

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	11
3. Diseño final	12
4. Aclaraciones	12
5. Glosario	12
A. Código fuente común	12
A.1. Cabeceras de código	12
A.2. Cuerpo del código	15
B. Código fuente nodo 1	22
B.1. Cabeceras de código	22
B.2. Cuerpo del código	24
C. Código fuente nodo 2	29
C.1. Cabeceras de código	29
C.2. Cuerpo del código	30

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 $[0, 200] \text{ km/h}$. Se usará para comprobar que se cumple la distancia de seguridad.
- Sensor de distancia, capaz de realizar lecturas en el rango $[5, 200] \text{ m}$ 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 objetos 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 distribuido entre ambos y viene representado por la figura 1:

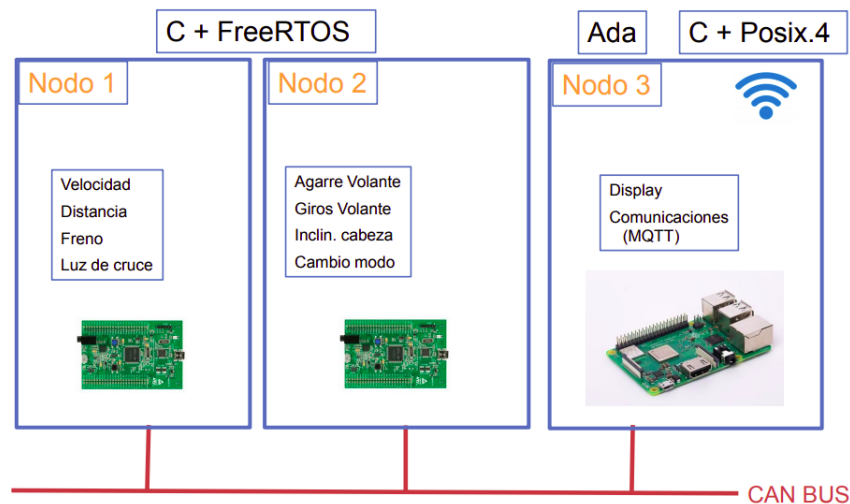


Figura 1: Modelo completo del sistema a implementar. Las tareas van distribuidas entre los dos nodos principales y se comunican entre ellos mediante CANBus.

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:

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

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:

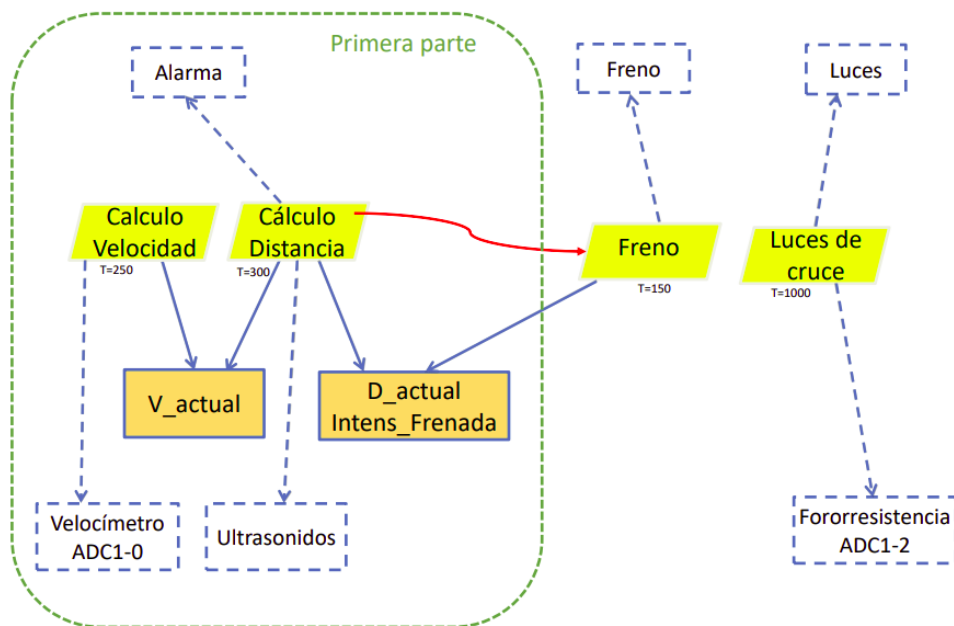


Figura 2: Modelado del nodo 1 junto con sus tareas, objetos protegidos, sensores y actuadores.

