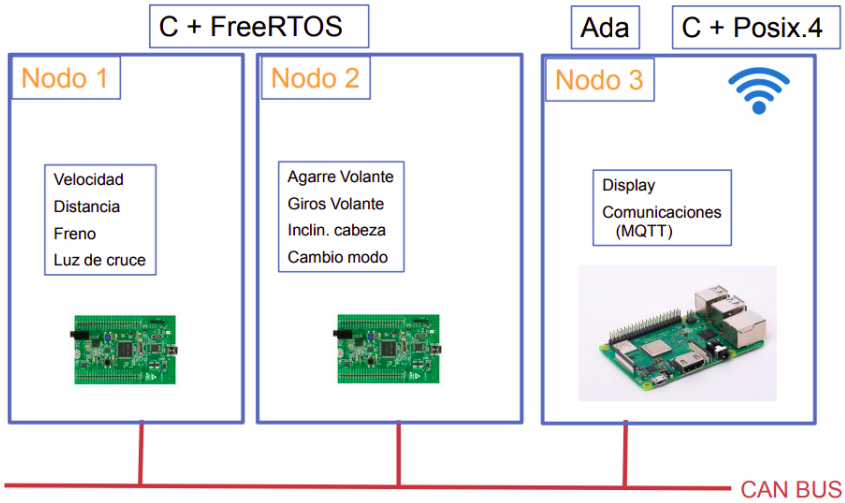

Figura 1: Modelo completo del sistema a implementar. Las tareas van distribuídas entre los dos nodos principales y se comunican entre ellos mediante CANBus.

1.1. Nodo 1

El primer nodo será el encargado principalmente de la actuación sobre distintos elementos del sistema, a saber: el freno, las luces de cruce e indirectamente sobre la alarma. Esto lo hará recogiendo datos de distintos sensores como son el velocímetro, el sensor de distancia y el sensor de luminosidad para adecuar su comportamiento a las circunstancias del entorno.

Este sistema contará con cuatro tareas en tiempo real y usará dos objetos protegidos: el primero de ellos para conservar el valor de la velocidad actual; y el segundo para guardar tanto el valor de la distancia con el vehículo precedente como la intensidad del freno que se ha de

aplicar en caso de peligro de colisión. Por su parte, las tareas en cuestión son:

1. `Cálculo velocidad` – cada 250 ms, realizará una lectura del sensor en cuestión mediante el ADC y actualizará el valor del objeto protegido `V_actual`.

2. `Cálculo distancia` – cada 300 ms, el sistema obtendrá la distancia con el vehículo precedente usando el sensor de ultrasonidos y actualizará el valor del objeto protegido `D_actual`. Además, leerá el valor de `V_actual` y computará lo que sería la distancia de seguridad mínima que hay que respetar, descrita por la ecuación 1:

$$d_{\text{mín}} = \left(\frac{V}{10}\right)^2, \quad \begin{cases} d_{\text{mín}} & : \text{distancia mínima que hay que mantener.} \\ V & : \text{velocidad actual del vehículo.} \end{cases} \tag{1}$$

   En caso de que la distancia de seguridad no se cumpla (y según el valor relativo con que no se cumple), la tarea indicará en `Intens_Frenada` con qué intensidad se ha de aplicar el freno para evitar una colisión. Finalmente, activará la tarea esporádica `Freno` para que realice su ejecución.

3. `Freno` – cada 150 ms como mucho, realizará la activación progresiva del freno cada 100 ms hasta alcanzar la intensidad apropiada. Al ser una tarea esporádica, depende directamente de la activación desde `Cálculo distancia`, lo cual añadirá un *jitter* al tiempo de respuesta global de la tarea.

4. `Luces de cruce` – cada 1 000 ms, el sistema realizará una valoración de la luminosidad del entorno y procederá a encender o apagar automáticamente las luces de cruce. Se establece que las luces se activarán si la intensidad lumínica está por debajo de 100.

