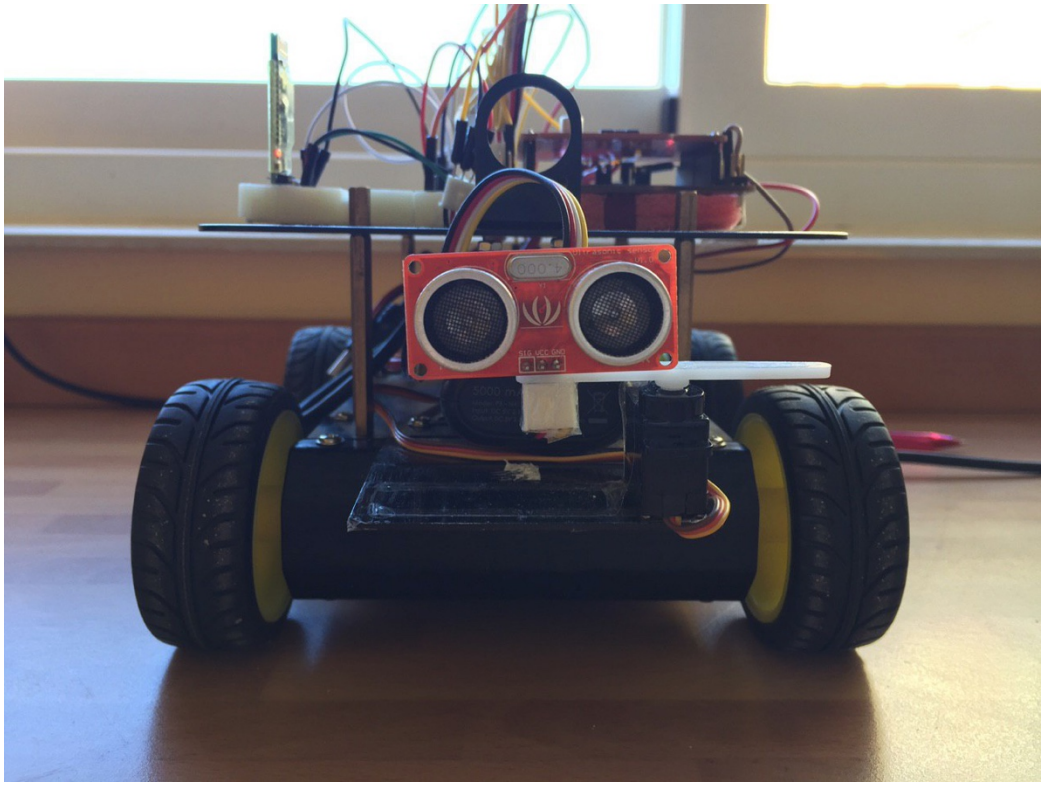
COCHE TELEDIRIGIDO POR BLUETOOTH

Diseño de Sistemas Basados en Microprocesador



MARIO PÉREZ SÁNCHEZ-MONTAÑEZ
DAVID CAMUÑAS SÁNCHEZ

Mario Pérez Sánchez-Montañez
David Camuñas Sánchez

ÍNDICE DE CONTENIDOS

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

INTRODUCCIÓN

El proyecto consiste en realizar un coche teledirigido por *bluetooth* con detección de obstáculos. El coche será controlado con un móvil *Android,* gracias a una aplicación llamada *Arduino bluetooth controller*[1].

Cuando se pulse el botón de alguna dirección el coche comprobará si no hay obstáculos, y si es así, empezará a moverse hacia esa dirección. Si en algún momento detecta un obstáculo se parará hasta que se reciba otra dirección o el obstáculo ya no esté. Para parar el coche se ha implementado el botón del **círculo.**

Cabe destacar, que cuando le mandemos la orden de dirigirse hacia atrás no se comprobará si hay obstáculos, ya que, el sensor ultrasónico esta situado en la parte delantera del coche y esa dirección no se puede comprobar.