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 $X > 20^\circ | Y > 20^\circ$, 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 riesgos 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:

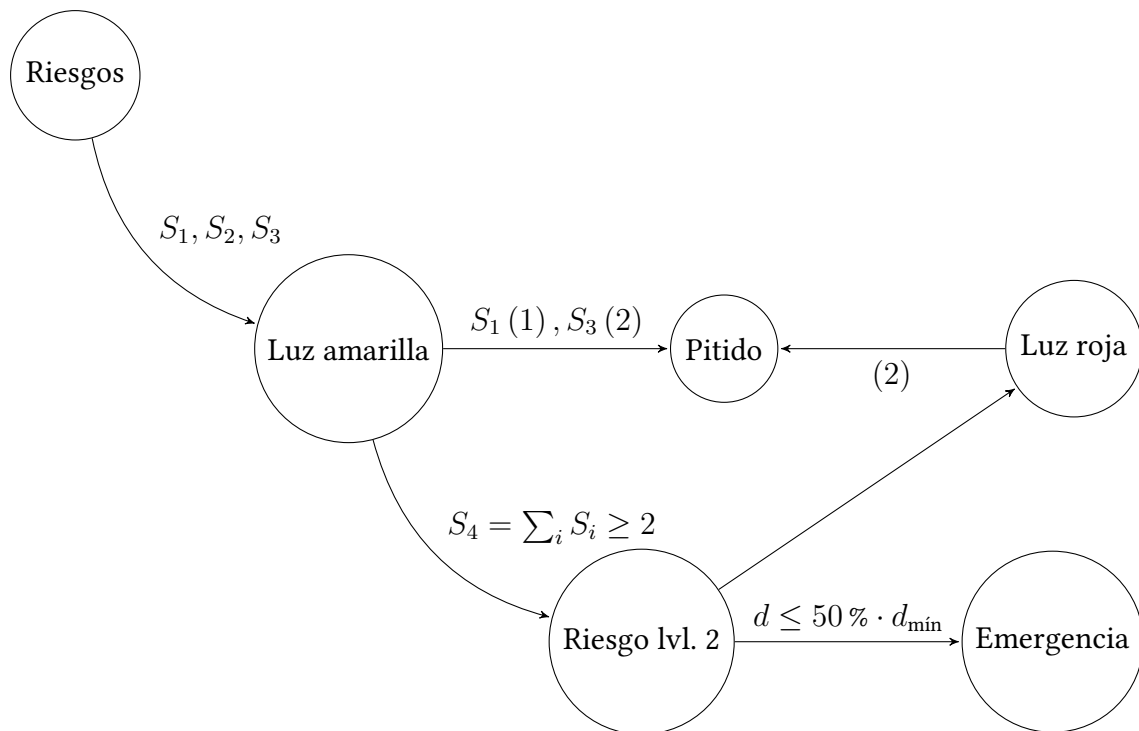


Figura 3: Diagrama que modela la interpretación de los riesgos, descritos en la enumeración anterior (S_i). La intensidad del pitido va acompañada entre paréntesis del síntoma que lo activa (por ejemplo, S_1 (1) indica una intensidad de pitido nivel 1) o en solitario, si es consecuencia de acciones en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

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

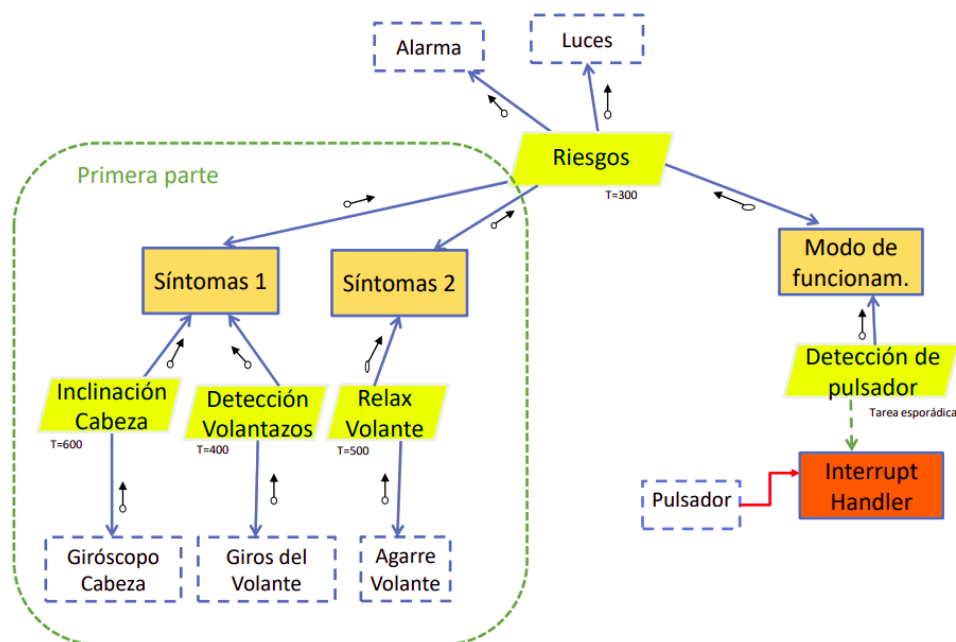


Figura 4: Modelado del nodo 2 junto con sus tareas, objetos protegidos, sensores y actuadores.