Todo este sistema viene modelado por la figura 2:

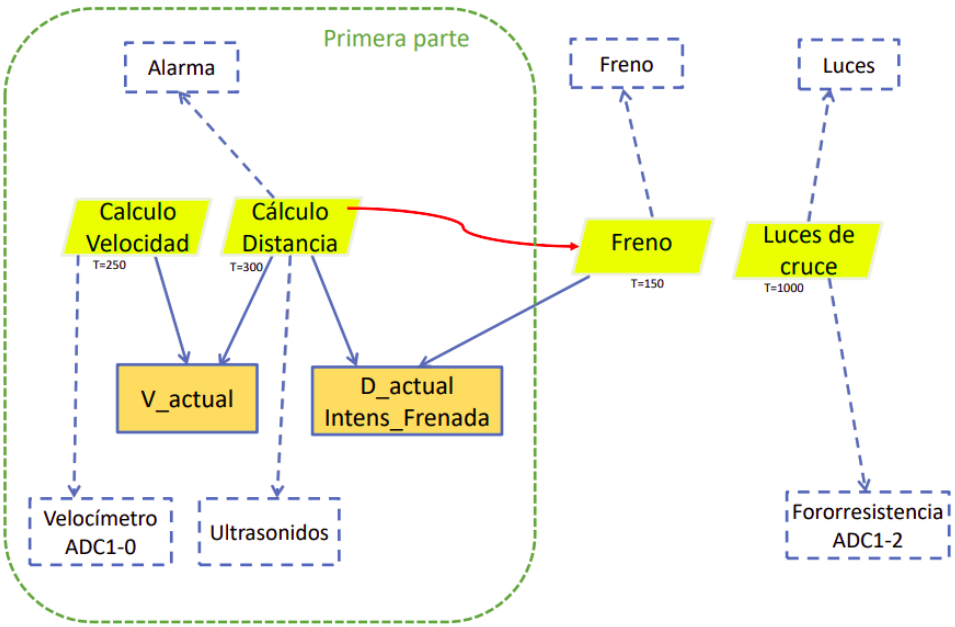

Figura 2: Modelado del nodo 1 junto con sus tareas, objetos protegidos, sensores y actuadores.

1.2. Nodo 2

El segundo nodo se encargará directamente de notificar al conductor cuando algún comportamiento es errático o peligroso. Entre otras tareas, este nodo se encarga de monitorizar el estado del conductor (y detectar posibles signos de somnolencia) y emitir avisos luminosos y sonoros cuando se produzcan situaciones de riesgo.

Este sistema cuenta con cinco tareas en tiempo real y tres objetos protegidos: el primero recoge datos sobre síntomas como son la inclinación de la cabeza o el giro del volante; el segundo, recoge información sobre si el conductor está sujetando o no el volante; y el tercero establecerá el modo de funcionamiento de los avisos del sistema. Con respecto a las tareas, se tiene:

1. `Inclinación cabeza` – cada 600 ms, leerá el valor del giroscopio integrado para actualizar los datos de las posiciones X e Y, en el objeto protegido `Síntomas 1`.

2. `Detección volantazos` – cada 400 ms el sistema leerá el valor de la posición del volante y actualizará el dato recogido en `Síntomas 1`. Si durante dos lecturas consecutivas la diferencia entre las posiciones del volante es de más de 150 y la velocidad es mayor a $70^{km}/_{h}$ entonces se considera que el conductor está dando volantazos. Si pasan más de 5 segundos sin que se repita esa situación, el conductor estará conduciendo normalmente.

3. `Relax al volante` – cada 500 ms, el sistema actualizará en `Síntomas 2` si el conductor está sujetando o no el volante.

4. `Detección pulsador` – tarea esporádica que será activada desde la rutina de tratamiento de interrupciones *hardware* que establecerá cíclicamente el modo de funcionamiento del sistema en el objeto protegido `Modo`.