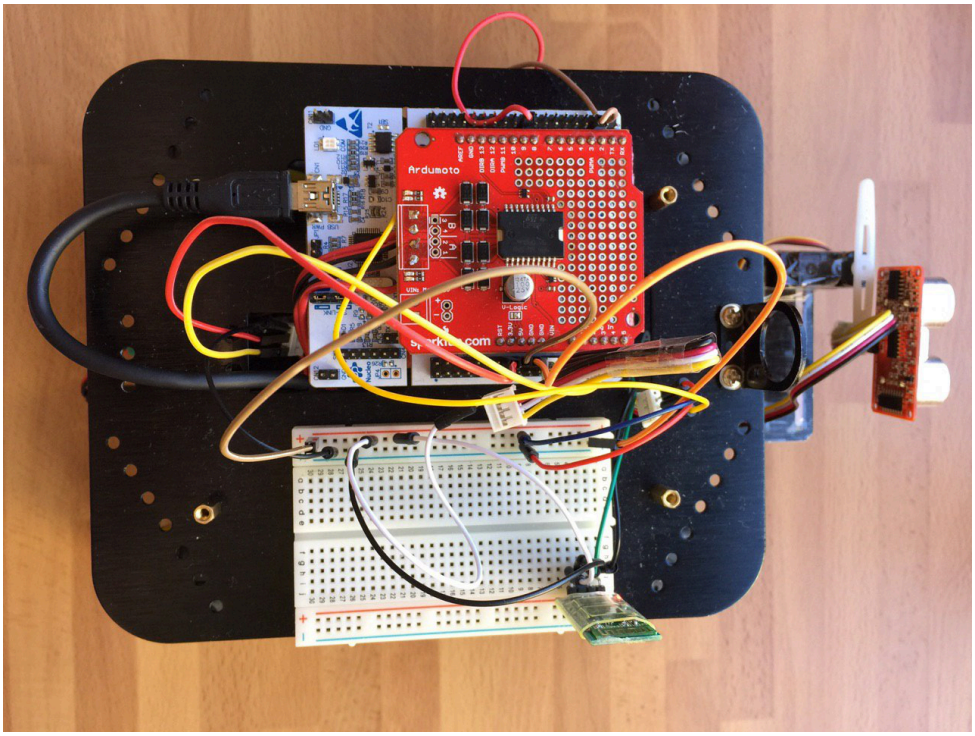


Ilustración 1: Diseño hardware implementado

[1] En la aplicación se han introducido los valores que se deben recibir para cada botón. Estos valores están declarados como constantes en el código del *main.c*

DISEÑO HARDWARE

CIRCUITO DISEÑADO

A continuación, se muestra un esquema del circuito implementado con la herramienta *fritzing*:

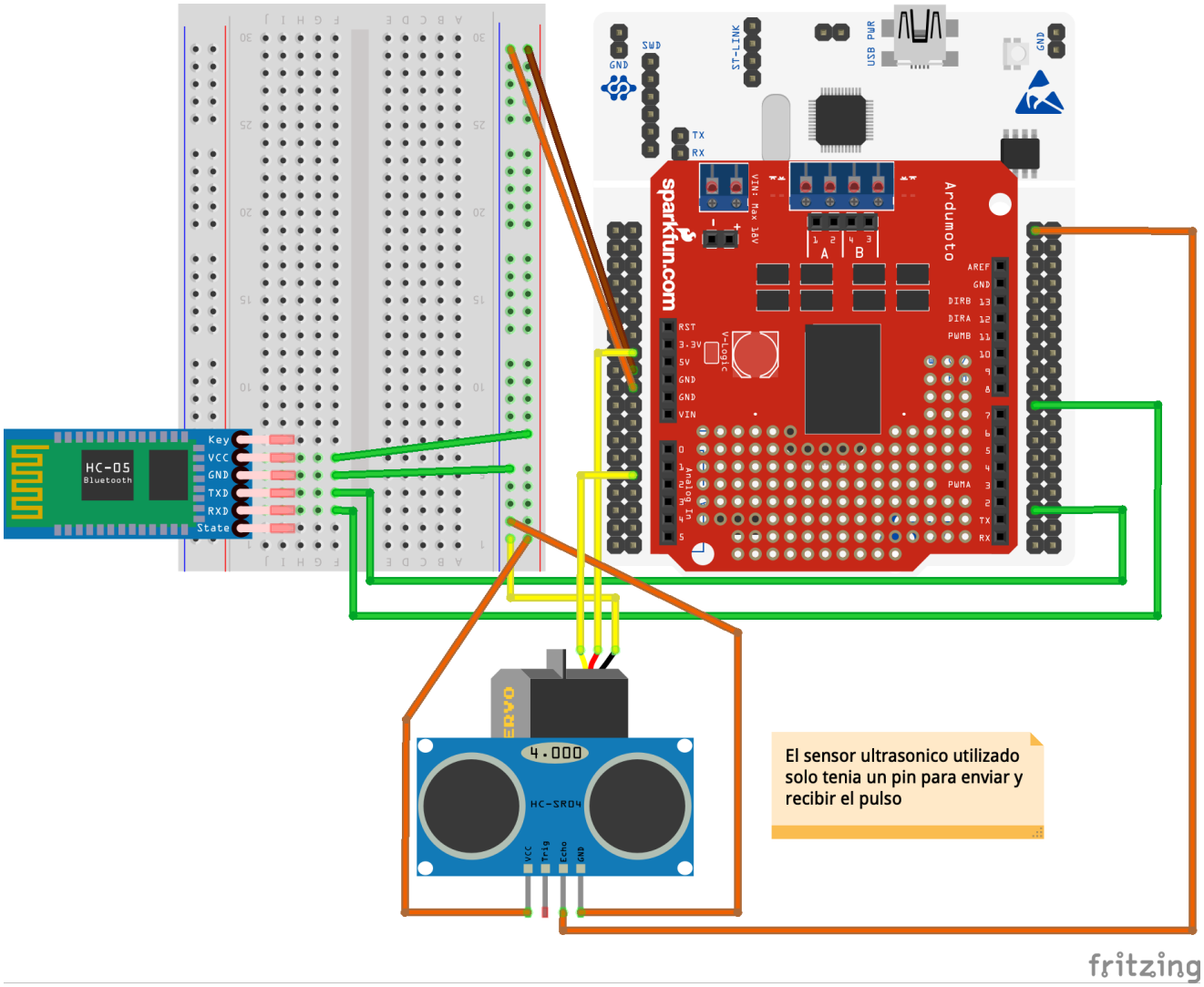


Ilustración 2: Esquema del circuito

CONFIGURACIÓN DE LOS PINES

Los pines elegidos para implementar el circuito se pueden apreciar en la siguiente tabla:

COMPONENTE	PIN
Ardumoto	PB3 (Timer 2), PA7 (Timer 3)
UART Bluetooth	PA10 (Rx), PA9 (Tx)
Servomotor del sensor ultrasónico	PA1 (Timer 5)
Sensor ultrasónico	PB8

Tabla 1: Configuración de los pines

[1]

En la siguiente imagen se puede apreciar la configuración de los pines realizada en el programa *STM32CubeMX*:

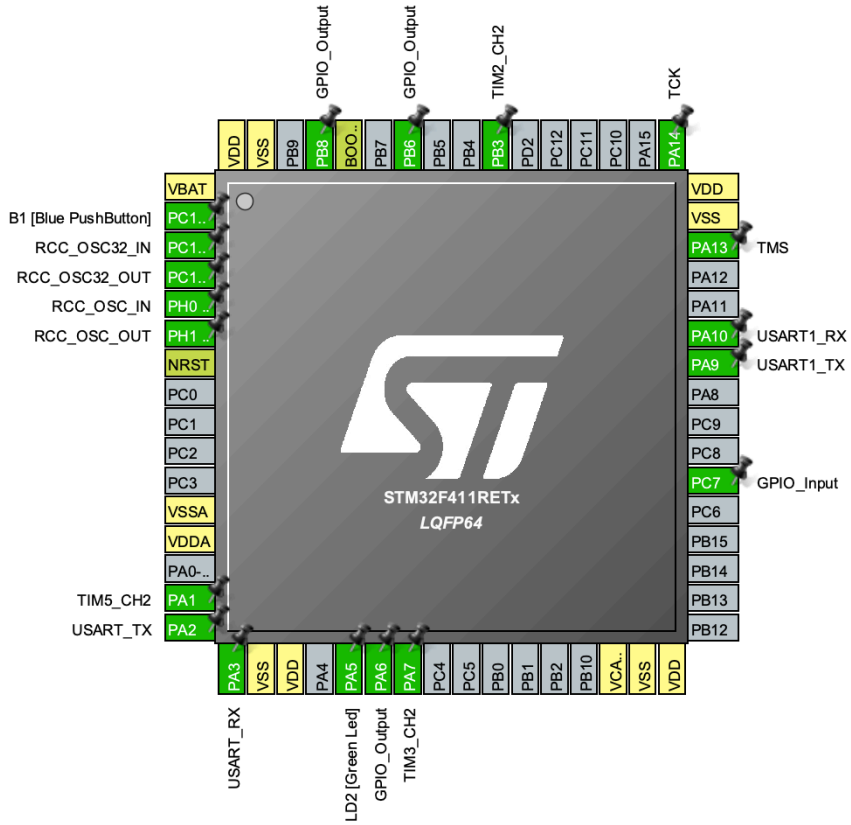


Ilustración 3: Configuración de los pines

DISEÑO SOFTWARE

CONFIGURACIÓN DE LOS TIMERS

Hemos utilizado un total de 4 *timers* en el proyecto, cada uno con objetivo específico, y son los siguientes:

TIMER	USO	PERIODO	PREESCALER
Tim2_CH2	Ardumoto	8499	1999
Tim3_CH2	Ardumoto	8499	1999
Tim5_CH2	Servomotor sensor ultrasónico	1679	999
Tim11	Calcular *ticks* sensor ultrasónico	839	1

Tabla 2: Timers

[2]

MÁQUINAS DE ESTADOS FINITOS (FSM)

Hemos implementado 2 máquinas de estados [3] en el proyecto, la justificación de cada una de ellas se aprecia a continuación:

- La *FSM* principal se encuentra en el bucle principal del *main.c,* la usamos para dependiendo la orden que se reciba el coche se mueva hacia una dirección u otra, y poder así pasar de un estado a otro dependiendo la elección. Cabe destacar, que solo se podrá acceder a esta máquina de estados si no se detecta un obstáculo

```
while (1)
  {
    HAL_UART_Receive_IT(&huart1, buffer_read, 1);
    ultrasonic();
    if (buffer_read[0] != BACK && dis <= NEAR)
      stop();
    else
    {
      switch (buffer_read[0])
      {
      case FRONT:
        move_front();
        break;
      case BACK:
        move_back();
        break;
      case RIGHT:
        move_right();
        break;
      case LEFT:
```

```
        move_left();
        break;
      case STOP:
        stop();
      }
    }
  }
```

- La segunda *FSM* puede parecer prescindible o se puede implementar en la principal, pero tiene objetivo concreto. Esta situada en el *callback Rx [4],* para que cada vez que se reciba una interrupción UART desde el sensor *bluetooth* el *servomotor* pueda moverse hacia la dirección recibida y así el sensor ultrasónico pueda detectar obstáculos.

  La justificación de poner esta segunda máquina de estados es porque si se ha detectado un obstáculo de frente por ejemplo, y el cambio de dirección del sensor ultrasónico se realiza en la FSM principal el servomotor no se moverá porque no podemos entrar en la máquina de estados, ya que, seguimos detectando el obstáculo. Esto incapacita a moverse en otra dirección distinta, exceptuando hacia atrás, porque, aunque mandemos otra dirección el sensor no se ha movido y seguimos detectando el objeto.

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{

  if (huart->Instance == USART1)
  {
    switch (buffer_read[0])
    {
    case FRONT:
    case BACK:
      move_check(MID_CHECK);
      break;
    case RIGHT:
      move_check(RIGHT_CHECK);
      break;
    case LEFT:
      move_check(LEFT_CHECK);
      break;
    }
  }
}
```

IMPLEMENTANDO EL MOVIMIENTO

Para producir que el coche se mueva debemos generar un pulso hacia los motores, además de indicar la dirección del giro. Para ello se han implementado una serie de funciones, cada una para producir el movimiento en una dirección específica. Además de las funciones para moverse, también se ha implementado otra función para parar el coche totalmente, que es trivial.

```
void move_front()
{

  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_SET);
  set_pwm_value(&htim2,MOVE_PULSE);
  set_pwm_value(&htim3,MOVE_PULSE);
}

void move_back()
{
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_RESET);
  set_pwm_value(&htim2, MOVE_PULSE);
  set_pwm_value(&htim3, MOVE_PULSE);
}

void move_right()
{

  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_SET);
  set_pwm_value(&htim2, MOVE_PULSE);
  set_pwm_value(&htim3, MOVE_PULSE);
}

void move_left()
{

  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_RESET);
  set_pwm_value(&htim2, MOVE_PULSE);
  set_pwm_value(&htim3, MOVE_PULSE);
}

void stop()
{
  set_pwm_value(&htim2, STOP_PULSE);
  set_pwm_value(&htim3, STOP_PULSE);
}
```

Como se puede observar para que los motores vayan hacia delante hay que establecer el pin correspondiente al motor a 1, y hacer lo contrario si se quiere ir hacia atrás. En el caso de los movimientos laterales, se debe establecer uno de los motores para que gire hacia delante y el otro hacia atrás, dependiendo cuál sea el movimiento.

En cuánto al pulso, hemos creado una función que recibe la estructura que, define el *timer* con el que se quiere trabajar, y el valor del pulso que se quiere establecer. [5]