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  */
98 void acelerador(const void *argument) {
99     int speed;
100     uint32_t wake_time = osKernelSysTick();
101     while(true) {
102         ADC_ChannelConfTypeDef sConfig = {0};
103         sConfig.Channel = ADC_CHANNEL_0;
104         sConfig.Rank = 1;
105         sConfig.SamplingTime = ADC_SAMPLETIME_28CYCLES;
106         HAL_ADC_ConfigChannel(&hadc1, &sConfig);
107         HAL_ADC_Start(&hadc1);
108         if (HAL_ADC_PollForConversion(&hadc1, 5) == HAL_OK) {
109             speed = map(HAL_ADC_GetValue(&hadc1), 0, 255, 0, 200);
110             SPEED_set(speed);
111             CAN_sendi(speed);
112         }
113         osDelayUntil(&wake_time, T_TAREAVELOCIDAD);
114     }
115 }

```

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 8 y 11. El objeto protegido SPEED sigue la definición estándar del resto de objetos protegidos y viene definido en los códigos 14 (cabeceras) y 18 (cuerpo).

Por otra parte, el envío de datos mediante el CANBus se realiza mediante la librería can, definida en los códigos 9 y 12.

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,
120  *        ↪ además,
121  *        se computa el valor de la intensidad de frenada para activar (o
122  *        ↪ no)
123  *        a la tarea esporádica #brake_task.
124  *
125  * @param args lista de posibles argumentos a usar. Vacía por defecto.
126  */
127 void distanceTask(const void *args) {
128     const uint16_t T_DISTANCE_TASK = 300U;
129     uint32_t wake_time = osKernelSysTick();
130     float distance;
131     float speed;
132     float secure_dist;
133     int old_intensity = 0;
134     int intensity = 0;
135     HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_RESET);
136     while (1) {
137         distance = (float) USS_read_distance() * 0.00171821F;
138         if (distance == 500000)
139             distance = 1;
140         DISTANCE_set(distance);
141
142         speed = SPEED_get();
143         secure_dist = (float) pow((speed / 10), 2);
144
145         if (distance < secure_dist) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9,
146             ↪ GPIO_PIN_SET);
147         else HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_RESET);
148
149         old_intensity = BRAKE_intensity_get();
150         if (distance ≤ .2 * secure_dist)
151             intensity = 4;
152         else if (distance ≤ .3 * secure_dist)
153             intensity = 3;
154         else if (distance ≤ .4 * secure_dist)
155             intensity = 2;
156         else if (distance ≤ .5 * secure_dist)
157             intensity = 1;
158         else
159             intensity = 0;
160
161         if (intensity ≠ old_intensity) {
162             BRAKE_intensity_set(intensity);
163             BRAKE_set_event();
164         }
165     }
166 }

```

```

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 15 y 19) 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 16 y 20); 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) (códigos 17 y 21); 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
169  *    ↪ intensidad
170  *    ↪ de la frenada cambia. Además, se limita la activación de la
171  *    ↪ tarea a, como
172  *    ↪ mucho, 150 ms de periodo para evitar cambios bruscos en la
173  *    ↪ intensidad
174  *    ↪ de la frenada y cómo afecta a la comodidad de los pasajeros.
175  *
176  * @param args lista de posibles argumentos a usar. Vacía por defecto.
177  */
178 void brake_task(const void *args) {
179     int intensity;
180     uint32_t wake_time = osKernelSysTick();
181     const uint32_t T_BRAKE_TASK = 150U;
182     while (true) {
183         BRAKE_wait_event();
184         intensity = BRAKE_intensity_get();
185
186         switch (intensity) {
187             case 0:
188                 HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_RESET);
189                 HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_RESET);
190                 HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_RESET);
191                 HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_RESET);
192                 break;
193
194             case 1:
195                 HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_SET);
196                 HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_RESET);
197                 HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_RESET);
198                 HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_RESET);
199                 break;
200
201             case 2:
202                 HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_SET);
203                 HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_SET);
204                 HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_RESET);
205                 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 16 y 20); 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  */
234 void lucesCruce(void const *argument) {
235     HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_RESET);
236     int luminosity;
237     uint32_t wake_time = osKernelSysTick();
238     while(true) {
239         ADC_ChannelConfTypeDef sConfig = {0};
240         sConfig.Channel = ADC_CHANNEL_1;
241         sConfig.Rank = 1;
242         sConfig.SamplingTime = ADC_SAMPLETIME_28CYCLES;
243         HAL_ADC_ConfigChannel(&hadc1, &sConfig);
244         HAL_ADC_Start(&hadc1);
245         if (HAL_ADC_PollForConversion(&hadc1, 5) == HAL_OK) {
246             luminosity = map(HAL_ADC_GetValue(&hadc1), 0, 255, 0, 200);
247             if (luminosity < 100) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, 0);
248             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.

Cabe destacar que los distintos objetos protegidos que se declaran y usan a lo largo del código se basan en la librería `lock`, definida en los códigos 10 y 13. Dicha librería requiere que los objetos protegidos sean inicializados antes de realizar ninguna operación con ellos. Por ende, en el bloque `main` es necesario dedicar unas líneas para iniciar cada una de los objetos protegidos que se quieren usar (código 5):

