**About Otto distance sensor and buzzer**

**J. Lemaire**

**(Pierrefeu September 2021)**

# Motivations

I recently discovered OttoDIY+ and was totally amazed by this little Robot, really brilliant by its simplicity and its possibilities, in particular by its approach using only 4 micro servos.

Having been a teacher in a computer environment in another lifetime (I'm now retired), I got caught up in the game and tried to understand how it moves and tried to simplify its Arduino code. I published in <https://github.com/jlemaire06/OttoMoves> a little Arduino library and a report about this work.

In this last work, I focused on the Otto movements. But Otto offers many other functionnalities, and specially those related to its sensors and expression tools ; so, here I consider[[1]](#footnote-2) :

* The **HC-SR04 ultrasonic distance sensor** ;
* The **piezoelctric buzzer**.

And the goal is : « how to add very simply these functionnalities to the previous library » ?

# Sonar and Buzzer classes

This goal seams easy to achieve, because there exist functions, in the Arduino IDE, to manage very simply an ultrasonic distance sensor (**pulseIn**) or a buzzer (**tone**). But unfortunately, using them classically leads to blocking treatments and, surprisingly, this appoach has been choosen in OttoDIYlib.

So we began to search other NON blocking interrupt based approaches ; they are described in the following short reports :

* « **Ultrasonic distance sensor** », which uses *Pin Change Interrupts*;
* « **Buzzer** », which uses *Timer2 interrupts*.

Each report propose in fine an optimized C++ class :

* **Sonar** (*Sonar.h* and *Sonar.cpp*) ;
* **Buzzer** (*Buzzer.h* and *Buzzer.cpp*).

**// Sonar.h**

#pragma once

#include "Arduino.h"

#include <elapsedMillis.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Class Sonar

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Class to manage an HC-SR04 sonar with periodic measures of the distance using

PCI (Pin Changed Interrupts) on the echo pin (need to be in [0, 7]).

\*/

extern "C" void PCINT2\_vect**();**

class Sonar

**{**

public**:**

Sonar**(**unsigned pinTrigger**,** unsigned pinEcho**,** unsigned int period**);** // Constructor (ms) for period

bool updateDistance**();**

double getDistance**();** // (cm)

private**:**

unsigned \_pinTrigger**;**

unsigned \_pinEcho**;**

unsigned \_period**;**

volatile unsigned long \_tauNew**,** \_tau**;**

elapsedMillis \_timer**;**

void funcISR**();**

friend void PCINT2\_vect**();** // To use private funcISR() in the ISR

**};**

**// Sonar.cpp**

#include "Sonar.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ISR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Sonar**\*** pSonar**;** // As static

ISR**(**PCINT2\_vect**)**

**{**

pSonar**->**funcISR**();**

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Class Sonar

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Sonar**::**Sonar**(**unsigned pinTrigger**,** unsigned pinEcho**,** unsigned int period**)**

**{**

pSonar **=** **this;**

\_pinTrigger **=** pinTrigger**;**

\_pinEcho **=** pinEcho**;**

\_period **=** period**;**

pinMode**(**\_pinTrigger**,** OUTPUT**);**

digitalWrite**(**\_pinTrigger**,** LOW**);**

pinMode**(**\_pinEcho**,** INPUT**);**

bitSet**(**PCMSK2**,** \_pinEcho**);** // Enable PCI for \_pinEcho

**}**

bool Sonar**::**updateDistance**()**

**{**

**if** **(**\_timer **>** \_period**)**

**{**

\_timer **=** 0**;**

// Ping

digitalWrite**(**\_pinTrigger**,** HIGH**);**

digitalWrite**(**\_pinTrigger**,** LOW**);**

// Enable PCI for Port2

bitSet**(**PCICR**,** PCIE2**);**

// return

**return** **true;**

**}**

**else** **return** **false;**

**}**

double Sonar**::**getDistance**()**

**{**

**return** **(**\_tau**/**58.82**);**

**}**

void Sonar**::**funcISR**()**

**{**

unsigned long t **=** micros**();**

**if** **(**digitalRead**(**\_pinEcho**))** \_tauNew **=** t**;**

**else**

**{**

\_tauNew **=** t **-** \_tauNew**;**

**if** **(**\_tauNew **>=** 50 **&&** \_tauNew **<=** 25000**)**

**{**

\_tau **=** \_tauNew**;**

bitClear**(**PCICR**,** PCIE2**);** // Stop PCI for Port2

**}**

**}**

**}**

**// Buzzer.h**

#pragma once

#include "Arduino.h"