```
void set_pwm_value(TIM_HandleTypeDef *htim,int value)
```

```
{
  TIM_OC_InitTypeDef sConfigOC;

  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = value;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  HAL_TIM_PWM_ConfigChannel(htim, &sConfigOC, TIM_CHANNEL_2);
  HAL_TIM_PWM_Start(htim, TIM_CHANNEL_2);
}
```

La estructura que define el *timer* se pasa por referencia para que se modifiquen los valores sobre los *timers* creados. Como todos los *timers* han sido implementados en el canal 2 pues esa macro podrá ser usada en cualquier caso.

Cabe destacar que esta función también la usamos en el caso del *servomotor,* para producir el movimiento correspondiente a la dirección deseada.

DETECCIÓN DE OBSTÁCULOS

La implementación que hemos realizado ha sido con un sensor ultrasónico y un *servomotor,* que se moverá en la dirección correspondiente. Como hemos dicho anteriormente el cambio dirección del sensor para detectar algún objeto se realiza en el *callback uart.* La detección de obstáculos se realiza calculando la distancia en cada iteración del bucle principal, dejando acceder a la máquina de estados principal cuando la distancia sea la adecuada. En el caso de que la orden recibida sea ir hacia atrás no se tendrá en cuenta la distancia calculada ya que el sensor no detecta obstáculos en esta dirección.

```
void move_check(int value)
{
  set_pwm_value(&htim5, value);
}
```

Como se puede observar, la función encargada de mover el sensor hacia la dirección deseada es trivial, simplemente hace una llamada a la función explicada anteriormente, que se encarga establecer un pulso en el motor que se quiera.

```
void ultrasonic()
{
  int t0, t1;
  /**** TRIGGER ****/
  definePinB8(OUT); //Pin a salida
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8, GPIO_PIN_RESET);
  HAL_Delay(50);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8, GPIO_PIN_SET); //Lanzar pulso
  ticks = 0;                                          //Reiniciar interrupciones timer
  while (ticks == 0)
    ;                                                   //Esperar interrupcion trigger
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8, GPIO_PIN_RESET); //Apagar pulso
```

```
  /**** ECHO ****/
  definePinB8(IN); //Pin a entrada
  while ((GPIOB->IDR & GPIO_IDR_ID8_Msk) == 0x00)
    ;         //Esperar un 1
  t0 = ticks; //Interrupciones antes de recibir 0
  while ((GPIOB->IDR & GPIO_IDR_ID8_Msk) != 0x00)
    ;         //Esperar un 0
  t1 = ticks; //Interrupciones despues de recibir 0
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8, GPIO_PIN_RESET);

  dis = (((t1 - t0) * 10) / 58) * 4; //Calcular distancia
}
```

En cuanto a la función encargada de calcular la distancia, se ha corregido la función utilizada en [6], según los fallos encontrados en la corrección. Para cambiar el pin utilizado para lanzar el pulso de entrada a salida o viceversa se ha utilizado la misma función que en [6].

```
void definePinB8(int mode)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  GPIO_InitStruct.Pin = GPIO_PIN_8;
  mode == OUT ? (GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP) : (GPIO_InitStruct.Mode =
GPIO_MODE_INPUT);
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  mode == OUT ? (GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH) : (GPIO_InitStruct.Speed =
GPIO_SPEED_FREQ_LOW);
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
}
```

VIDEO DEMOSTRACIÓN

El video de la demostración se ha subido a *Youtube* como se indicaba, el enlace es el siguiente:

https://www.youtube.com/watch?v=kM8rx6FPwG0

BIBLIOGRAFÍA Y REFERENCIAS

[1] JADIAZ, «iescamp,» [En línea]. Available: http://www.iescamp.es/miarduino/2016/02/22/ardumoto-shield/.
[2] frta, «B105,» [En línea]. Available: http://elb105.com/stm32f4-configurar-el-pwm-con-timers/.
[3] *Modulo 4: sw-basics,* 2019.
[4] *Modulo 5: interrupts,* 2019.
[5] *Lab 6: Adding Barrier for Cars (PWM),* 2019.
[6] *Lab 5: Car Detecting (Timers),* 2019.