STRD – Detección de distracciones al volante

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

2021

Resumen

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

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

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

Índice

1. Introducción

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

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

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

El sistema a desarrollar contará con múltiples sensores:

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

y múltiples actuadores:

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

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

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

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

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

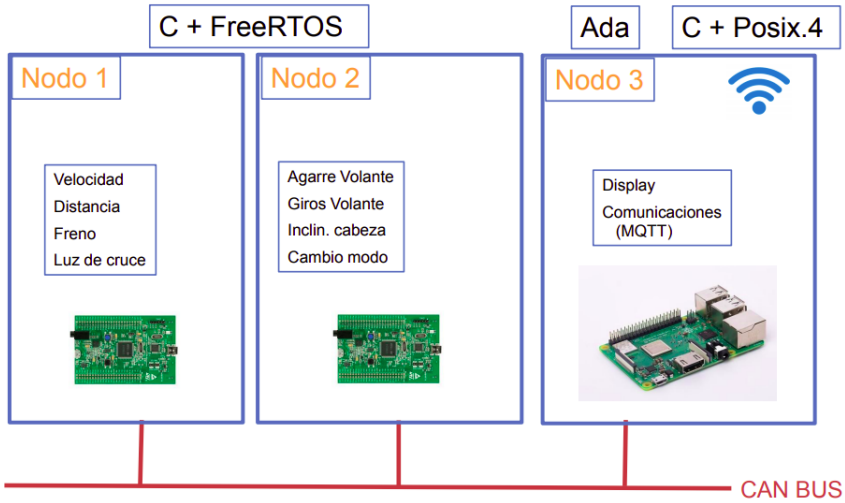

Figura 1: Modelo completo del sistema a implementar. Las tareas van distribuídas entre los dos nodos principales y se comunican entre ellos mediante CANBus.

1.1. Nodo 1

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

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

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

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

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

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

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

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

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

Todo este sistema viene modelado por la figura 2:

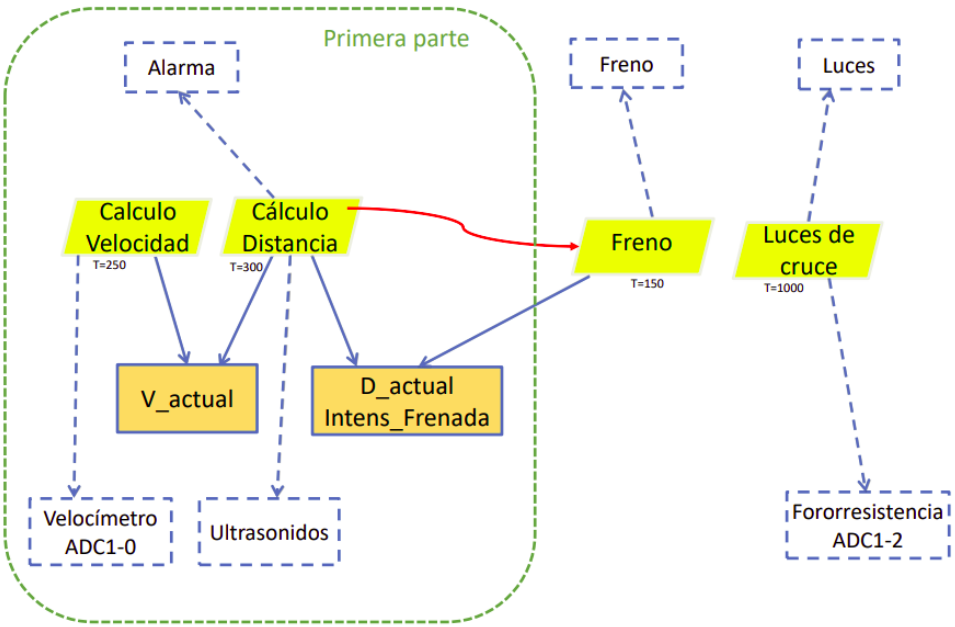

Figura 2: Modelado del nodo 1 junto con sus tareas, objetos protegidos, sensores y actuadores.

1.2. Nodo 2

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

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

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

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

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

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

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

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

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

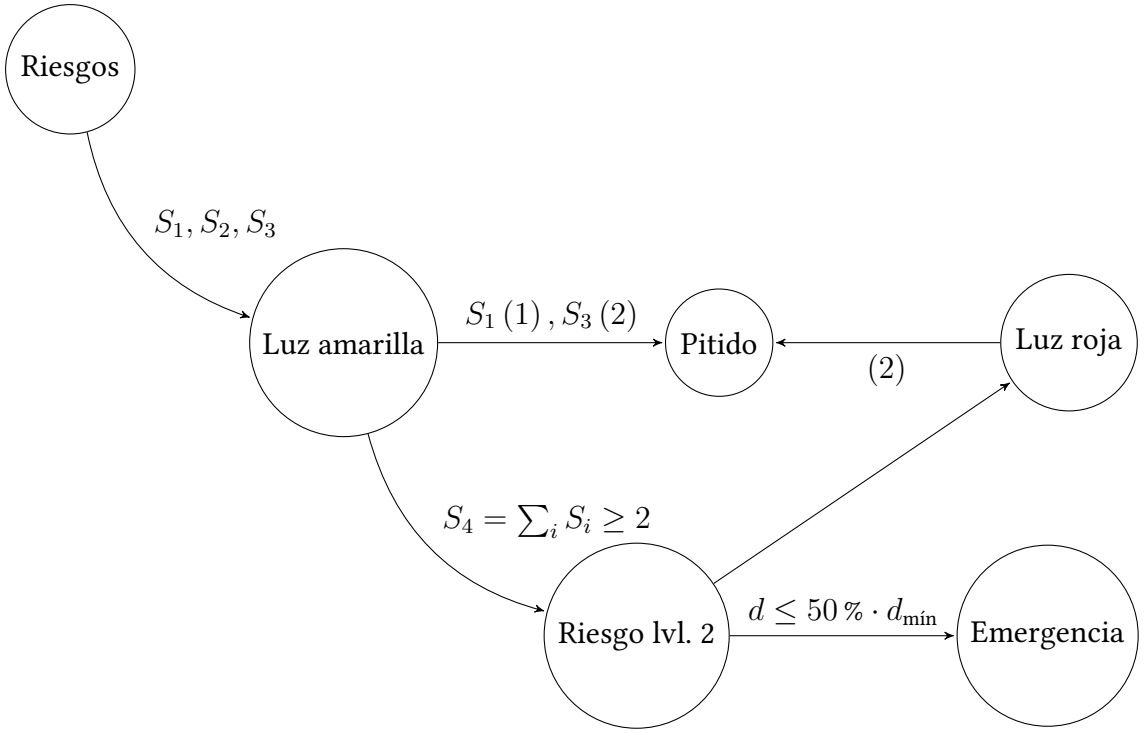

Figura 3: Diagrama que modela la interpretación de los riesgos, descritos en la enumeración anterior (S_i). La intensidad del pitido va acompañada entre paréntesis del síntoma que lo activa (por ejemplo, $S_1\,(1)$ indica una intensidad de pitido nivel 1) o en solitario, si es consecuencia de acciones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

