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 **1**

1.1. Nodo 1 2

1.2. Nodo 2 4

2. Implementación **6**

2.1. Nodo 1 6

2.2. Nodo 2 10

3. Diseño final **10**

4. Aclaraciones **10**

5. Glosario **10**

A. Código fuente común **10**

A.1. Cabeceras de código 10

A.2. Cuerpo del código 12

B. Código fuente nodo 1 **18**

B.1. Cabeceras de código 18

B.2. Cuerpo del código 19

C. Código fuente nodo 2 **23**

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:

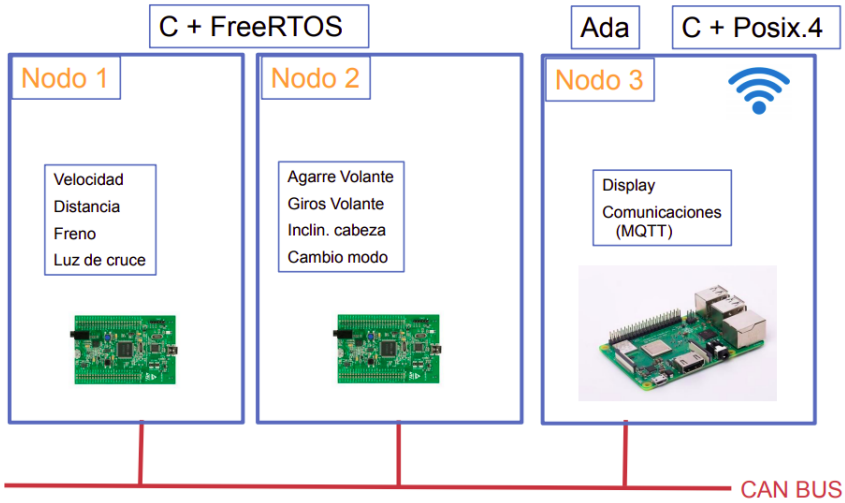

Figura 1: Modelo completo del sistema a implementar. Las tareas van distribuídas entre los dos nodos principales y se comunican entre ellos mediante CANBus.

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:

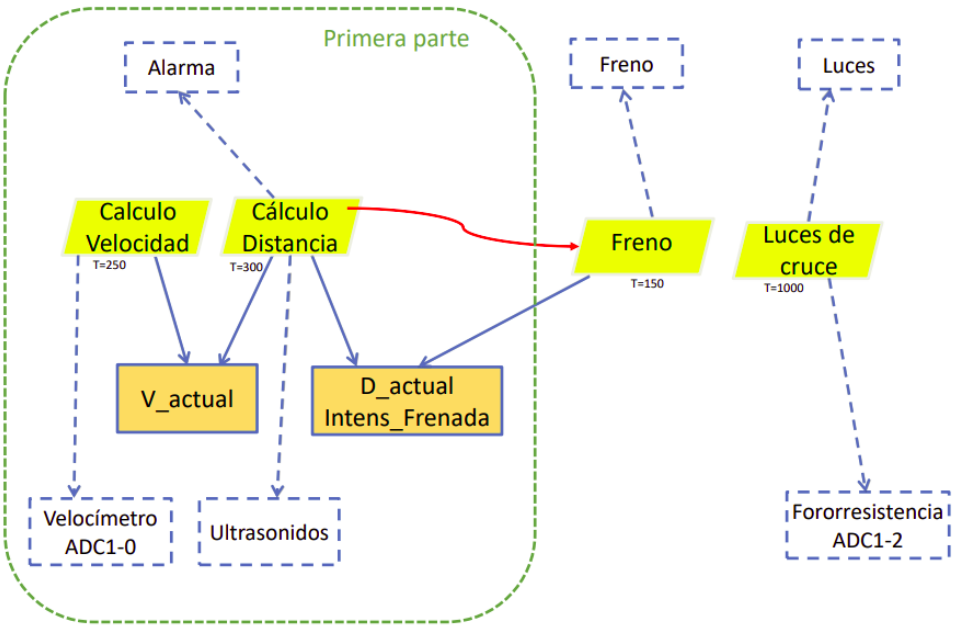

Figura 2: Modelado del nodo 1 junto con sus tareas, objetos protegidos, sensores y actuadores.