5. `Riesgos` – cada 300 ms, el sistema evaluará los datos recogidos en los objetos protegidos `Síntomas 1`, `Síntomas 2` y `Modo` y establecerá el nivel de alarma para con el conductor. Dicha detección de riesgos viene definida por la siguiente secuencia:

   - S_1 – si el conductor presenta una inclinación de la cabeza en los ejes X, Y de más de $20°$ y no tiene sujeto el volante se considera que está manipulando el móvil u otro aparato. Se activa la luz amarilla y se emite un pitido nivel 1.
   - S_2 – si la inclinación de la cabeza es $X > 20° | Y > 20°$, el volante está agarrado y la velocidad es mayor de $70^{km}/_{h}$ se interpreta que el conductor no está prestando atención a la carretera y se encenderá la luz amarilla.
   - S_3 – si se detecta una inclinación en el eje X de más de $30°$ y el conductor está dando volantazos se interpreta como síntoma de somnolencia. Se encenderá la luz amarilla y se emitirá un pitido nivel 2.
   - S_4 – si se dan simultáneamente dos de los riegos anteriores se pasa a estar en **NIVEL 2** de alerta y se encenderá la luz roja y emitirá un pitido nivel 2.
   - S_5 – si se produce un riesgo **NIVEL 2** y la distancia con el vehículo precedente es menor al 50 % de la distancia de seguridad recomendada, se estará ante una situación de **EMERGENCIA** y se activará el freno, junto con todo lo anterior.

La evaluación de riesgos se puede modelar mediante el diagrama 3:

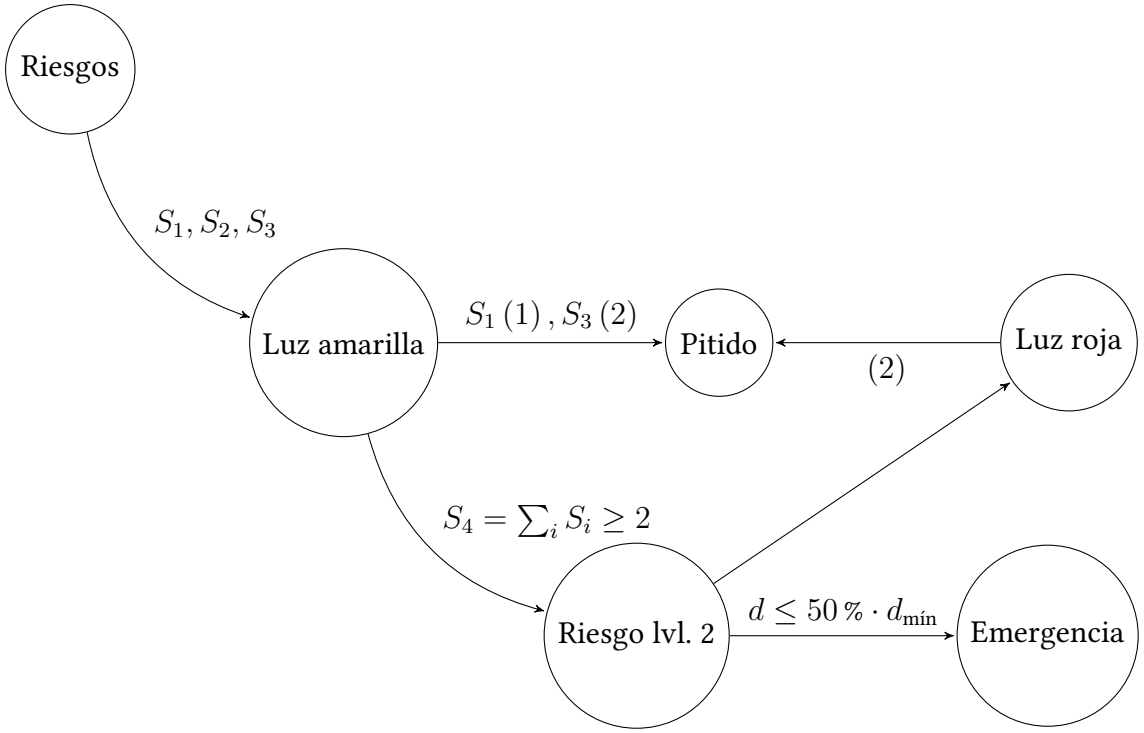

Figura 3: Diagrama que modela la interpretación de los riesgos, descritos en la enumeración anterior (S_i). La intensidad del pitido va acompañada entre paréntesis del síntoma que lo activa (por ejemplo, $S_1\,(1)$ indica una intensidad de pitido nivel 1) o en solitario, si es consecuencia de acciones en cadena.

Y, en general, el nodo 2 se puede representar mediante la figura 4:

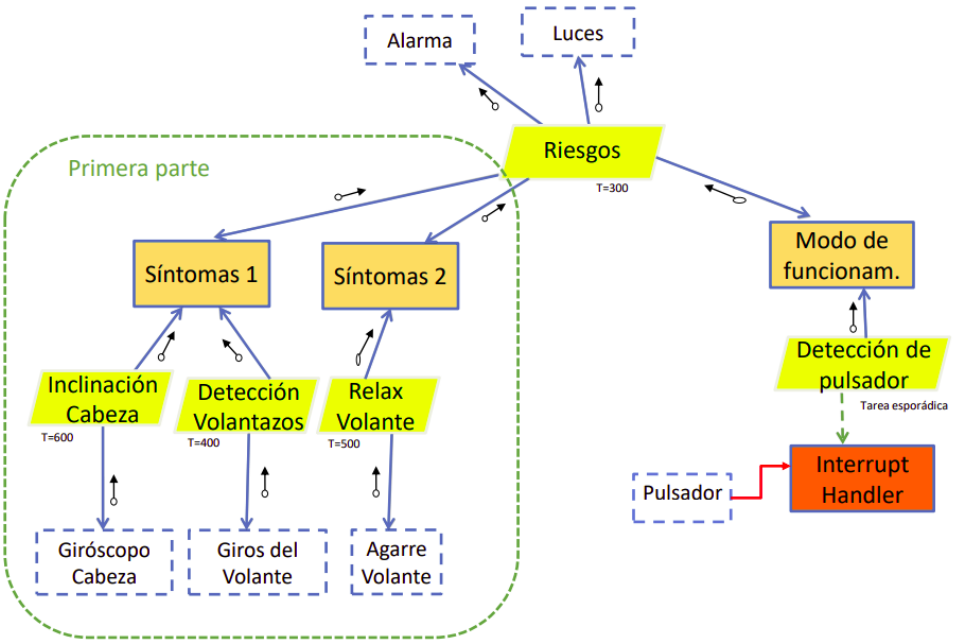

Figura 4: Modelado del nodo 2 junto con sus tareas, objetos protegidos, sensores y actuadores.

2. Implementación

Una vez se ha introducido el sistema, se va a explicar la implementación que se ha realizado finalmente en cada uno de los nodos. Como esta memoria es de explicación del código y de las decisiones tomadas, se incluirán distintos fragmentos del mismo para acompañar a las explicaciones y entrar en mayor o menor detalle en las funciones.

Por otra parte, se va a explicar qué tareas se han implementado correctamente en cada uno de los nodos y cómo se han implementado.

Finalmente, destacar que hay fragmentos de código fuente que son comunes a ambos nodos y que no aparecerán explicados en detalle por cada nodo sino que se indican en el anexo A.

2.1. Nodo 1

En el nodo 1 se han implementado en principio todas las tareas cumpliendo con las restricciones pedidas.