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

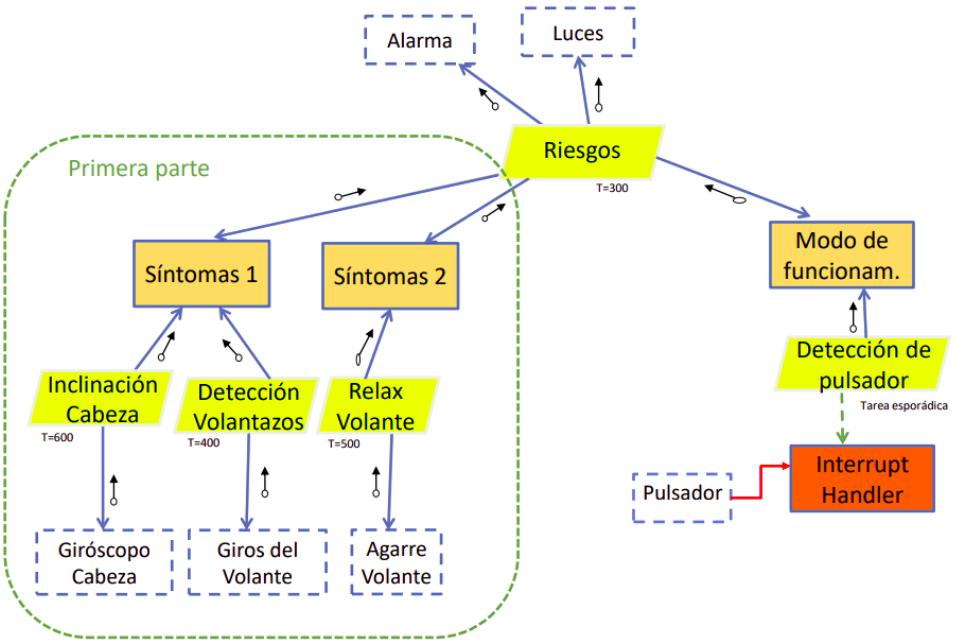

Figura 4: Modelado del nodo 2 junto con sus tareas, objetos protegidos, sensores y actuadores.

2. Implementación

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

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

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

2.1. Nodo 1

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

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

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

Listing 1: Tarea periódica que controla el acelerador.

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

La función de `map` viene definida en los códigos 15 y 18. El objeto protegido `SPEED` sigue la definición estándar del resto de objetos protegidos y viene definido en los códigos 21 (cabeceras) y 25 (cuerpo).

Por otra parte, el envío de datos mediante el CANBus se realiza mediante la librería `can`, definida en los códigos 16 y 19.

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 22 y 26) 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 23 y 27); 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 24 y 28); 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 23 y 27); 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.

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 17 y 20. 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   /* Reset of all peripherals, Initializes the Flash interface and the Systick.
      ↪ */
262   HAL_Init();
263
264   /* Configure the system clock */
265   SystemClock_Config();
266
267   /* Initialize all configured peripherals */
268   MX_GPIO_Init();
269   MX_ADC1_Init();
270   MX_SPI1_Init();
271   CAN_init();
272   SPEED_init();
273   DISTANCE_init();
274   BRAKE_init();
275
276   /* Create the thread(s) */
277   xTaskCreate((TaskFunction_t) acelerador,
278               "lectura potenciometro",
279               configMINIMAL_STACK_SIZE,
280               NULL, PR_TAREA2, NULL);
281   xTaskCreate((TaskFunction_t) lucesCruce,
282               "lectura luces",
283               configMINIMAL_STACK_SIZE,
284               NULL, PR_TAREA2, NULL);
285   xTaskCreate((TaskFunction_t) distanceTask,
286               "lectura distancia",
287               configMINIMAL_STACK_SIZE,
288               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 19, 25, 27, 28.

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 void deteccionPulsador(const void *argument) {
117   uint32_t wake_time = osKernelSysTick();
118   while(true) {
119     if (xSemaphoreTake(interrupcion, portMAX_DELAY) == pdTRUE) {
120       modo = ++modo % 3;
121       MODE_set(modo);
122     }
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 29 y 32).

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.

Por otra parte, la tarea que se encarga de detectar si hay o no volantazos viene definida por el código 8:

```
126 /**
127  * @brief Tarea periódica (400 ms) que actualiza la posición
128  *        del volante.
129  *
130  * @param argument lista de posibles argumentos a usar. Vacía por defecto.
131  */
132 void giroVolante(const void *argument) {
133   int actual;
134   int speed;
135   bool is_swerving = false;
136   uint8_t counter = 0;
137   uint32_t wake_time = osKernelSysTick();
138   while (true) {
139     ADC_ChannelConfTypeDef sConfig = {0};
140     sConfig.Channel = ADC_CHANNEL_0;
141     sConfig.Rank = 1;
142     sConfig.SamplingTime = ADC_SAMPLETIME_28CYCLES;
143     HAL_ADC_ConfigChannel(&hadc1, &sConfig);
144     HAL_ADC_Start(&hadc1);
145     if (HAL_ADC_PollForConversion(&hadc1, 5) == HAL_OK) {
146       actual = HAL_ADC_GetValue(&hadc1);
147       WHEEL_set(actual);
148       speed = SPEED_get();
149       is_swerving = WHEEL_update_swerving(speed);
150       if (is_swerving) counter = 0;
151       else if (counter == 13) WHEEL_set_is_swerving(false);
152       else {
153         WHEEL_set_is_swerving(true);
154         ++counter;
155       }
156     }
157     osDelayUntil(&wake_time, T_TAREAGIRO);
158   }
159 }
```

Listing 8: Tarea periódica de control del giro del volante.

En dicha tarea se lee desde el ADC (canal 0) y se actualiza el valor de la posición del volante en la variable `actual` (líneas 139 – 146); a continuación se conserva el dato en el objeto protegido `symptoms` (línea 147) y se comprueba, accediendo a la velocidad actual, si el conductor está dando volantazos o no. Esto se realiza en las líneas 148 – 155, en donde

se aprovechan los recursos provistos por el objeto protegido `wheel` para verificar si la se ha pegado un volantazo (códigos 30 y 33). Si durante 13 iteraciones no se ha reseteado el contador (no ha habido volantazo) se quita el síntoma.

En lo referente a la tarea que identifica si el volante está siendo sujeto o no, se define mediante el código 9:

```
161 /**
162  * @brief Tarea periódica (500 ms) que actualiza si el volante está agarrado o
        ↪ no
163  *
164  * @param argument lista de posibles argumentos a usar. Vacía por defecto.
165  */
166 void volanteAgarrado(const void *argument) {
167   int actual;
168   uint32_t wake_time = osKernelSysTick();
169   while (true) {
170     actual = HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_8);
171     WHEEL_grab(actual);
172     osDelayUntil(&wake_time, T_TAREAAGARRADO);
173   }
174 }
```

Listing 9: Tarea periódica de control de sujección del volante.