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:

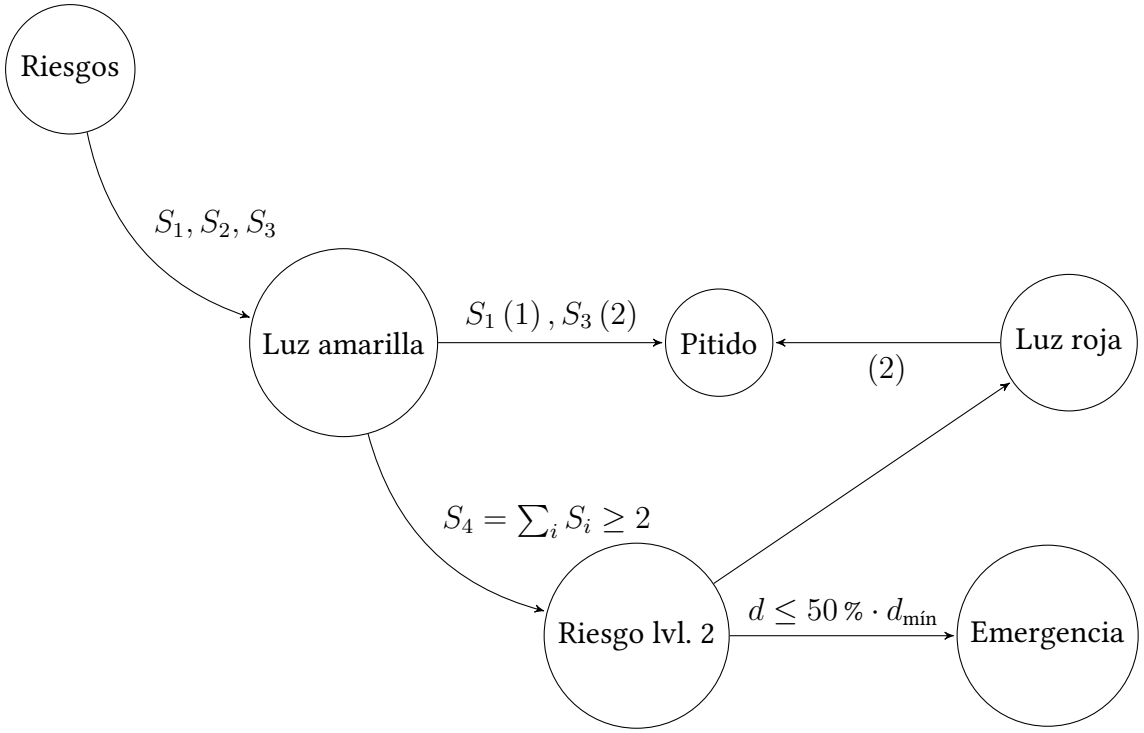

Figura 3: Diagrama que modela la interpretación de los riesgos, descritos en la enumeración anterior (S_i). La intensidad del pitido va acompañada entre paréntesis del síntoma que lo activa (por ejemplo, $S_1\,(1)$ indica una intensidad de pitido nivel 1) o en solitario, si es consecuencia de acciones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