```

253 /**
254  * @brief The application entry point.
255  * @retval int
256  */
257 int main(void) {
258
259     /* MCU Configuration
260      ↪ _____ */
261
262     /* Reset of all peripherals, Initializes the Flash interface and the
263      ↪ SysTick. */
264     HAL_Init();
265
266     /* Configure the system clock */
267     SystemClock_Config();
268
269     /* Initialize all configured peripherals */
270     MX_GPIO_Init();
271     MX_ADC1_Init();
272     MX_SPI1_Init();
273     CAN_init();
274     SPEED_init();
275     DISTANCE_init();
276     BRAKE_init();
277
278     /* Create the thread(s) */
279     xTaskCreate((TaskFunction_t) acelerador,
280               "lectura potenciómetro",
281               configMINIMAL_STACK_SIZE,
282               NULL, PR_TAREA2, NULL);
283     xTaskCreate((TaskFunction_t) lucesCruce,
284               "lectura luces",
285               configMINIMAL_STACK_SIZE,
286               NULL, PR_TAREA2, NULL);
287     xTaskCreate((TaskFunction_t) distanceTask,
288               "lectura distancia",
289               configMINIMAL_STACK_SIZE,
290               NULL, PR_DISTANCIA, NULL);

```

```

289 | xTaskCreate((TaskFunction_t) brake_task,
290 |           "tarea freno",
291 |           configMINIMAL_STACK_SIZE,
292 |           NULL, PR_BRAKE, NULL);
293 |
294 | /* Start scheduler */
295 | vTaskStartScheduler();
296 | /* We should never get here as control is now taken by the scheduler */
297 |
298 | /* Infinite loop */
299 | while (1);
300 | }

```

Listing 5: Código del main.

En las líneas 271 – 274 se inicializan los objetos protegidos, dejándolos listos para su uso. El cuerpo viene definido en los códigos 12, 18, 20, 21.

2.2. Nodo 2

En el nodo 2 se han implementado también todas las tareas y requisitos pedidos en el documento de requisitos.

La tarea de actualización del modo de funcionamiento viene definida por el código 6:

```

110 | /**
111 |  * @brief Tarea esporádica que espera la detección del pulsador para
112 |  *        el cambio de modo.
113 |  *
114 |  * @param argument lista de posibles argumentos a usar. Vacía por defecto
115 |  *        ↪ .
116 |  */
117 | void deteccionPulsador(const void *argument) {
118 |     uint32_t wake_time = osKernelSysTick();
119 |     while(true) {
120 |         if (xSemaphoreTake(interrupcion, portMAX_DELAY) == pdTRUE) {
121 |             modo = ++modo % 3;
122 |             MODE_set(modo);
123 |         }
124 |     }

```

Listing 6: Tarea esporádica que controla el modo.

Dicha tarea espera a un semáforo binario que indica que se ha producido una interrupción (línea 119). Una vez desbloqueada, incrementa el modo hasta un máximo de '2' (línea 120) y finalmente actualiza el objeto protegido mode (códigos 22 y 23).

Por otra parte, como hace falta liberar el semáforo cuando se produce la interrupción del pulsador, es necesario definir un nuevo fragmento del código que habilite a la placa para esa tarea (código 7):

```

663 | /* Funcion para el tratamiento de interrupciones */
664 |
665 | void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)

```

```
666 {  
667     long yield = pdFALSE;  
668     // /* Prevent unused argument(s) compilation warning */  
669     UNUSED(GPIO_Pin);  
670     portYIELD_FROM_ISR(yield);  
671  
672     if (GPIO_Pin == GPIO_PIN_3) {  
673         xSemaphoreGiveFromISR(interrupcion, &yield);  
674     }  
675 }
```

Listing 7: Rutina de tratamiento de interrupciones.

Cuando se activa el GPIO3 se “devolverá” el semáforo, lo que permitirá que la tarea esporádica pueda adquirirlo y realizar su ejecución. Sin embargo, cuando lo intente adquirir de nuevo se bloqueará y hasta que no se repita este proceso quedará a la espera de que se libere el semáforo.

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 $1 \dots n$ 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
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.h.
18 */
19 #ifndef UTILS_H
20 #define UTILS_H
21 #include <stdint.h>
22
23 // Gets the size of an array
24 #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 8: Cabecera con funciones de utilidad.