Dicha tarea sencillamente leerá el valor del GPIO correspondiente (línea 170) y actualizará el valor en el objeto protegido (línea 171), definido en los códigos 30 y 33.

Para trabajar con la inclinación de la cabeza, se ha de acceder a los valores provistos por el giroscopio integrado en la placa. El código de la tarea viene definido por 10:

```
176 /**
177  * @brief Tarea periódica (600 ms) que actualiza la posición de la cabeza.
178  *
179  * @param argument lista de posibles argumentos a usar. Vacía por defecto.
180  */
181 void Tarea_Control_Inclinacion(void const *argument) {
182   uint32_t wake_time = osKernelSysTick();
183   while (true) {
184     Ix1 = SPI_Read(0x28);
185     Ix2 = SPI_Read(0x29);
186     Ix = (Ix2 << 8) + Ix1;
187     if (Ix ⩾ 0x8000)
188       Ix = -(65536 - Ix);
189     X = Ix / 16384.0;
190
191     Iy1 = SPI_Read(0x2A);
192     Iy2 = SPI_Read(0x2B);
193     Iy = (Iy2 << 8) + Iy1;
194     if (Iy ⩾ 0x8000)
195       Iy = -(65536 - Iy);
196     Y = Iy / 16384.0;
197
198     Iz1 = SPI_Read(0x2C);
199     Iz2 = SPI_Read(0x2D);
200     Iz = (Iz2 << 8) + Iz1;
201     if (Iz ⩾ 0x8000)
202       Iz = -(65536 - Iz);
```

```
203     Z = Iz / 16384.0;
204
205     rotX = atan2(Y, sqrt(X * X + Z * Z)) * 180.0 / 3.1416;
206     rotY = -atan2(X, sqrt(Y * Y + Z * Z)) * 180.0 / 3.1416;
207     GIROSCOPE_set(rotX, rotY, 0);
208
209     osDelayUntil(&wake_time, T_CABEZA);
210   }
211 }
```

Listing 10: Tarea periódica de control de la inclinación de la cabeza.

De todo el código anterior, los datos se actualizan en la línea 207 en donde guardan en el objeto protegido `symptoms` (códigos 30 y 33).

Finalmente, la tarea de riesgos. Esta tarea accede a todos los objetos protegidos y además obtiene información mediante el CANBus, por lo que su programación es delicada. Dicha tarea viene definida por el código 11:

```
213 /**
214  * @brief Tarea de riesgos periódica (300 ms) que actúa directamente
215  *        sobre los distintos sensores y actuadores para aletar al
216  *        conductor de ciertos peligros o problemas. Lee de todos los
217  *        objetos protegidos pero no actualiza datos, solo muestra
218  *        información.
219  *
220  * @param args lista de posibles argumentos a usar. Vacía por defecto.
221  */
222 void risks_task(const void *args) {
223   const uint32_t T_RISKS_TASK = 300U;
224   uint32_t wake_time = osKernelSysTick();
225   float x, y;
226   bool wheel_is_grabbed;
227   int speed;
228   bool wheel_is_swerving;
229   bool is_distance_ok;
230   int risk_count = 0;
231   int working_mode = 0;
232   while (true) {
233     x = GIROSCOPE_get_X();
234     y = GIROSCOPE_get_Y();
235     wheel_is_grabbed = WHEEL_is_grabbed();
236     speed = SPEED_get();
237     wheel_is_swerving = WHEEL_get_is_swerving();
238     is_distance_ok = DISTANCE_get_security();
239     working_mode = MODE_get();
240
241     if (working_mode < 2) {
242       if (abs(x) ⩾ 20 && abs(y) ⩾ 20 && !wheel_is_grabbed) {
243         if (working_mode == 0) {
244           // Pitido lvl. 1
245           // Luz amarilla ON
246           HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_SET);
247         }
248         risk_count++;
249       }
250       if ((abs(x) ⩾ 20 || abs(y) ⩾ 20) && wheel_is_grabbed && speed ⩾ 70) {
251         if (working_mode == 0) {
```

```
252             // Luz amarilla ON
253             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_SET);
254           }
255           risk_count++;
256         }
257         if (abs(x) >= 30 && wheel_is_swerving) {
258           if (working_mode == 0) {
259             // Pitido lvl. 2
260             // Luz amarilla ON
261             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_SET);
262           }
263           risk_count++;
264         }
265         if (risk_count == 0) {
266           // Pitido off
267           // Luz amarilla OFF
268           HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_RESET);
269         } else if (risk_count >= 2) {
270           if (working_mode >= 0 || !is_distance_ok) {
271             // Pitido lvl. 2
272             // Luz roja
273             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_SET);
274           }
275         } else {
276           // Luz roja off
277           HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_RESET);
278         }
279       } else {
280         // Pitido OFF
281         HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_RESET);
282         HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_RESET);
283       }
284       risk_count = 0;
285
286       osDelayUntil(&wake_time, T_RISKS_TASK);
287     }
288 }
```

Listing 11: Tarea periódica de control de riesgos.

Lo primero que se realiza en esta tarea es comprobar si el modo de funcionamiento de las alarmas es menor a 2 (línea 241). Esto quiere decir que las alarmas están habilitadas al completo o parcialmente (un valor de 2 indicaría que ninguna alarma está activa).

Si hay un modo de alarma activado, se empiezan a realizar las comprobaciones:

- En la línea 242 se comprueba que existe una inclinación simultánea en los ejes X e Y y que además no se tiene el volante sujeto. Si está el estado de alarma básico activo (línea 243), se enciende la luz amarilla y se emite un pitido con intensidad 1.
- En la línea 250 se comprueba si la cabeza está inclinada al menos en uno de los ejes, si el volante está sujeto y si se circula a más de $70^{km}/_{h}$, en cuyo caso se encendería la luz amarilla.
- Si se inclina la cabeza más de 30º en el eje X y se están dando volantazos (línea 257) entonces se emite un pitido nivel dos y se enciende la luz amarilla.

- Si no se ha dado ninguna situación de riesgo (`risk_count == 0`), se apaga el pitido y la luz amarilla (líneas 265 – 268). Si, por el contrario, se ha producido al menos dos riesgos se activa el pitido nivel dos y se enciende la luz roja (líneas 269 – 275). En otro caso, se apagan.

Con esta lógica, el diagrama 3 queda implementado a nivel de código. En este caso, se utilizan los objetos protegidos de síntomas (códigos 30 y 33), los que contienen los datos recibidos por el CANBus (`node1` definido por 31 y 34) y el modo (códigos 29 y 32).