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

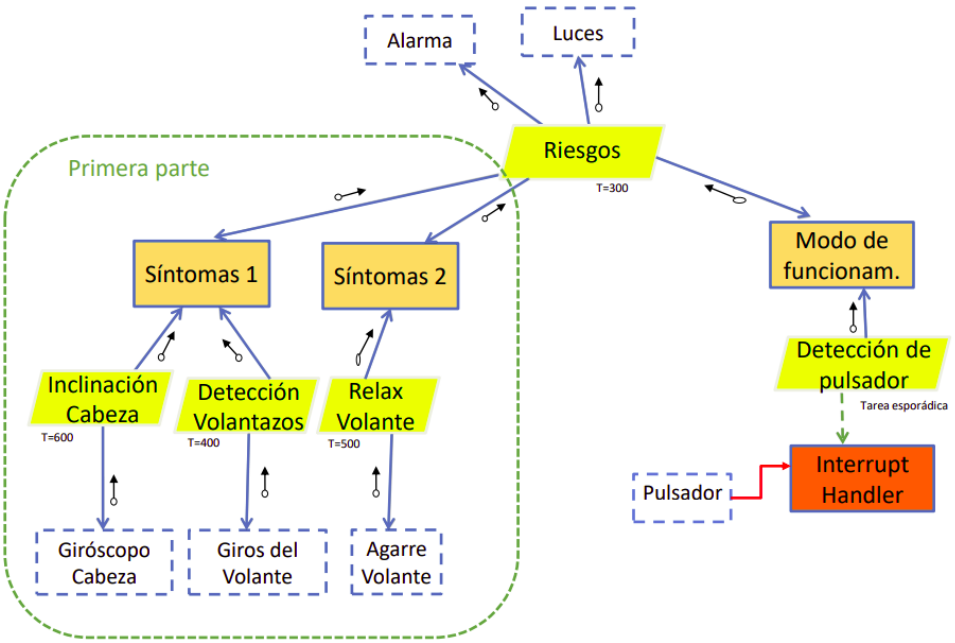

Figura 4: Modelado del nodo 2 junto con sus tareas, objetos protegidos, sensores y actuadores.

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 5 y 7. El objeto protegido `SPEED` sigue la definición estándar del resto de objetos protegidos y viene definido en los códigos 9 (cabeceras) y 12 (cuerpo).

Por otra parte, el envío de datos mediante el CANBus se realiza mediante la librería `can`, definida en los códigos 6 y 8.

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 ⩽ .2 * secure_dist)
148       intensity = 4;
149     else if (distance ⩽ .3 * secure_dist)
150       intensity = 3;
151     else if (distance ⩽ .4 * secure_dist)
152       intensity = 2;
153     else if (distance ⩽ .5 * 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 10 y 13) 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 11 y 14); se computa la distancia de seguridad y se calcula la intensidad de la frenada según unos porcentajes establecidos (líneas 140 – 156); si el valor de la intensidad de la frenada ha cambiado, se actualiza el objeto protegido y se notifica a la tarea esporádica que puede continuar su ejecución (líneas 158 – 161); finalmente, se envía el valor de la nueva distancia por el CANBus (línea 162) y se programa la siguiente ejecución 300 ms después del instante de activación (línea 163).

La tarea esporádica `Freno` viene definida por el código 3:

```
167 /**
168  * @brief Tarea esporádica que es activada por #distanceTask cuando la
       ↪ intensidad
169  *        de la frenada cambia. Además, se limita la activación de la
       ↪ tarea a, como
170  *        mucho, 150 ms de periodo para evitar cambios bruscos en la
       ↪ intensidad
171  *        de la frenada y cómo afecta a la comodidad de los pasajeros.
172  *
173  * @param args lista de posibles argumentos a usar. Vacía por defecto.
174  */
175 void brake_task(const void *args) {
176   int intensity;
177   uint32_t wake_time = osKernelSysTick();
178   const uint32_t T_BRAKE_TASK = 150U;
179   while (true) {
180     BRAKE_wait_event();
181     intensity = BRAKE_intensity_get();
182
183     switch (intensity) {
184     case 0:
185       HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_RESET);
186       HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_RESET);
187       HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_RESET);
188       HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_RESET);
189       break;
190
191     case 1:
192       HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_SET);
193       HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_RESET);
194       HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_RESET);
195       HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_RESET);
196       break;
197
198     case 2:
199       HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_SET);
200       HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_SET);
201       HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_RESET);
202       HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_RESET);
```

```
203       break;
204
205     case 3:
206       HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_SET);
207       HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_SET);
208       HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_SET);
209       HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_RESET);
210       break;
211         case 4:
212       HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_SET);
213       HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_SET);
214       HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_SET);
215       HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_SET);
216       break;
217
218     default:
219       break;
220     }
221
222     osDelayUntil(&wake_time, T_BRAKE_TASK);
223   }
224 }
```

Listing 3: Tarea esporádica que controla la intensidad de la frenada.