```

1  /*
2  * Copyright © 2021 - present | can.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 10/04/21 - can.h.
18 | */
19 | #ifndef CAN_H
20 | #define CAN_H
21 | #include <FreeRTOSConfig.h>
22 | #include <stdint.h>
23 | #include <stm32f4xx_hal.h>
24 | #ifndef CAN1
25 | #define CAN1
26 | #endif
27 |
28 | // Standard TX/RX ID 1
29 | extern const uint32_t STD_ID1;
30 |
31 | // Standard TX/RX ID 2
32 | extern const uint32_t STD_ID2;
33 |
34 | // High filter ID
35 | extern const uint32_t HFILTER_ID;
36 |
37 | #ifdef NODE_2
38 | // High filter mask for node 2 only
39 | extern const uint32_t HFILTER_MASK;
40 | #endif
41 |
42 | void CAN_init(void);
43 |
44 | void CAN_sendi(uint8_t);
45 |
46 | void CAN_sendf(float);
47 |
48 | uint8_t CAN_recv(void);
49 |
50 | float CAN_recvf(void);
51 |
52 | void CAN_Handle_IRQ(void);
53 |
54 | #endif /* CAN_H */
```

Listing 9: Cabecera de la librería CANBus.

```
1 | /*
2 | * Copyright © 2021 - present | lock_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 05/03/21 - lock_h.
18  */
19
20 #ifndef LOCK_H
21 #define LOCK_H
22 #include <FreeRTOS.h>
23 #include <stddef.h>
24 #include <semphr.h>
25
26 // Custom data type used for identifying LOCK created locks.
27 typedef SemaphoreHandle_t Lock_t;
28
29 Lock_t LOCK_create(StaticSemaphore_t*);
30 void LOCK_destroy(Lock_t);
31 long LOCK_acquire(Lock_t);
32 void LOCK_release(Lock_t);
33
34 #endif /* LOCK_H */
```

Listing 10: Cabecera de la librería lock.

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.
27 * @param out_min    the minimum output value to produce.
```

```

28 * @param out_max the maximum output value to produce.
29 * @return int - the 'x' value mapped in between [out_min, out_max].
30 */
31 int map(int x, int in_min, int in_max, int out_min, int out_max) {
32     return (int)((x - in_min) * (out_max - out_min) / (in_max - in_min) +
33         ↪ out_min);
34 }
35 /**
36 * @brief With the given float value, produces the equivalent 4 bytes
37 * representing that value.
38 *
39 * Notice that this function relies on that a float is 4 bytes
40 * in memory. Higher (or lower) values will require this method
41 * to be overwritten.
42 *
43 * @param value the input float to convert.
44 * @param bytes the output bytes array (4) to produce.
45 */
46 void f2b(float value, uint8_t bytes[4]) {
47     FloatU_t u;
48
49     u.float_var = value;
50     memcpy(bytes, u.bytes_repr, 4);
51 }
52
53 /**
54 * @brief With the given bytes array, produces the equivalent float value
55 * represented by that 4 bytes.
56 *
57 * Notice that this function relies on that a float is 4 bytes
58 * in memory. Higher (or lower) values will require this method
59 * to be overwritten.
60 *
61 * @param bytes the input bytes array (4) to read.
62 * @return float - the converted float data from bytes.
63 */
64 float b2f(uint8_t bytes[4]) {
65     FloatU_t u;
66     memcpy(u.bytes_repr, bytes, 4);
67
68     return u.float_var;
69 }

```

Listing 11: 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_rcv = 0;
49  static volatile float float_rcv = .0F;
50
51  static volatile CAN_FilterTypeDef filter_config;
52
53  /**
54   * @brief CAN1 Initialization Function - extracted from main.
55   */
56  static void MX_CAN1_Init(void) {
57      hcan1.Instance = CAN1;
58      hcan1.Init.Prescaler = 21U;
59      hcan1.Init.Mode = CAN_MODE_NORMAL;
60      hcan1.Init.SyncJumpWidth = CAN_SJW_1TQ;
61      hcan1.Init.TimeSeg1 = CAN_BS1_12TQ;
62      hcan1.Init.TimeSeg2 = CAN_BS2_4TQ;
63      hcan1.Init.TimeTriggeredMode = DISABLE;
64      hcan1.Init.AutoBusOff = DISABLE;
65      hcan1.Init.AutoWakeUp = DISABLE;
66      hcan1.Init.AutoRetransmission = DISABLE;
67      hcan1.Init.ReceiveFifoLocked = DISABLE;
68      hcan1.Init.TransmitFifoPriority = DISABLE;
69
70      configASSERT(HAL_CAN_Init(&hcan1) == HAL_OK);
```