Finalmente, para poder recibir los datos por el CANBus se han creado dos tareas esporádicas las cuales esperan un evento para poder actualizar los valores contenidos en los objetos protegidos. Dichas funciones vienen definidas por el código 12:

```
290 /**
291  * @brief Tarea esporádica encargada únicamente de actualizar el valor
292  *        contenido de la velocidad cuando llega un mensaje por el
293  *        CANBus que contiene una cabecera #STD_ID1.
294  *
295  *        El tratamiento se realiza en una tarea esporádica para evitar
296  *        bloquear en demasía la rutina de tratamiento de interrupción
297  *        del CANBus.
298  *
299  * @param args lista de posibles argumentos a usar. Vacía por defecto.
300  */
301 void CAN_speed_task(const void *args) {
302   while (true) {
303     SPEED_wait_recv();
304     SPEED_set(CAN_recv());
305   }
306 }
307
308 /**
309  * @brief Tarea esporádica encargada únicamente de actualizar el valor
310  *        contenido de la distancia cuando llega un mensaje por el
311  *        CANBus que contiene una cabecera #STD_ID2.
312  *
313  *        El tratamiento se realiza en una tarea esporádica para evitar
314  *        bloquear en demasía la rutina de tratamiento de interrupción
315  *        del CANBus.
316  *
317  * @param args lista de posibles argumentos a usar. Vacía por defecto.
318  */
319 void CAN_distance_task(const void *args) {
320   while (true) {
321     DISTANCE_wait_recv();
322     DISTANCE_set(CAN_recvf());
323   }
324 }
```

Listing 12: Tareas esporádicas de gestión del CANBus.

Por otra parte, según la definición del código del CANBus (ver código 19), es necesario llamar a la función `CAN_Handle_IRQ` desde la rutina de tratamiento de interrupción para recibir el mensaje de forma efectiva. Por ello, es necesario editar el fichero `stm32f4xx_it.c` y añadir la llamada a la función, como se muestra en el código 13:

```c
#include "main.h"
#include "stm32f4xx_it.h"
#include "cmsis_os.h"
#include "can.h"
/**
  * @brief This function handles CAN1 RX0 interrupts.
  */
void CAN1_RX0_IRQHandler(void)
{
  /* USER CODE BEGIN CAN1_RX0_IRQn 0 */

  /* USER CODE END CAN1_RX0_IRQn 0 */
  // Handle the interruption using can.h object
  CAN_Handle_IRQ();
  /* USER CODE BEGIN CAN1_RX0_IRQn 1 */

  /* USER CODE END CAN1_RX0_IRQn 1 */
}
```

Listing 13: Rutina de tratamiento de interrupción del CANBus.

Al igual que en el nodo 1, el uso de las distintas librerías requiere una carga inicial en la función `main`, ya que en otro caso las funciones resultarán inaccesibles. Para este nodo, la función de inicialización sería la definida en el código 14:

```c
/**
 * @brief  The application entry point.
 * @retval int
 */
int main(void) {

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick.
   ↪ */
  HAL_Init();
  /* Configure the system clock */
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_ADC1_Init();
  MX_SPI1_Init();
  MODE_init();
  CAN_init();
  NODE1_init();
  SYMPTOMS_init();
  Inicializa_Acelerometro();

  interrupcion = xSemaphoreCreateBinary();

  xTaskCreate((TaskFunction_t) giroVolante,
              "lectura potenciometro Giro Volante",
              configMINIMAL_STACK_SIZE,
              NULL, PR_TAREAGIRO, NULL);

```

```
397   xTaskCreate((TaskFunction_t) volanteAgarrado,
398               "lectura sensor agarrado",
399               configMINIMAL_STACK_SIZE,
400               NULL, PR_TAREAAGARRADO, NULL);
401
402   xTaskCreate((TaskFunction_t) Tarea_Control_Inclinacion,
403               "lectura giroscopio",
404               configMINIMAL_STACK_SIZE,
405               NULL, PR_CABEZA, NULL);
406
407   xTaskCreate((TaskFunction_t) deteccionPulsador,
408               "Tarea esporadica",
409               configMINIMAL_STACK_SIZE,
410               NULL, 0, NULL);
411
412   xTaskCreate((TaskFunction_t) risks_task,
413               "Risks task",
414               configMINIMAL_STACK_SIZE,
415               NULL, PR_RIESGOS, NULL);
416
417   xTaskCreate((TaskFunction_t) CAN_speed_task,
418               "CANBus speed recv task",
419               configMINIMAL_STACK_SIZE,
420               NULL, PR_SPEED_MSG, NULL);
421
422   xTaskCreate((TaskFunction_t) CAN_distance_task,
423               "CANBus distance recv task",
424               configMINIMAL_STACK_SIZE,
425               NULL, PR_DISTANCE_MSG, NULL);
426
427   /* Start scheduler */
428   vTaskStartScheduler();
429   /* We should never get here as control is now taken by the scheduler */
430
431   /* Infinite loop */
432   while (1);
433 }
```

Listing 14: Código del `main` para el nodo 2.

3. Diseño final

El diseño final del circuito se muestra en la figura 5:

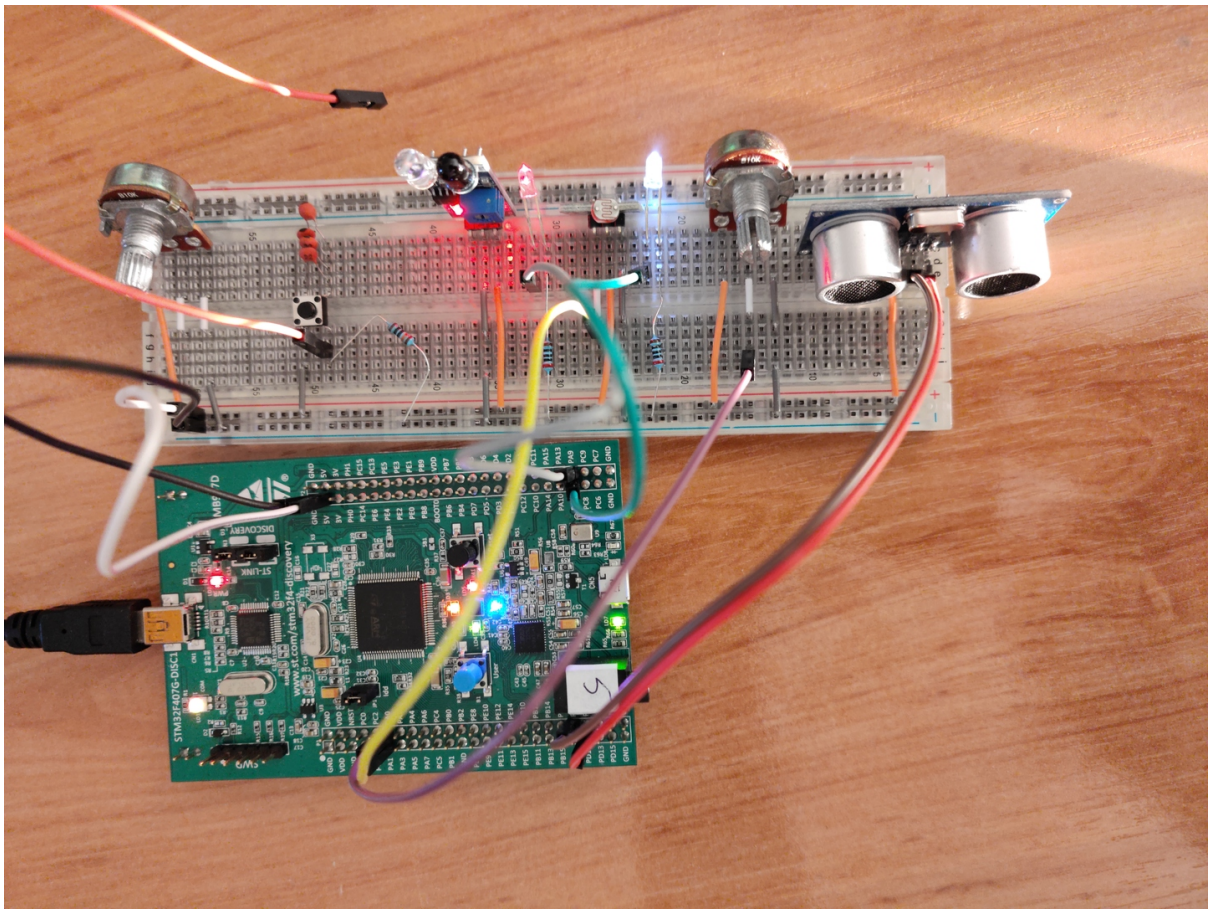

Figura 5: Diseño final del circuito montado sobre una *protoboard*.

Como se puede ver en la figura anterior, todo el circuito (de los dos nodos) se ha podido montar sobre la misma placa de desarrollo: se pueden apreciar los dos potenciómetros (volante y acelerador), el sensor de presencia para el agarre del volante, LDR para luz ambiental, múltiples LEDs para mostrar señales, etc.

Con el código implementado se ha conseguido que las tareas funcionen como se esperaban y que se puedan realizar algunas comunicaciones mediante el CANBus. Sin embargo, la falta de tiempo (e inexperiencia del equipo) durante el desarrollo del proyecto ha dejado *bugs* sin resolver y el código no está todo lo depurado que convendría para un sistema en tiempo real.

Por otra parte, y en estrecha relación con lo mencionado anteriormente, solo se han podido implementar dos nodos, teniendo que postergar el desarrollo del nodo *display* para otro momento.

Finalmente, se asume que el sistema es planificable ya que los accesos a CPU y recursos no son tan grandes como para que alguna tarea saliese no planificable. Sin embargo, sería interesante poder realizar observaciones y mediciones sobre un prototipo en avanzado estado de desarrollo para extraer el WCET C_i de cada tarea, realizar el RTA (*Response Time Analysis*) y verificar que efectivamente el sistema es planificable o, en otro caso, realizar los ajustes pertinentes para que lo sea.

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.
- En el código 8 se esperan 13 iteraciones que equivalen a un tiempo de $5{,}2$ s (en lugar de los 5 s pedidos). Esto es debido a que el periodo no es múltiplo, por lo que se comete un error "a la alta" en lugar de "a la baja".
- Los ficheros `lock.h/.c` permiten trabajar con semáforos definidos de forma estática en lugar de forma dinámica (código 20). Sin embargo, para poder aprovechar dicha funcionalidad es necesario habilitar en el fichero `FreeRTOSConfig.h` la macro:

  ```
  #define configSUPPORT_STATIC_ALLOCATION 1
  ```

  con un valor '1'. En otro caso, aunque se pase un parámetro válido, no se usará dicha funcionalidad.
- El pitido no se ha implementado a nivel de código ya que no venía especificado en los distintos diagramas a qué pin habría que conectarlo ni la lógica de control a usar para que funcionase con cierta intensidad.
- En el código 12 se han implementado dos tareas esporádicas para la gestión de los mensajes del CANBus porque dicho dispositivo recibe mensajes desde una rutina de interrupción. Como se quieren actualizar variables las cuales utilizan un `lock` internamente, se deriva su gestión a un par de tareas esporádicas cuya única finalidad es la de actualizar el valor de los objetos protegidos.
- En los códigos 16 y 19, como son en apariencia iguales para ambos nodos, es necesario definir una macro para que la versión del CANBus se adapte a la placa en que se ejecuta. De esta forma, en el nodo 2 habría que definir en alguna cabecera (se sugiere `FreeRTOSConfig.h`) la macro:

  ```
  #define NODE_2
  ```
- En el código 11 no se ha implementado la parte de interacción con el freno ya que la tarea `Cálculo distancia` (código 2) se encarga individualmente de accionar el freno según la distancia de seguridad y una intensidad predefinida. Además, en las diapositivas de la especificación no se menciona esta comunicación con el nodo 1, por lo que se ha asumido que el verdadero encargado de accionar el freno es el nodo 1.

5. Glosario

A. Código fuente común

A.1. Cabeceras de código

```c
/*
 * Copyright © 2021 - present | utils.h by Javinator9889
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see https://www.gnu.org/licenses/.
 *
 * Created by Javinator9889 on 10/04/21 - utils.h.
 */
#ifndef UTILS_H
#define UTILS_H
#include <stdint.h>

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

// 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 15: 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 16: Cabecera de la librería CANBus.

```
/*
 * Copyright © 2021 - present | lock_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 - lock_h.
 */

#ifndef LOCK_H
#define LOCK_H
#include <FreeRTOS.h>
#include <stddef.h>
#include <semphr.h>

// Custom data type used for identifying LOCK created locks.
typedef SemaphoreHandle_t Lock_t;

Lock_t LOCK_create(StaticSemaphore_t*);
void LOCK_destroy(Lock_t);
long LOCK_acquire(Lock_t);
void LOCK_release(Lock_t);

#endif /* LOCK_H */
```

Listing 17: Cabecera de la librería `lock`.

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.
 *
```

```
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) + out_min)
       ↪ ;
33 }
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 18: Cuerpo de las funciones de utilidad.