Dicha tarea espera a que se le notifique que se ha de ejecutar (línea 180) y después accede al objeto protegido que contiene la intensidad de la frenada (códigos 11 y 14); a continuación, según la intensidad de la frenada, enciende o apaga diversos LEDs en la placa a modo de indicativo visual de que se está frenando (líneas 183 – 220). Finalmente, para evitar que la tarea se pueda activar con una baja periodicidad se esperan al menos 150 ms desde el instante de activación.

Finalmente, la tarea de gestión de las `Luces de cruce` viene definida por el código 4:

```
226 /**
227  * @brief Tarea periódica (1 s) que se encarga de detectar cambios en
228  *        en entorno para activar las luces de cruce en condiciones
229  *        de poca visibilidad.
230  *
231  * @param argument lista de posibles argumentos a usar. Vacía por defecto
       ↪ .
232  */
233 void lucesCruce(void const *argument) {
234   HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_RESET);
235   int luminosity;
236   uint32_t wake_time = osKernelSysTick();
237   while(true) {
238     ADC_ChannelConfTypeDef sConfig = {0};
239     sConfig.Channel = ADC_CHANNEL_1;
240     sConfig.Rank = 1;
241     sConfig.SamplingTime = ADC_SAMPLETIME_28CYCLES;
242     HAL_ADC_ConfigChannel(&hadc1, &sConfig);
243     HAL_ADC_Start(&hadc1);
244     if (HAL_ADC_PollForConversion(&hadc1, 5) == HAL_OK) {
245       luminosity = map(HAL_ADC_GetValue(&hadc1), 0, 255, 0, 200);
246       if (luminosity < 100) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, 0);
247       else HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, 1);
```

```
248     }
249     osDelayUntil(&wake_time, T_TAREALUCESCRUCE);
250   }
251 }
```

Listing 4: Tarea periódica que controla las luces de cruce.

Dicha tarea lee desde el ADC (canal 1) el valor recibido por el LDR y, tras comprobar su luminosidad con el rango establecido enciende o apaga las luces de cruce (líneas 245 – 247). Finalmente, se programa la siguiente ejecución 1 s después de la activación. Esta tarea no accede a ningún objeto protegido.

2.2. Nodo 2

3. Diseño final

4. Aclaraciones

- En los códigos 1 y 4, 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.
- En diversos códigos (como 2 o 3) se utilizan eventos para la sincronización de tareas entre sí. Los eventos aparecen en la documentación estándar de FreeRTOS y constituyen un mecanismo muy sencillo y eficiente que respeta el tiempo real para bloquear y desbloquear tareas sin necesidad de programar la lógica subyacente. Un evento, en esencia, se conforma de $1 \ldots n$ procesos que esperan y, en principio, un único proceso k que "produce" el evento. En ese instante, aquellas tareas que estaban esperando al evento se desbloquean y prosiguen con su ejecución; mientras tanto, el proceso k reiniciaría el evento de forma que nuevas tareas pueden esperar a que se produzca.

  De esta manera, una tarea esporádica estaría esperando a que un evento se produzca y existiría una tarea periódica activadora la cual indicaría mediante dicho evento a la tarea esporádica que se tiene que ejecutar.

5. Glosario

A. Código fuente común

A.1. Cabeceras de código

```
1 /*
2  * Copyright © 2021 - present | utils.h 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
```

```
 * (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))

// Iterates through an array
#define foreach(idxtype, item, array) \
    idxtype* item; \
    size_t size = arrsize(array); \
    for (item = array; item < (array + size); ++item)

/**
 * @brief Custom datatype representing the union of
 *        a float value and its representation as a
 *        array of four bytes. Useful when converting
 *        from float to bytes and viceversa.
 */
typedef union float_u {
    float float_var;
    uint8_t bytes_repr[4];
} FloatU_t;


int map(int, int, int, int, int);

void f2b(float, uint8_t*);
float b2f(uint8_t*);

#endif /* UTILS_H */
```