```
71 | }
72 |
73 | /**
74 |  * @brief Initializes CANBus communications. This function
75 |  * must be called early during initialization at main() as
76 |  * the CAN functions won't work (and will block forever)
77 |  * if called.
78 |  */
79 | void CAN_init(void) {
80 |     MX_CAN1_Init();
81 | #ifndef NODE_2
82 |     // Message size of 1 byte
83 |     tx_header.DLC = 1U;
84 |     // Identifier to standard
85 |     tx_header.IDE = CAN_ID_STD;
86 |     // Data type to remote transmission
87 |     tx_header.RTR = CAN_RTR_DATA;
88 |     // Standard identifier
89 |     tx_header.StdId = STD_ID1;
90 |
91 |     // Message size of 4 bytes (float)
92 |     tx_header2.DLC = 4U;
93 |     // Identifier to standard
94 |     tx_header2.IDE = CAN_ID_STD;
95 |     // Data type to remote transmission
96 |     tx_header2.RTR = CAN_RTR_DATA;
97 |     // Standard identifier
98 |     tx_header2.StdId = STD_ID2;
99 | #endif
100 |
101 |     // Filter one (stack light blink)
102 |     filter_config.FilterFIFOAssignment = CAN_FILTER_FIFO0;
103 |     // ID we're looking for
104 |     filter_config.FilterIdHigh = HFILTER_ID;
105 |     filter_config.FilterIdLow = 0U;
106 |
107 | #ifndef NODE_2
108 |     filter_config.FilterMaskIdHigh = 0U;
109 | #else
110 |     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_rcv(void) {
166 |     return byte_rcv;
167 | }
168 |
169 | /**
170 | * @brief When a message arrives, the received float (if any) is stored
171 | *       ↪ in
172 | *       a private variable. Use this method to recover its value.
173 | *
174 | *       Notice that this value will only be updated when the received
175 | *       message ID matches the #STD_ID2.
176 | *
177 | * @return float - the stored float.
178 | */
179 | float CAN_rcvf(void) {
180 |     return float_rcv;
181 | }
182 |
183 | /**
184 | * @brief This method must be called if the board wants to receive
185 | *       CANBus messages during the CANBus interruption routine.
```

```

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/
195 *      ↪ semaphore).
196 *
197 *      The affected protected objects are the ones that store the
198 *      SPEED and the DISTANCE.
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_rcv = bytes[0];
208         SPEED_set_rcv();
209     }
210     if (rx_header.StdId == STD_ID2) {
211         float_rcv = b2f(&bytes[0]);
212         DISTANCE_set_rcv();
213     }
214 #endif
215 }

```

Listing 12: Cuerpo de la librería CANBus.

```

1 /*
2  * Copyright © 2021 - present | lock_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 05/03/21 - lock_c.
18 */
19 #include "lock.h"
20 #include <FreeRTOS.h>
21 #include <FreeRTOSConfig.h>
22 #include <portmacro.h>
23 #include <projdefs.h>
24 #include <semphr.h>

```

```
25 #include "task.h"
26
27 /**
28  * @brief creates a new lock (mutex) using FreeRTOS API. In addition,
29  *       some health checks are performed in order to return a valid
30  *       lock or, in other case, to block the execution forever
31  *       (both configASSERT will block if the condition is not met).
32  *
33  *       In addition, the ability to create static mutexes is given
34  *       to the function if #mutexBuffer is not NULL and if
35  *       #configSUPPORT_STATIC_ALLOCATION equals 1. In other case,
36  *       a lock allocated in heap will be created.
37  *
38  * @param mutexBuffer pointer to the static semaphore memory region.
39  *       NULL if wanna create a heap-based semaphore.
40  * @return Lock_t - the created lock.
41  */
42 Lock_t LOCK_create(StaticSemaphore_t *mutexBuffer) {
43     SemaphoreHandle_t xSemaphore = NULL;
44     BaseType_t xReturned;
45
46     #if defined(configSUPPORT_STATIC_ALLOCATION) && (config
47     ↪ SUPPORT_STATIC_ALLOCATION == 1)
48     if (mutexBuffer != NULL) xSemaphore = xSemaphoreCreateMutexStatic(
49     ↪ mutexBuffer);
50     else
51     #endif
52     xSemaphore = xSemaphoreCreateMutex();
53
54     configASSERT(xSemaphore != NULL);
55     configASSERT(xSemaphoreGive(xSemaphore) != pdTRUE);
56
57     return (Lock_t) xSemaphore;
58 }
59
60 /**
61  * @brief destroys the given lock (release from memory). After called,
62  *       the memory will be empty and the lock cannot be used anymore.
63  *
64  * @param sem the lock to destroy.
65  */
66 inline void LOCK_destroy(Lock_t sem) {
67     vSemaphoreDelete((SemaphoreHandle_t) sem);
68 }
69
70 /**
71  * @brief tries to acquire the given lock, blocking forever if necessary.
72  *
73  * @param sem the lock to acquire.
74  * @return long - #pdTRUE if the semaphore was acquired. #pdFALSE
75  *       ↪ otherwise.
76  */
77 inline long LOCK_acquire(Lock_t sem) {
78     configASSERT(sem != NULL);
79     return xSemaphoreTake((SemaphoreHandle_t) sem, portMAX_DELAY);
80 }
```