```c
/*
 * Copyright © 2021 - present | can.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 - can.c.
 */
#include "can.h"
#include <stm32f4xx_hal.h>
#include <FreeRTOS.h>
#include <FreeRTOSConfig.h>
#include <task.h>
#include <stdint.h>
#include "utils.h"
#ifdef NODE_2
#include "node1.h"
#endif

const uint32_t STD_ID1 = 0x6FA;
const uint32_t STD_ID2 = 0x6FB;
const uint32_t HFILTER_ID = 0x6FF << 5;

#ifdef NODE_2
const uint32_t HFILTER_MASK = 0x7F0 << 5;
#endif

static volatile CAN_HandleTypeDef hcan1;
#ifndef NODE_2
static volatile CAN_TxHeaderTypeDef tx_header;
static volatile CAN_TxHeaderTypeDef tx_header2;
#else
static volatile CAN_RxHeaderTypeDef rx_header;
#endif
static volatile uint32_t tx_mailbox;

static volatile uint8_t byte_sent = 0;
static volatile uint8_t byte_recv = 0;
static volatile float float_recv = .0F;

static volatile CAN_FilterTypeDef filter_config;

/**
  * @brief CAN1 Initialization Function - extracted from main.
  */
```

```
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_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.
```

```
 *
 *        Notice that this value will only be updated when the received
 *        message ID matches the #STD_ID2.
 *
 * @return float - the stored float.
 */
float CAN_recvf(void) {
    return float_recv;
}

/**
 * @brief This method must be called if the board wants to receive
 *        CANBus messages during the CANBus interruption routine.
 *
 *        By filtering the ID, identifies whether the received array
 *        is either a single byte or a float value.
 *
 *        In addition, this method sets a flag at the respective
 *        protected objects indicating that a new message is received
 *        and is ready to be used (notice that this method is called
 *        from an IRQ, so the processing must be as efficient as
 *        possible. In this function, setting a flag is easy and
 *        not blocking - at least not as much as changing a lock/semaphore).
 *
 *        The affected protected objects are the ones that store the
 *        SPEED and the DISTANCE.
 *
 * @see node1.h
 */
void CAN_Handle_IRQ(void) {
    HAL_CAN_IRQHandler(&hcan1);
#ifdef NODE_2
    uint8_t bytes[4];
    HAL_CAN_GetRxMessage(&hcan1, CAN_RX_FIFO0, &rx_header, &bytes);
    if (rx_header.StdId == STD_ID1) {
        byte_recv = bytes[0];
        SPEED_set_recv();
    }
    if (rx_header.StdId == STD_ID2) {
        float_recv = b2f(&bytes[0]);
        DISTANCE_set_recv();
    }
#endif
}
```

Listing 19: Cuerpo de la librería CANBus.

```
/*
 * Copyright © 2021 - present | lock_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
```

```
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) && (configSUPPORT_STATIC_ALLOCATION
    ↪  == 1)
47     if (mutexBuffer ≠ NULL) xSemaphore = xSemaphoreCreateMutexStatic(
    ↪ mutexBuffer);
48     else
49 #endif
50     xSemaphore = xSemaphoreCreateMutex();
51
52     configASSERT(xSemaphore ≠ NULL);
53     configASSERT(xSemaphoreGive(xSemaphore) ≠ pdTRUE);
54
55     return (Lock_t) xSemaphore;
56 }
57
58 /**
59  * @brief destroys the given lock (release from memory). After called,
60  *        the memory will be empty and the lock cannot be used anymore.
61  *
62  * @param sem the lock to destroy.
63  */
64 inline void LOCK_destroy(Lock_t sem) {
65     vSemaphoreDelete((SemaphoreHandle_t) sem);
66 }
67
```

```
68 /**
69  * @brief tries to acquire the given lock, blocking forever if necessary.
70  *
71  * @param sem the lock to acquire.
72  * @return long - #pdTRUE if the semaphore was acquired. #pdFALSE otherwise.
73  */
74 inline long LOCK_acquire(Lock_t sem) {
75     configASSERT(sem ≠ NULL);
76     return xSemaphoreTake((SemaphoreHandle_t) sem, portMAX_DELAY);
77 }
78
79 /**
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 20: 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 21: 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 22: 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 23: 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 24: 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.
```

```
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see https://www.gnu.org/licenses/.
 *
 * Created by Javinator9889 on 05/03/21 - speed.c.
 */
#include "speed.h"
#include <lock.h>
#include <semphr.h>
#include <FreeRTOS.h>
#include <FreeRTOSConfig.h>
#include <task.h>

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

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

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

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

Listing 25: 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 26: 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
```

```
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 /**
```

```
 * @brief Deletes all stored objects and resets the
 *        distance value. After this method call,
 *        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 27: Cuerpo del objeto protegido `distance`.

```
/*
 * Copyright © 2021 - present | brake.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 13/03/21 - brake.c.
 */
#include "brake.h"
```

```c
#include <FreeRTOS.h>
#include <FreeRTOSConfig.h>
#include <task.h>
#include <event_groups.h>

// Private variable storing the BRAKE flag
static EventGroupHandle_t BRAKE_event = NULL;

/**
 * @brief Initializes BRAKE protected object. This method must be called
 *        during the early boot of the application in order to be able
 *        to use the object's methods.
 *
 */
void BRAKE_init(void) {
    BRAKE_event = xEventGroupCreate();
}

/**
 * @brief Waits until the BRAKE event flag is set. Then, resets
 *        the event itself so it can wait for it again.
 */
void BRAKE_wait_event(void) {
    configASSERT(BRAKE_event ≠ NULL);
    xEventGroupWaitBits(BRAKE_event, BIT_SET, pdTRUE, pdFALSE, portMAX_DELAY);
}

/**
 * @brief Updates the flag #BRAKE_event indicating that the
 *        brake intensity has changed so the brake task must run.
 *
 *        Notice that this method is not intended to be called from
 *        an ISR.
 *
 */
void BRAKE_set_event(void) {
    configASSERT(BRAKE_event ≠ NULL);
    xEventGroupSetBits(BRAKE_event, BIT_SET);
}

/**
 * @brief Clears the BRAKE protected object, making all
 *        further calls to protected object's methods
 *        fail and block forever.
 *
 *        A successful call to #BRAKE_init will allow
 *        new tasks to access these methods.
 *
 */
void BRAKE_clr(void) {
    xEventGroupClearBits(BRAKE_event, BIT_SET);
    vEventGroupDelete(BRAKE_event);
}
```

Listing 28: 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 29: Cabecera del objeto protegido `mode`.

```
1  /*
2   * Copyright © 2021 - present | symptomps.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 - symptoms.h.
18  */
19 #ifndef SYMPTOMS_H
20 #define SYMPTOMS_H
21 #include <stdbool.h>
22
23 void SYMPTOMS_init(void);
24
25 void GIROSCOPE_set(float, float, float);
26 float GIROSCOPE_get_X(void);
27 float GIROSCOPE_get_Y(void);
28 float GIROSCOPE_get_Z(void);
29
30 void WHEEL_set(int);
31 int WHEEL_get(void);
32
33 void WHEEL_set_is_swerving(bool);
34 bool WHEEL_get_is_swerving(void);
35 bool WHEEL_update_swerving(int);
36
37 void WHEEL_grab(bool);
38 bool WHEEL_is_grabbed(void);
39
40 #endif /* SYMPTOMPS_H */
```

Listing 30: Cabecera del objeto protegido `symptoms`.

```c
/*
 * Copyright © 2021 - present | node1.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 22/04/21 - node1.h.
 */
#ifndef NODE1_H
#define NODE1_H
#include <stdbool.h>

// The BIT flag set used for identifying if the flag
// is set or not.
#define BIT_SET (0x02UL)

void NODE1_init(void);

void SPEED_set(int);
int SPEED_get(void);
void SPEED_set_recv(void);
void SPEED_wait_recv(void);

void DISTANCE_set(float);
float DISTANCE_get(void);
bool DISTANCE_get_security(void);
void DISTANCE_set_recv(void);
void DISTANCE_wait_recv(void);

#endif /* NODE1_H */
```

Listing 31: Cabecera del objeto protegido node1.