La tarea del `Cálculo de velocidad` viene definida por el listado de código 1:

```
90  /**
91   * @brief Tarea periódica (250 ms) que lee y actualiza el valor de la
92   *        velocidad del vehículo. Además, en cada iteración envía los
93   *        datos de la velocidad actualizados al nodo 2.
94   *
95   * @param argument lista de posibles argumentos a usar. Vacía por defecto
       ↪ .
96   */
97  void acelerador(const void *argument) {
98    int speed;
99    uint32_t wake_time = osKernelSysTick();
100   while(true) {
101     ADC_ChannelConfTypeDef sConfig = {0};
102     sConfig.Channel = ADC_CHANNEL_0;
103     sConfig.Rank = 1;
104     sConfig.SamplingTime = ADC_SAMPLETIME_28CYCLES;
105     HAL_ADC_ConfigChannel(&hadc1, &sConfig);
106     HAL_ADC_Start(&hadc1);
107     if (HAL_ADC_PollForConversion(&hadc1, 5) == HAL_OK) {
108       speed = map(HAL_ADC_GetValue(&hadc1), 0, 255, 0, 200);
109       SPEED_set(speed);
110       CAN_sendi(speed);
111     }
112     osDelayUntil(&wake_time, T_TAREAVELOCIDAD);
113   }
114 }
```

Listing 1: Tarea periódica que controla el acelerador.

En la susodicha tarea se lee el ADC desde el canal 0 y el valor recibido de la velocidad se mapea de 0 a $200^{km}/_{h}$ (línea 108). A continuación, se actualiza el valor del objeto protegido (línea 109) y se envía el dato recibido por el CANBus (línea 110). Finalmente, se programa la siguiente ejecución dentro de 250 ms desde el instante de activación (línea 112).

La función de `map` viene definida en los códigos 3 y 4. El objeto protegido `SPEED` sigue la definición estándar del resto de objetos protegidos y viene definido en los códigos 5 (cabeceras) y 7 (cuerpo).

La tarea del `Cálculo distancia` viene definida por el código 2:

```
116 /**
117  * @brief Tarea periódica (300 ms) que lee y actualiza el valor de la
118  *        distancia con el vehículo precedente. Además, en cada iteración
119  *        se envía el valor de la distancia por el CANBus al nodo 2 y,
     ↪ además,
120  *        se computa el valor de la intensidad de frenada para activar (o
     ↪  no)
121  *        a la tarea esporádica #brake_task.
122  *
123  * @param args lista de posibles argumentos a usar. Vacía por defecto.
124  */
125 void distanceTask(const void *args) {
126   const uint16_t T_DISTANCE_TASK = 300U;
127   uint32_t wake_time = osKernelSysTick();
128   float distance;
129   float speed;
130   float secure_dist;
131   int old_intensity = 0;
132   int intensity = 0;
133   HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_RESET);
134   while (1) {
135     distance = (float) USS_read_distance() * 0.00171821F;
136     if (distance == 500000)
137       distance = 1;
138     DISTANCE_set(distance);
139
140     speed = SPEED_get();
141     secure_dist = (float) pow((speed / 10), 2);
142
143     if (distance < secure_dist) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9,
     ↪ GPIO_PIN_SET);
144     else HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_RESET);
145
146     old_intensity = BRAKE_intensity_get();
147     if (distance ⩽ secure_dist)
148       intensity = 4;
149     else if (distance ⩽ 2 * secure_dist)
150       intensity = 3;
151     else if (distance ⩽ 3 * secure_dist)
152       intensity = 2;
153     else if (distance ⩽ 4 * secure_dist)
154       intensity = 1;
155     else
156       intensity = 0;
157
158     if (intensity ≠ old_intensity) {
159       BRAKE_intensity_set(intensity);
160       BRAKE_set_event();
161     }
162     CAN_sendf(distance);
163     osDelayUntil(&wake_time, T_DISTANCE_TASK);
164   }
```

```
165 }
```

Listing 2: Tarea periódica que controla la distancia.