```
80 * @brief tries to release the given lock, blocking forever if empty (
    ↪ lock is
81 *     NULL).
82 *
83 * @param sem the lock to release.
84 */
85 inline void LOCK_release(Lock_t sem) {
86     configASSERT(sem ≠ NULL);
87     xSemaphoreGive((SemaphoreHandle_t) sem);
88 }
```

Listing 13: Cuerpo de la librería lock.

B. Código fuente nodo 1

B.1. Cabeceras de código

```
1 /*
2  * Copyright © 2021 - present | speed.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 05/03/21 - speed.h.
18 */
19 #ifndef SPEED_H
20 #define SPEED_H
21 #include <FreeRTOS.h>
22 #include <stddef.h>
23 #include <semphr.h>
24
25 void SPEED_init(void);
26 void SPEED_set(int);
27 int SPEED_get(void);
28
29 #endif /* SPEED_H */
```

Listing 14: Cabecera del objeto protegido speed.

```
1 /*
2  * Copyright © 2021 - present | uss.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 26/03/21 - uss.h.
18 */
19 #ifndef USS_H
20 #define USS_H
21 #include <stdint.h>
22
23 uint32_t USS_read_distance(void);
24
25 #endif
```

Listing 15: 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 16: Cabecera del objeto protegido distance.

```
1 /*
2  * Copyright © 2021 - present | brake.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 26/03/21 - brake.h.
18 */
19 #ifndef BRAKE_H
20 #define BRAKE_H
21
22 // The BIT flag set used for identifying if the flag
23 // is set or not.
24 #define BIT_SET (0x02UL)
25
26 void BRAKE_init(void);
27 void BRAKE_wait_event(void);
28 void BRAKE_set_event(void);
29 void BRAKE_clr(void);
30
31 #endif /* BRAKE_H */
```

Listing 17: Cabecera del objeto protegido brake.

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/.
16 *
17 * Created by Javinator9889 on 05/03/21 - speed.c.
18 */
19 #include "speed.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 containing the speed lock.
27 | static Lock_t SPEED_sem = NULL;
28 | // Private variable containing the speed itself.
29 | static int SPEED_speed = 0;
30 |
31 | /**
32 |  * @brief Initializes the speed protected object.
33 |  *
34 |  *      This method must be called during the early boot as,
35 |  *      until then, any call to any method will fail and block
36 |  *      forever.
37 |  *
38 |  */
39 | void SPEED_init(void) {
40 |     SPEED_sem = LOCK_create(NULL);
41 | }
42 |
43 | /**
44 |  * @brief Safely updates the stored speed value.
45 |  *
46 |  * @param speed the new speed.
47 |  */
48 | void SPEED_set(int speed) {
49 |     LOCK_acquire(SPEED_sem);
50 |     SPEED_speed = speed;
51 |     LOCK_release(SPEED_sem);
52 | }
53 |
54 | /**
55 |  * @brief Safely obtains the stored speed value.
56 |  *
57 |  * @return int - the speed. If any error occurs, returns -1.
58 |  */
59 | int SPEED_get(void) {
60 |     int speed = -1;
61 |     LOCK_acquire(SPEED_sem);
62 |     speed = SPEED_speed;
63 |     LOCK_release(SPEED_sem);
64 |     return speed;
65 | }
```

Listing 18: Cuerpo del objeto protegido speed.

```
1 | /*
2 |  * Copyright © 2021 - present | uss.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 26/03/21 - uss.c.
18  */
19  #include "uss.h"
20  #include <FreeRTOS.h>
21  #include <FreeRTOSConfig.h>
22  #include <task.h>
23  #include <stm32f4xx_hal.h>
24  #include "dwt_stm32_delay.h"
25
26  /**
27   * @brief Reads the measured distance from the ultrasonic sensor.
28   *
29   * @return uint32_t - the measured distance, in meters.
30   */
31  uint32_t USS_read_distance(void) {
32      __IO uint8_t flag = 0;
33      __IO uint32_t disTime = 0;
34
35      HAL_GPIO_WritePin(GPIOD, GPIO_PIN_10, GPIO_PIN_SET);
36      DWT_Delay_us(10);
37      HAL_GPIO_WritePin(GPIOD, GPIO_PIN_10, GPIO_PIN_RESET);
38
39      while(flag == 0) {
40          while(HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_11) == GPIO_PIN_SET) {
41              disTime++;
42              flag = 1;
43          }
44      }
45      return disTime;
46  }
```