C.2. Cuerpo del código

```c
/*
 * Copyright © 2021 - present | modes.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 - modes.c.
 */
#include "modes.h"
#include <lock.h>
#include <FreeRTOS.h>
#include <FreeRTOSConfig.h>

// Private variable storing protected object lock.
static Lock_t MODE_sem = NULL;

// Private variable storing the mode itself.
static int MODE_mode = 0;

/**
 * @brief Initializes the protected object itself. This method
 *        must be called during code initialization so the other
 *        methods calls would work.
 */
void MODE_init(void) {
    MODE_sem = LOCK_create(NULL);
}

/**
 * @brief Updates the stored mode safely using the #MODE_sem lock.
 *
 * @param mode the new mode to store.
 */
void MODE_set(int mode) {
    if (LOCK_acquire(MODE_sem) == pdTRUE) {
        MODE_mode = mode;
        LOCK_release(MODE_sem);
    }
}

/**
 * @brief Obtains safely the stored mode, using the #MODE_sem lock.
 *
 * @return int - the stored mode. If any error occurs, returns -1.
 */
int MODE_get(void) {
    int mode = -1;
    if (LOCK_acquire(MODE_sem) == pdTRUE) {
        mode = MODE_mode;
        LOCK_release(MODE_sem);
    }
    return mode;
}
```

Listing 32: Cuerpo del objeto protegido `mode`.

```
/*
 * Copyright © 2021 - present | symptomps.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 - symptoms.h.
 */
#include "symptoms.h"
#include <stdlib.h>
#include <lock.h>
#include <semphr.h>
#include <FreeRTOS.h>
#include <FreeRTOSConfig.h>
#include <task.h>
#include <stdbool.h>

// Private variable storing the SYMPTOMS1 lock.
static Lock_t SYMPTOMS1_sem = NULL;

// Private variable storing the SYMPTOMS1 lock.
static Lock_t SYMPTOMS2_sem = NULL;

// Private variable storing giroscope X value.
static float GIROSCOPE_x = .0F;
// Private variable storing giroscope Y value.
static float GIROSCOPE_y = .0F;
// Private variable storing giroscope Z value.
static float GIROSCOPE_z = .0F;

// Private variable storing the steering wheel position.
static int WHEEL_old_position = 0;
static int WHEEL_position = 0;
// Private variable for setting if the steering wheel is swerving
// or not
static bool WHEEL_is_swerving = false;
// Private variable storing whether the steering wheel is
// grabbed or not
static bool WHEEL_status_grab = false;

/**
 * @brief Initializes the protected object containing the symptoms.
 *        This method must be called during the early boot of the
 *        code so the rest of the methods available will work
 *        as expected.
 */
void SYMPTOMS_init(void) {
    SYMPTOMS1_sem = LOCK_create(NULL);
    SYMPTOMS2_sem = LOCK_create(NULL);
}
```

```
61
62 /**
63  * @brief Safely updates the stored values of the giroscope positions.
64  *
65  * @param x the new X position.
66  * @param y the new Y position.
67  * @param z the new Z position.
68  */
69 void GIROSCOPE_set(float x, float y, float z) {
70     LOCK_acquire(SYMPTOMS1_sem);
71     GIROSCOPE_x = x;
72     GIROSCOPE_y = y;
73     GIROSCOPE_z = z;
74     LOCK_release(SYMPTOMS1_sem);
75 }
76
77 /**
78  * @brief Safely obtains the X value of the giroscope.
79  *
80  * @return float - the X value.
81  */
82 float GIROSCOPE_get_X(void) {
83     float x = -1;
84     LOCK_acquire(SYMPTOMS1_sem);
85     x = GIROSCOPE_x;
86     LOCK_release(SYMPTOMS1_sem);
87     return x;
88 }
89
90 /**
91  * @brief Safely obtains the Y value of the giroscope.
92  *
93  * @return float - the Y value.
94  */
95 float GIROSCOPE_get_Y(void) {
96     float y = -1;
97     LOCK_acquire(SYMPTOMS1_sem);
98     y = GIROSCOPE_y;
99     LOCK_release(SYMPTOMS1_sem);
100    return y;
101 }
102
103 /**
104  * @brief Safely obtains the Z value of the giroscope.
105  *
106  * @return float - the Z value.
107  */
108 float GIROSCOPE_get_Z(void) {
109    float z = -1;
110    LOCK_acquire(SYMPTOMS1_sem);
111    z = GIROSCOPE_z;
112    LOCK_release(SYMPTOMS1_sem);
113    return z;
114 }
115
116 /**
117  * @brief Safely sets the steering wheel position, in angles.
118  *
```

```
119  * @param position the new wheel position.
120  */
121 void WHEEL_set(int position) {
122     LOCK_acquire(SYMPTOMS1_sem);
123     WHEEL_old_position = WHEEL_position;
124     WHEEL_position = position;
125     LOCK_release(SYMPTOMS1_sem);
126 }
127
128 /**
129  * @brief Safely obtains the steering wheel position, in angles.
130  *
131  * @return int - the steering wheel position.
132  */
133 int WHEEL_get(void) {
134     int position = -1;
135     LOCK_acquire(SYMPTOMS1_sem);
136     position = WHEEL_position;
137     LOCK_release(SYMPTOMS1_sem);
138     return position;
139 }
140
141 /**
142  * @brief Safely sets if the steering wheel is swerving or not.
143  *
144  * @param is_swerving whether if the steering wheel is swerving or not.
145  */
146 void WHEEL_set_is_swerving(bool is_swerving) {
147     LOCK_acquire(SYMPTOMS1_sem);
148     WHEEL_is_swerving = is_swerving;
149     LOCK_release(SYMPTOMS1_sem);
150 }
151
152 /**
153  * @brief Safely checks if the steering wheel is swerving or not.
154  *
155  * @return true - if swerving.
156  * @return false - otherwise or if any error occurs.
157  */
158 bool WHEEL_get_is_swerving(void) {
159     bool is_swerving = false;
160     LOCK_acquire(SYMPTOMS1_sem);
161     is_swerving = WHEEL_is_swerving;
162     LOCK_release(SYMPTOMS1_sem);
163     return is_swerving;
164 }
165
166 /**
167  * @brief With the given speed, safely updates whether the vehicle is
168  *        swerving or not, based on proposed conditions.
169  *
170  * @param speed the current vehicle speed.
171  * @return true - if the vehicle is swerving.
172  * @return false - otherwise.
173  */
174 bool WHEEL_update_swerving(int speed) {
175     LOCK_acquire(SYMPTOMS1_sem);
176     WHEEL_is_swerving = ((speed > 70) && (abs(WHEEL_position -
```

```
    ↪ WHEEL_old_position) ⩾ 150));
177     LOCK_release(SYMPTOMS1_sem);
178     return WHEEL_is_swerving;
179 }
180
181 /**
182  * @brief Safely sets whether if the steering wheel is grabbed or not.
183  *
184  * @param is_grabbed true if grabbed/false otherwise.
185  */
186 void WHEEL_grab(bool is_grabbed) {
187     LOCK_acquire(SYMPTOMS2_sem);
188     WHEEL_status_grab = is_grabbed;
189     LOCK_release(SYMPTOMS2_sem);
190 }
191
192 /**
193  * @brief Safely obtains whether the steering wheel is grabbed or not.
194  *
195  * @return true - if the wheel is grabbed.
196  * @return false - if the wheel is not grabbed.
197  */
198 bool WHEEL_is_grabbed(void) {
199     bool is_grabbed = false;
200     LOCK_acquire(SYMPTOMS2_sem);
201     is_grabbed = WHEEL_status_grab;
202     LOCK_release(SYMPTOMS2_sem);
203     return is_grabbed;
204 }
205
206 /**
207  * @brief Destroys the symptoms protected object, freeing all the
208  *        used memory. After this method is called, the SYMPTOMS
209  *        object is not usable anymore until #SYMPTOMS_init is called
210  *        again.
211  */
212 void SYMPTOMS_delete(void) {
213     LOCK_destroy(SYMPTOMS1_sem);
214     LOCK_destroy(SYMPTOMS2_sem);
215
216     SYMPTOMS1_sem = NULL;
217     SYMPTOMS2_sem = NULL;
218
219     GIROSCOPE_x = .0F;
220     GIROSCOPE_y = .0F;
221     GIROSCOPE_z = .0F;
222
223     WHEEL_position = 0;
224     WHEEL_status_grab = false;
225 }
```