Listing 5: Cabecera con funciones de utilidad.

```
/*
 * Copyright © 2021 - present | can.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 - can.h.
 */
#ifndef CAN_H
#define CAN_H
#include <FreeRTOSConfig.h>
#include <stdint.h>
#include <stm32f4xx_hal.h>
#ifndef CAN1
#define CAN1
#endif

// Standard TX/RX ID 1
extern const uint32_t STD_ID1;

// Standard TX/RX ID 2
extern const uint32_t STD_ID2;

// High filter ID
extern const uint32_t HFILTER_ID;

#ifdef NODE_2
// High filter mask for node 2 only
extern const uint32_t HFILTER_MASK;
#endif

void CAN_init(void);

void CAN_sendi(uint8_t);

void CAN_sendf(float);

uint8_t CAN_recv(void);

float CAN_recvf(void);

void CAN_Handle_IRQ(void);

#endif /* CAN_H */
```

Listing 6: Cabecera de la librería CANBus.

A.2. Cuerpo del código

```
/*
 * Copyright © 2021 - present | utils.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 10/04/21 - utils.c.
 */
#include "utils.h"

/**
 * @brief Maps a given value in between a given proportional range.
 *
 * @param x         the value to map.
 * @param in_min    the minimum input value to map.
 * @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);
```

```
67
68      return u.float_var;
69  }
```