Listing 19: Cuerpo del controlador de ultrasonidos.

```
1  /*
2  * Copyright © 2021 - present | distance.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 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
73  *       #DISTANCE_init is called again.
74  *
75  */
76 void DISTANCE_delete(void) {
77     LOCK_destroy(INSTANCE_sem);
78     INSTANCE_sem = NULL;
79     DISTANCE_distance = 0;
80     BRAKE_intensity = 0;
```

```
81 | }
82 |
83 | /**
84 |  * @brief Safely updates the brake intensity value.
85 |  *
86 |  * @param intensity the new intensity.
87 |  */
88 | void BRAKE_intensity_set(int intensity) {
89 |     LOCK_acquire(INSTANCE_sem);
90 |     BRAKE_intensity = intensity;
91 |     LOCK_release(INSTANCE_sem);
92 | }
93 |
94 | /**
95 |  * @brief Safely obtains the stored brake intensity value.
96 |  *
97 |  * @return int - the stored intensity value.
98 |  */
99 | int BRAKE_intensity_get(void) {
100 |     int intensity = -1;
101 |     LOCK_acquire(INSTANCE_sem);
102 |     intensity = BRAKE_intensity;
103 |     LOCK_release(INSTANCE_sem);
104 |     return intensity;
105 | }
```

Listing 20: Cuerpo del objeto protegido distance.

```
1 | /*
2 |  * Copyright © 2021 - present | brake.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 13/03/21 - brake.c.
18 |  */
19 | #include "brake.h"
20 | #include <FreeRTOS.h>
21 | #include <FreeRTOSConfig.h>
22 | #include <task.h>
23 | #include <event_groups.h>
24 |
25 | // Private variable storing the BRAKE flag
26 | static EventGroupHandle_t BRAKE_event = NULL;
27 |
28 | /**
29 |  * @brief Initializes BRAKE protected object. This method must be called
30 |  * during the early boot of the application in order to be able
```

```

31  *      to use the object's methods.
32  *
33  */
34 void BRAKE_init(void) {
35     BRAKE_event = xEventGroupCreate();
36 }
37
38 /**
39  * @brief Waits until the BRAKE event flag is set. Then, resets
40  *      the event itself so it can wait for it again.
41  */
42 void BRAKE_wait_event(void) {
43     configASSERT(BRAKE_event != NULL);
44     xEventGroupWaitBits(BRAKE_event, BIT_SET, pdTRUE, pdFALSE,
45         ↪ portMAX_DELAY);
46 }
47
48 /**
49  * @brief Updates the flag #BRAKE_event indicating that the
50  *      brake intensity has changed so the brake task must run.
51  *
52  *      Notice that this method is not intended to be called from
53  *      an ISR.
54  */
55 void BRAKE_set_event(void) {
56     configASSERT(BRAKE_event != NULL);
57     xEventGroupSetBits(BRAKE_event, BIT_SET);
58 }
59
60 /**
61  * @brief Clears the BRAKE protected object, making all
62  *      further calls to protected object's methods
63  *      fail and block forever.
64  *
65  *      A successful call to #BRAKE_init will allow
66  *      new tasks to access these methods.
67  */
68
69 void BRAKE_clr(void) {
70     xEventGroupClearBits(BRAKE_event, BIT_SET);
71     vEventGroupDelete(BRAKE_event);
72 }

```

Listing 21: Cuerpo del objeto protegido brake.

C. Código fuente nodo 2

C.1. Cabeceras de código

```

1  #ifndef MODE_H
2  #define MODE_H
3
4  void MODE_init(void);

```

```
5 void MODE_set(int);
6 int MODE_get(void);
7
8 #endif /* MODE_H */
```

Listing 22: Cabecera del objeto protegido mode.

C.2. Cuerpo del código

```
1 /*
2  * Copyright © 2021 - present | modes.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 26/03/21 - modes.c.
18 */
19 #include "modes.h"
20 #include <lock.h>
21 #include <FreeRTOS.h>
22 #include <FreeRTOSConfig.h>
23
24 // Private variable storing protected object lock.
25 static Lock_t MODE_sem = NULL;
26
27 // Private variable storing the mode itself.
28 static int MODE_mode = 0;
29
30 /**
31  * @brief Initializes the protected object itself. This method
32  * must be called during code initialization so the other
33  * methods calls would work.
34  */
35 void MODE_init(void) {
36     MODE_sem = LOCK_create(NULL);
37 }
38
39 /**
40  * @brief Updates the stored mode safely using the #MODE_sem lock.
41  *
42  * @param mode the new mode to store.
43  */
44 void MODE_set(int mode) {
45     if (LOCK_acquire(MODE_sem) == pdTRUE) {
46         MODE_mode = mode;
47         LOCK_release(MODE_sem);
```



```
48|     }
49| }
50|
51| /**
52|  * @brief Obtains safely the stored mode, using the #MODE_sem lock.
53|  *
54|  * @return int - the stored mode. If any error occurs, returns -1.
55|  */
56| int MODE_get(void) {
57|     int mode = -1;
58|     if (LOCK_acquire(MODE_sem) == pdTRUE) {
59|         mode = MODE_mode;
60|         LOCK_release(MODE_sem);
61|     }
62|     return mode;
63| }
```

Listing 23: Cuerpo del objeto protegido mode.