Listing 33: Cuerpo del objeto protegido `symptoms`.

```
1 /*
2  * Copyright © 2021 - present | node1.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
```

```
 * 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 22/04/21 - node1.c.
 */
#include "node1.h"
#include <math.h>
#include <stdbool.h>
#include <FreeRTOS.h>
#include <FreeRTOSConfig.h>
#include <lock.h>
#include <event_groups.h>

// Private variable storing the SPEED lock
static Lock_t SPEED_sem = NULL;

// Private variable storing the DISTANCE lock
static Lock_t DISTANCE_sem = NULL;

// Private variable storing the SPEED flag
static EventGroupHandle_t SPEED_event = NULL;

// Private variable storing the DISTANCE flag
static EventGroupHandle_t DISTANCE_event = NULL;

// Private variable storing the speed itself.
static volatile int speed = 0;

// Private variable storing the distance itself.
static volatile float distance = .0F;


/**
 * @brief initializes the NODE1 protected object. This method must
 *        be called during the early beggining in order to be able
 *        to use the object's methods.
 *
 *        Notice that if there is no more memory available this
 *        call will block forever until board reboot (for safety
 *        reasons).
 */
void NODE1_init(void) {
    SPEED_sem = LOCK_create(NULL);
    DISTANCE_sem = LOCK_create(NULL);
    SPEED_event = xEventGroupCreate();
    DISTANCE_event = xEventGroupCreate();
}

/**
 * @brief Safely updates the stored speed with the new one.
```

```
64  *
65  * @param new_speed the new speed to set.
66  */
67 void SPEED_set(int new_speed) {
68     LOCK_acquire(SPEED_sem);
69     speed = new_speed;
70     LOCK_release(SPEED_sem);
71 }
72
73 /**
74  * @brief Safely gets the stored speed.
75  *
76  * @return int - the speed itself. -1 if any error occurs.
77  */
78 int SPEED_get(void) {
79     int current = -1;
80     LOCK_acquire(SPEED_sem);
81     current = speed;
82     LOCK_release(SPEED_sem);
83     return current;
84 }
85
86 /**
87  * @brief Updates the flag #SPEED_event indicating that a new
88  *        value for the speed has been received from CANBus
89  *        (see {can.c#CAN_Handle_IRQ}).
90  *
91  *        Notice that this method is intended to be called
92  *        from an interruption routine.
93  *
94  */
95 void SPEED_set_recv(void) {
96     configASSERT(SPEED_event ≠ NULL);
97     xEventGroupSetBitsFromISR(SPEED_event, BIT_SET, NULL);
98 }
99
100 /**
101  * @brief Waits until the SPEED flag has been set. Then, resets
102  *        the event itself so it can wait for it again.
103  *
104  */
105 void SPEED_wait_recv(void) {
106     configASSERT(SPEED_event ≠ NULL);
107     xEventGroupWaitBits(SPEED_event, BIT_SET, pdTRUE, pdFALSE, portMAX_DELAY);
108 }
109
110 /**
111  * @brief Safely updates the stored distance with the new one.
112  *
113  * @param new_distance the new distance to set.
114  */
115 void DISTANCE_set(float new_distance) {
116     LOCK_acquire(DISTANCE_sem);
117     distance = new_distance;
118     LOCK_release(DISTANCE_sem);
119 }
120
121 /**
```

```
122  * @brief Safely gets the stored distance.
123  *
124  * @return float - the stored distance. -1.0F if any error occurs.
125  */
126 float DISTANCE_get(void) {
127     float dist = -1.0F;
128     LOCK_acquire(DISTANCE_sem);
129     dist = distance;
130     LOCK_release(DISTANCE_sem);
131     return dist;
132 }
133
134 /**
135  * @brief Computes and returns if the stored distance is OK
136  *        based on the speed and the following equation:
137  *
138  *                            |  speed   |²
139  *                   dist <   |──────────|  · 0.5
140  *                            |    10    |
141  *
142  * @return true - if the current distance is geq than 50% of recommended.
143  * @return false - if the current distance is lower than 50% of recommended.
144  */
145 bool DISTANCE_get_security(void) {
146     float current_distance = DISTANCE_get();
147     int current_speed = SPEED_get();
148     return (current_distance < (.5F * ((float) pow((current_speed / 10), 2))));
149 }
150
151 /**
152  * @brief Updates the flag #DISTANCE_event indicating that a new
153  *        value for the distance has been received from CANBus
154  *        (see {can.c#CAN_Handle_IRQ}).
155  *
156  *        Notice that this method is intended to be called
157  *        from an interruption routine.
158  */
159 void DISTANCE_set_recv(void) {
160     configASSERT(DISTANCE_event ≠ NULL);
161     xEventGroupSetBitsFromISR(DISTANCE_event, BIT_SET, NULL);
162 }
163
164 /**
165  * @brief Waits until the DISTANCE flag has been set. Then, resets
166  *        the event itself so it can wait for it again.
167  */
168 void DISTANCE_wait_recv(void) {
169     configASSERT(DISTANCE_event ≠ NULL);
170     xEventGroupWaitBits(DISTANCE_event, BIT_SET, pdTRUE, pdFALSE, portMAX_DELAY)
    ↪ ;
171 }
```

Listing 34: Cuerpo del objeto protegido node1.