#include "Pitches.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Class Buzzer

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Class to manage asynchonous tone on pin 11 using Timer2 CompA interrupts

and PWM phase-correct with top in OCRA

\*/

extern "C" void TIMER2\_COMPA\_vect**();**

struct Note

**{**

unsigned int freq**;**

unsigned int len**;**

**};**

class Buzzer

**{**

public**:**

Buzzer**();**

void tone**(**Note note**);**

void play**(**Note tNote**[],** unsigned int nbNote**,** unsigned int nbPlay**=**1**);**

bool isPlaying**()** const **{return** \_isPlaying**;}**

private**:**

unsigned int \_idNote**,** \_nbNote**;**

unsigned int \_idPlay**,** \_nbPlay**;**

bool \_okTone**;**

Note **\***\_tNote**;**

unsigned long \_endTime**;**

void funcISR**();**

bool \_isPlaying**;**

friend void TIMER2\_COMPA\_vect**();** // To use private funcISR() in the ISR

**};**

**// Buzzer.cpp**

#include "Buzzer.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ISR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Buzzer**\*** pBuzzer**;** // As static

ISR**(**TIMER2\_COMPA\_vect**)**

**{**

pBuzzer**->**funcISR**();**

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Class Buzzer

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Buzzer**::**Buzzer**()**

**{**

pBuzzer **=** **this;**

pinMode**(**11**,** OUTPUT**);**

\_isPlaying **=** **false;**

// WGM2 = 0b101 => Phase-correct PWM with TOP in OCR2A

TCCR2A **=** **(**1**<<**WGM20**);**

TCCR2B **=** **(**1**<<**WGM22**);**

**}**

void Buzzer**::**tone**(**Note note**)**

**{**

// End time

**if** **(**note**.**len **>** 0**)** \_endTime **=** millis**()** **+** note**.**len**;**

**else** \_endTime **=** 0xFFFFFFFF**;**

unsigned int freq **=** note**.**freq**;**

**if** **(**freq **>** 0**)**

**{**

// CS2 and OCR2A

unsigned CS2**;**

unsigned int fact**;**

**if** **(**freq **<** 60**)** **{**CS2 **=** 7**;** fact **=** 3906**;** OCR2A **=** fact**/**freq**;}** // Prescaler = 1024

**else** **if** **(**freq **<** 120**)** **{**CS2 **=** 6**;** fact **=** 15625**;** OCR2A **=** fact**/**freq**;}** // Prescaler = 256

**else** **if** **(**freq **<** 240**)** **{**CS2 **=** 5**;** fact **=** 31250**;** OCR2A **=** fact**/**freq**;}** // Prescaler = 128

**else** **if** **(**freq **<** 480**)** **{**CS2 **=** 4**;** fact **=** 62500**;** OCR2A **=** fact**/**freq**;}** // Prescaler = 64

**else** **{**CS2 **=** 4**;** fact **=** 62500**;** OCR2A **=** round**(**fact**/**freq**);}** // Prescaler = 64

// Enable PWM output on pin 11 with 50% duty cycle (Toggle OC2A on compare match)

bitSet**(**TCCR2A**,** COM2A0**);**

// Set prescaler using CS2

TCCR2B **&=** 0xF8**;** // Clear CS2 bits

TCCR2B **|=** CS2**;** // Set CS2 bits

// Enable COMPA interrupt

TIMSK2 **=** **(**1**<<**OCIE2A**);**

// Tone flag

\_okTone **=** **true;**

**}**

**}**

void Buzzer**::**play**(**Note tNote**[],** unsigned int nbNote**,** unsigned int nbPlay**=**1**)**