Listing 7: Cuerpo de las funciones de utilidad.

```
1  /*
2   * Copyright © 2021 - present | can.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 - can.c.
18  */
19 #include "can.h"
20 #include <stm32f4xx_hal.h>
21 #include <FreeRTOS.h>
22 #include <FreeRTOSConfig.h>
23 #include <task.h>
24 #include <stdint.h>
25 #include "utils.h"
26 #ifdef NODE_2
27 #include "node1.h"
28 #endif
29
30 const uint32_t STD_ID1 = 0x6FA;
31 const uint32_t STD_ID2 = 0x6FB;
32 const uint32_t HFILTER_ID = 0x6FF << 5;
33
34 #ifdef NODE_2
35 const uint32_t HFILTER_MASK = 0x7F0 << 5;
36 #endif
37
38 static volatile CAN_HandleTypeDef hcan1;
39 #ifndef NODE_2
40 static volatile CAN_TxHeaderTypeDef tx_header;
41 static volatile CAN_TxHeaderTypeDef tx_header2;
42 #else
43 static volatile CAN_RxHeaderTypeDef rx_header;
44 #endif
45 static volatile uint32_t tx_mailbox;
46
47 static volatile uint8_t byte_sent = 0;
48 static volatile uint8_t byte_recv = 0;
49 static volatile float float_recv = .0F;
50
51 static volatile CAN_FilterTypeDef filter_config;
52
```

```c
/**
  * @brief CAN1 Initialization Function - extracted from main.
  */
static void MX_CAN1_Init(void) {
    hcan1.Instance = CAN1;
    hcan1.Init.Prescaler = 21U;
    hcan1.Init.Mode = CAN_MODE_NORMAL;
    hcan1.Init.SyncJumpWidth = CAN_SJW_1TQ;
    hcan1.Init.TimeSeg1 = CAN_BS1_12TQ;
    hcan1.Init.TimeSeg2 = CAN_BS2_4TQ;
    hcan1.Init.TimeTriggeredMode = DISABLE;
    hcan1.Init.AutoBusOff = DISABLE;
    hcan1.Init.AutoWakeUp = DISABLE;
    hcan1.Init.AutoRetransmission = DISABLE;
    hcan1.Init.ReceiveFifoLocked = DISABLE;
    hcan1.Init.TransmitFifoPriority = DISABLE;

    configASSERT(HAL_CAN_Init(&hcan1) == HAL_OK);
}

/**
 * @brief Initializes CANBus communications. This function
 * must be called early during initialization at main() as
 * the CAN functions won't work (and will block forever)
 * if called.
 */
void CAN_init(void) {
    MX_CAN1_Init();
#ifndef NODE_2
    // Message size of 1 byte
    tx_header.DLC = 1U;
    // Identifier to standard
    tx_header.IDE = CAN_ID_STD;
    // Data type to remote transmission
    tx_header.RTR = CAN_RTR_DATA;
    // Standard identifier
    tx_header.StdId = STD_ID1;

    // Message size of 4 bytes (float)
    tx_header2.DLC = 4U;
    // Identifier to standard
    tx_header2.IDE = CAN_ID_STD;
    // Data type to remote transmission
    tx_header2.RTR = CAN_RTR_DATA;
    // Standard identifier
    tx_header2.StdId = STD_ID2;
#endif

    // Filter one (stack light blink)
    filter_config.FilterFIFOAssignment = CAN_FILTER_FIFO0;
    // ID we're looking for
    filter_config.FilterIdHigh = HFILTER_ID;
    filter_config.FilterIdLow = 0U;

#ifndef NODE_2
    filter_config.FilterMaskIdHigh = 0U;
#else
    filter_config.FilterMaskIdHigh = HFILTER_MASK;
```

```
111     filter_config.FilterMode = CAN_FILTERMODE_IDMASK;
112 #endif
113     filter_config.FilterMaskIdLow = 0U;
114 
115     filter_config.FilterScale = CAN_FILTERSCALE_32BIT;
116     filter_config.FilterActivation = ENABLE;
117 
118     // Setup CAN filter
119     HAL_CAN_ConfigFilter(&hcan1, &filter_config);
120     HAL_CAN_Start(&hcan1);
121     HAL_CAN_ActivateNotification(&hcan1, CAN_IT_RX_FIFO0_MSG_PENDING);
122 }
123 
124 /**
125  * @brief Sends a byte through the CANBus using the #STD_ID1
126  *        message identifier.
127  *
128  *        Note: this method will do nothing if NODE_2 is defined.
129  *
130  * @param b the byte to send.
131  */
132 void CAN_sendi(uint8_t b) {
133 #ifndef NODE_2
134     byte_sent = b;
135     HAL_CAN_AddTxMessage(&hcan1, &tx_header, &byte_sent, &tx_mailbox);
136 #endif
137 }
138 
139 /**
140  * @brief Sends a float through the CANBus using the #STD_ID2
141  *        message identifier.
142  *
143  *        Note: this method will do nothing if NODE_2 is defined.
144  *
145  * @param value the float value to send.
146  * @see   f2b(float, uint8_t*)
147  */
148 void CAN_sendf(float value) {
149 #ifndef NODE_2
150     uint8_t bytes[4];
151     f2b(value, bytes);
152     HAL_CAN_AddTxMessage(&hcan1, &tx_header2, &bytes[0], &tx_mailbox);
153 #endif
154 }
155 
156 /**
157  * @brief When a message arrives, the received byte (if any) is stored in
158  *        a private variable. Use this method to recover its value.
159  *
160  *        Notice that this value will only be updated when the received
161  *        message ID matches the #STD_ID1.
162  *
163  * @return uint8_t the stored byte.
164  */
165 uint8_t CAN_recv(void) {
166     return byte_recv;
167 }
168 
```

```
169 /**
170  * @brief When a message arrives, the received float (if any) is stored
      ↪ in
171  *        a private variable. Use this method to recover its value.
172  *
173  *        Notice that this value will only be updated when the received
174  *        message ID matches the #STD_ID2.
175  *
176  * @return float - the stored float.
177  */
178 float CAN_recvf(void) {
179     return float_recv;
180 }
181 
182 /**
183  * @brief This method must be called if the board wants to receive
184  *        CANBus messages during the CANBus interruption routine.
185  *
186  *        By filtering the ID, identifies whether the received array
187  *        is either a single byte or a float value.
188  *
189  *        In addition, this method sets a flag at the respective
190  *        protected objects indicating that a new message is received
191  *        and is ready to be used (notice that this method is called
192  *        from an IRQ, so the processing must be as efficient as
193  *        possible. In this function, setting a flag is easy and
194  *        not blocking - at least not as much as changing a lock/
      ↪ semaphore).
195  *
196  *        The affected protected objects are the ones that store the
197  *        SPEED and the DISTANCE.
198  *
199  * @see node1.h
200  */
201 void CAN_Handle_IRQ(void) {
202     HAL_CAN_IRQHandler(&hcan1);
203 #ifdef NODE_2
204     uint8_t bytes[4];
205     HAL_CAN_GetRxMessage(&hcan1, CAN_RX_FIFO0, &rx_header, &bytes);
206     if (rx_header.StdId == STD_ID1) {
207         byte_recv = bytes[0];
208         SPEED_set_recv();
209     }
210     if (rx_header.StdId == STD_ID2) {
211         float_recv = b2f(&bytes[0]);
212         DISTANCE_set_recv();
213     }
214 #endif
215 }
```

Listing 8: Cuerpo de la librería CANBus.