En dicha tarea se utiliza la librería `uss` (códigos 6 y 8) para leer desde el sensor de ultrasonidos (líneas 135 – 137); se actualiza el valor de la distancia en el objeto protegido `distance` (línea 138) (códigos **??** y **??**); se computa la distancia de seguridad y se calcula la intensidad de la frenada según unos porcentajes establecidos (líneas 140 – 156);

2.2. Nodo 2

3. Diseño final

4. Aclaraciones

- En el código 1, el mapeo se realiza con valores de entrada $[0, 255]$ porque el ADC de la placa es de 8 bits, por lo que su resolución máxima es 255.

5. Glosario

A. Código fuente común

A.1. Cabeceras de código

```c
/*
 * Copyright © 2021 - present | utils.h by Javinator9889
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program 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 General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see https://www.gnu.org/licenses/.
 *
 * Created by Javinator9889 on 10/04/21 - utils.h.
 */
#ifndef UTILS_H
#define UTILS_H
#include <stdint.h>

// Gets the size of an array
#define arrsize(array) (sizeof (array) / sizeof *(array))
```

```
25
26 // Iterates through an array
27 #define foreach(idxtype, item, array) \
28     idxtype* item; \
29     size_t size = arrsize(array); \
30     for (item = array; item < (array + size); ++item)
31
32 /**
33  * @brief Custom datatype representing the union of
34  *        a float value and its representation as a
35  *        array of four bytes. Useful when converting
36  *        from float to bytes and viceversa.
37  */
38 typedef union float_u {
39     float float_var;
40     uint8_t bytes_repr[4];
41 } FloatU_t;
42
43
44 int map(int, int, int, int, int);
45
46 void f2b(float, uint8_t*);
47 float b2f(uint8_t*);
48
49 #endif /* UTILS_H */
```

Listing 3: Cabecera con funciones de utilidad.

A.2. Cuerpo del código

```
1  /*
2   * Copyright © 2021 - present | utils.c by Javinator9889
3   *
4   * This program is free software: you can redistribute it and/or modify
5   * it under the terms of the GNU General Public License as published by
6   * the Free Software Foundation, either version 3 of the License, or
7   * (at your option) any later version.
8   *
9   * This program is distributed in the hope that it will be useful,
10  * but WITHOUT ANY WARRANTY; without even the implied warranty of
11  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
12  * GNU General Public License for more details.
13  *
14  * You should have received a copy of the GNU General Public License
15  * along with this program.  If not, see https://www.gnu.org/licenses/.
16  *
17  * Created by Javinator9889 on 10/04/21 - utils.c.
18  */
19 #include "utils.h"
20
21 /**
22  * @brief Maps a given value in between a given proportional range.
23  *
24  * @param x         the value to map.
25  * @param in_min    the minimum input value to map.
26  * @param in_max    the maximum input value to map.
```

```
 * @param out_min   the minimum output value to produce.
 * @param out_max   the maximum output value to produce.
 * @return int - the 'x' value mapped in between [out_min, out_max].
 */
int map(int x, int in_min, int in_max, int out_min, int out_max) {
  return (int)((x - in_min) * (out_max - out_min) / (in_max - in_min) +
    ↪ out_min);
}

/**
 * @brief With the given float value, produces the equivalent 4 bytes
 *        representing that value.
 *
 *        Notice that this function relies on that a float is 4 bytes
 *        in memory. Higher (or lower) values will require this method
 *        to be overwritten.
 *
 * @param value the input float to convert.
 * @param bytes the output bytes array (4) to produce.
 */
void f2b(float value, uint8_t bytes[4]) {
    FloatU_t u;

    u.float_var = value;
    memcpy(bytes, u.bytes_repr, 4);
}

/**
 * @brief With the given bytes array, produces the equivalent float value
 *        represented by that 4 bytes.
 *
 *        Notice that this function relies on that a float is 4 bytes
 *        in memory. Higher (or lower) values will require this method
 *        to be overwritten.
 *
 * @param bytes the input bytes array (4) to read.
 * @return float - the converted float data from bytes.
 */
float b2f(uint8_t bytes[4]) {
    FloatU_t u;
    memcpy(u.bytes_repr, bytes, 4);

    return u.float_var;
}
```

Listing 4: Cuerpo de las funciones de utilidad.