**{**

\_tNote **=** tNote**;**

\_idNote **=** 0**;**

\_nbNote **=** nbNote**;**

\_idPlay **=** 0**;**

\_nbPlay **=** nbPlay**;**

**if** **(**\_nbNote **>** 0**)**

**{**

\_isPlaying **=** **true;**

tone**(**\_tNote**[**0**]);**

**}**

**}**

void Buzzer**::**funcISR**()**

**{**

**if** **(**millis**()>=**\_endTime**)**

**{**

**if** **(**\_okTone**)**

**{**

bitClear**(**TCCR2A**,** COM2A0**);** // Stop PWM output on pin 11 (but keep COMPA interrupt active)

\_okTone **=** **false;** // Tone flag

\_endTime **+=** 50**;** // 50ms delay after tone

**if** **(**\_idNote **==** \_nbNote**-**1**)** \_endTime **+=** 200**;**

**}**

**else**

**{**

\_idNote**++;** // Next note

**if** **(**\_idNote **==** \_nbNote**)**

**{**

\_idNote **=** 0**;**

\_idPlay**++;**

**}**

**if** **(**\_idPlay **<** \_nbPlay**)** tone**(**\_tNote**[**\_idNote**]);** // Tone call

**else**

**{**

TIMSK2 **=** 0**;** // Stop COMPA interrupt

\_isPlaying **=** **false;**

**}**

**}**

**}**

**}**

**// Pitches.h**

#define NOTE\_B0 31

#define NOTE\_C1 33

#define NOTE\_CS1 35

#define NOTE\_D1 37

#define NOTE\_DS1 39

#define NOTE\_E1 41

#define NOTE\_F1 44

#define NOTE\_FS1 46

#define NOTE\_G1 49

#define NOTE\_GS1 52

#define NOTE\_A1 55

#define NOTE\_AS1 58

#define NOTE\_B1 62

#define NOTE\_C2 65

#define NOTE\_CS2 69

#define NOTE\_D2 73

#define NOTE\_DS2 78

#define NOTE\_E2 82

#define NOTE\_F2 87

#define NOTE\_FS2 93

#define NOTE\_G2 98

#define NOTE\_GS2 104

#define NOTE\_A2 110

#define NOTE\_AS2 117

#define NOTE\_B2 123

#define NOTE\_C3 131

#define NOTE\_CS3 139

#define NOTE\_D3 147

#define NOTE\_DS3 156

#define NOTE\_E3 165

#define NOTE\_F3 175

#define NOTE\_FS3 185

#define NOTE\_G3 196