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 9: 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>
```

```
22
23 uint32_t USS_read_distance(void);
24
25 #endif
```

Listing 10: Cabecera del controlador de ultrasonidos.

```
1  /*
2   * Copyright © 2021 - present | distance.h 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 13/03/21 - distance.h.
18  */
19 #ifndef DISTANCE_H
20 #define DISTANCE_H
21
22 void DISTANCE_init(void);
23 void DISTANCE_set(float);
24 float DISTANCE_get(void);
25 void DISTANCE_delete(void);
26 void BRAKE_intensity_set(int);
27 int BRAKE_intensity_get(void);
28
29 #endif /* DISTANCE_H */
```

Listing 11: Cabecera del objeto protegido `distance`.

B.2. Cuerpo del código

```
1  /*
2   * Copyright © 2021 - present | speed.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/.
```

```
 *
 * 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 12: 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 13: Cuerpo del controlador de ultrasonidos.

```
/*
 * Copyright © 2021 - present | distance.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
```

```
15  * along with this program.  If not, see https://www.gnu.org/licenses/.
16  *
17  * Created by Javinator9889 on 13/03/21 - distance.c.
18  */
19 #include "distance.h"
20 #include <lock.h>
21 #include <semphr.h>
22 #include <FreeRTOS.h>
23 #include <FreeRTOSConfig.h>
24 #include <task.h>
25
26 // Private variable for locking distance instance
27 static Lock_t INSTANCE_sem = NULL;
28 // Private variable that stores the distance itself.
29 static volatile float DISTANCE_distance = 0;
30 // Private variable that stores the brake intensity itself.
31 static volatile int BRAKE_intensity = 0;
32
33 /**
34  * @brief Initializes the distance protected object alongside
35  *        the brake intensity one (both share the same lock).
36  *
37  *        This method must be called during the early boot as,
38  *        until then, any call to any method will fail and block
39  *        forever.
40  */
41 void DISTANCE_init(void) {
42     INSTANCE_sem = LOCK_create(NULL);
43 }
44
45 /**
46  * @brief Safely updates the stored distance value.
47  *
48  * @param distance the new distance.
49  */
50 void DISTANCE_set(float distance) {
51     LOCK_acquire(INSTANCE_sem);
52     DISTANCE_distance = distance;
53     LOCK_release(INSTANCE_sem);
54 }
55
56 /**
57  * @brief Safely obtains the stored distance value.
58  *
59  * @return float - the stored distance.
60  */
61 float DISTANCE_get(void) {
62     float distance = -1;
63     LOCK_acquire(INSTANCE_sem);
64     distance = DISTANCE_distance;
65     LOCK_release(INSTANCE_sem);
66     return distance;
67 }
68
69 /**
70  * @brief Deletes all stored objects and resets the
71  *        distance value. After this method call,
72  *        all subsequent calls will fail until
```

```
 *        #DISTANCE_init is called again.
 *
 */
void DISTANCE_delete(void) {
    LOCK_destroy(INSTANCE_sem);
    INSTANCE_sem = NULL;
    DISTANCE_distance = 0;
    BRAKE_intensity = 0;
}

/**
 * @brief Safely updates the brake intensity value.
 *
 * @param intensity the new intensity.
 */
void BRAKE_intensity_set(int intensity) {
    LOCK_acquire(INSTANCE_sem);
    BRAKE_intensity = intensity;
    LOCK_release(INSTANCE_sem);
}

/**
 * @brief Safely obtains the stored brake intensity value.
 *
 * @return int - the stored intensity value.
 */
int BRAKE_intensity_get(void) {
    int intensity = -1;
    LOCK_acquire(INSTANCE_sem);
    intensity = BRAKE_intensity;
    LOCK_release(INSTANCE_sem);
    return intensity;
}
```

Listing 14: Cuerpo del objeto protegido `distance`.

C. Código fuente nodo 2