B. Código fuente nodo 1

B.1. Cabeceras de código

```
/*
 * Copyright © 2021 - present | speed.h by Javinator9889
 *
```

```
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program 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 General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see https://www.gnu.org/licenses/.
 *
 * Created by Javinator9889 on 05/03/21 - speed.h.
 */
#ifndef SPEED_H
#define SPEED_H
#include <FreeRTOS.h>
#include <stddef.h>
#include <semphr.h>

void SPEED_init(void);
void SPEED_set(int);
int SPEED_get(void);

#endif /* SPEED_H */
```

Listing 5: Cabecera del objeto protegido SPEED.

```
/*
 * Copyright © 2021 - present | uss.h by Javinator9889
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program 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 General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see https://www.gnu.org/licenses/.
 *
 * Created by Javinator9889 on 26/03/21 - uss.h.
 */
#ifndef USS_H
#define USS_H
#include <stdint.h>

uint32_t USS_read_distance(void);

#endif
```

Listing 6: Cabecera del controlador de ultrasonidos.

B.2. Cuerpo del código

```c
/*
 * Copyright © 2021 - present | speed.c by Javinator9889
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program 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 General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see https://www.gnu.org/licenses/.
 *
 * Created by Javinator9889 on 05/03/21 - speed.c.
 */
#include "speed.h"
#include <lock.h>
#include <semphr.h>
#include <FreeRTOS.h>
#include <FreeRTOSConfig.h>
#include <task.h>

// Private variable containing the speed lock.
static Lock_t SPEED_sem = NULL;
// Private variable containing the speed itself.
static int SPEED_speed = 0;

/**
 * @brief Initializes the speed protected object.
 *
 *        This method must be called during the early boot as,
 *        until then, any call to any method will fail and block
 *        forever.
 *
 */
void SPEED_init(void) {
    SPEED_sem = LOCK_create(NULL);
}

/**
 * @brief Safely updates the stored speed value.
 *
 * @param speed the new speed.
 */
void SPEED_set(int speed) {
    LOCK_acquire(SPEED_sem);
    SPEED_speed = speed;
    LOCK_release(SPEED_sem);
}

/**
 * @brief Safely obtains the stored speed value.
 *
```

```
 * @return int - the speed. If any error occurs, returns -1.
 */
int SPEED_get(void) {
    int speed = -1;
    LOCK_acquire(SPEED_sem);
    speed = SPEED_speed;
    LOCK_release(SPEED_sem);
    return speed;
}
```

Listing 7: Cuerpo del objeto protegido SPEED.

```
/*
 * Copyright © 2021 - present | uss.c by Javinator9889
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program 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 General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see https://www.gnu.org/licenses/.
 *
 * Created by Javinator9889 on 26/03/21 - uss.c.
 */
#include "uss.h"
#include <FreeRTOS.h>
#include <FreeRTOSConfig.h>
#include <task.h>
#include <stm32f4xx_hal.h>
#include "dwt_stm32_delay.h"

/**
 * @brief Reads the measured distance from the ultrasonic sensor.
 *
 * @return uint32_t - the measured distance, in meters.
 */
uint32_t USS_read_distance(void) {
    __IO uint8_t flag = 0;
    __IO uint32_t disTime = 0;

    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_10, GPIO_PIN_SET);
    DWT_Delay_us(10);
    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_10, GPIO_PIN_RESET);

    while(flag == 0) {
        while(HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_11) == GPIO_PIN_SET) {
            disTime++;
            flag = 1;
        }
    }
    return disTime;
}
```

Listing 8: Cuerpo del controlador de ultrasonidos.

C. Código fuente nodo 2