#define NOTE\_GS3 208

#define NOTE\_A3 220

#define NOTE\_AS3 233

#define NOTE\_B3 247

#define NOTE\_C4 262

#define NOTE\_CS4 277

#define NOTE\_D4 294

#define NOTE\_DS4 311

#define NOTE\_E4 330

#define NOTE\_F4 349

#define NOTE\_FS4 370

#define NOTE\_G4 392

#define NOTE\_GS4 415

#define NOTE\_A4 440

#define NOTE\_AS4 466

#define NOTE\_B4 494

#define NOTE\_C5 523

#define NOTE\_CS5 554

#define NOTE\_D5 587

#define NOTE\_DS5 622

#define NOTE\_E5 659

#define NOTE\_F5 698

#define NOTE\_FS5 740

#define NOTE\_G5 784

#define NOTE\_GS5 831

#define NOTE\_A5 880

#define NOTE\_AS5 932

#define NOTE\_B5 988

#define NOTE\_C6 1047

#define NOTE\_CS6 1109

#define NOTE\_D6 1175

#define NOTE\_DS6 1245

#define NOTE\_E6 1319

#define NOTE\_F6 1397

#define NOTE\_FS6 1480

#define NOTE\_G6 1568

#define NOTE\_GS6 1661

#define NOTE\_A6 1760

#define NOTE\_AS6 1865

#define NOTE\_B6 1976

#define NOTE\_C7 2093

#define NOTE\_CS7 2217

#define NOTE\_D7 2349

#define NOTE\_DS7 2489

#define NOTE\_E7 2637

#define NOTE\_F7 2794

#define NOTE\_FS7 2960

#define NOTE\_G7 3136

#define NOTE\_GS7 3322

#define NOTE\_A7 3520

#define NOTE\_AS7 3729

#define NOTE\_B7 3951

#define NOTE\_C8 4186

#define NOTE\_CS8 4435

#define NOTE\_D8 4699

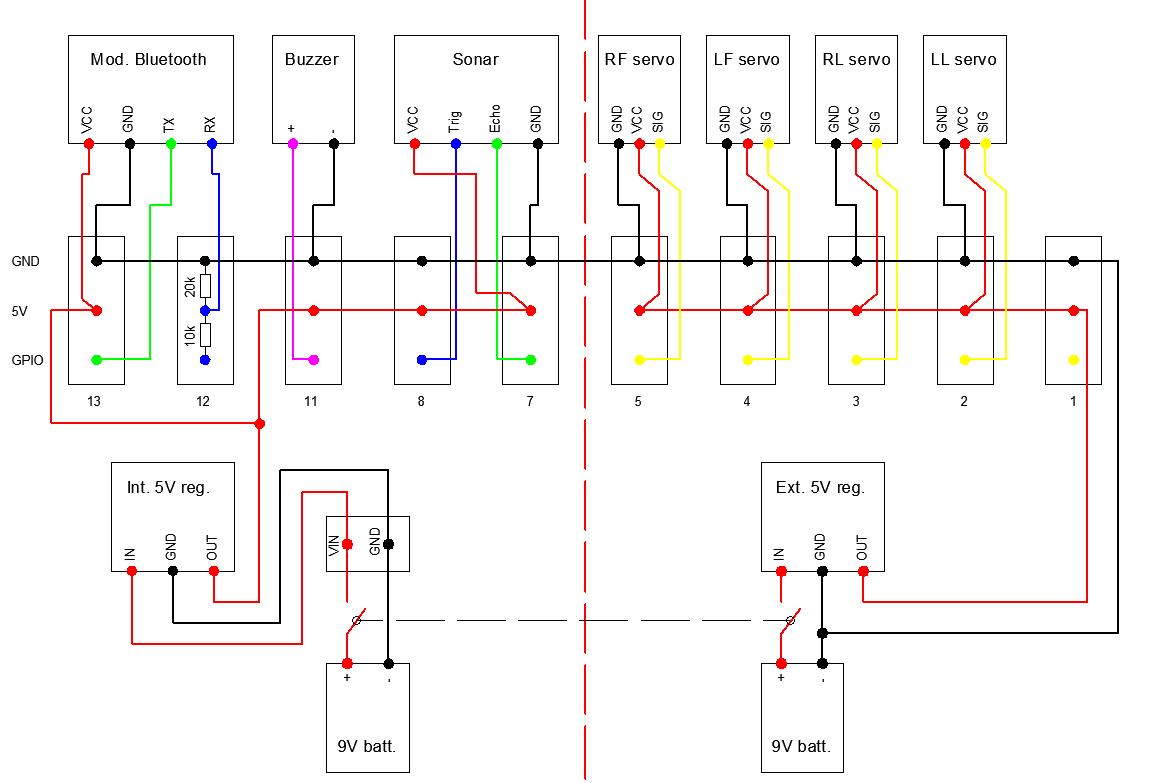
#define NOTE\_DS8 4978

**Remarks**

* These two classes can instance only one object because the ISR request static variables (via a static pointer here). But, as it can be observed in the two sketchs, they encapsulate completely the interrupt staff and the enable a great simplicity.
* The echo pin for the sonar and the positive pin for the buzzer are restricted : as precised in the previous documents, the fisrt need to be in Port2 [0, 7], because of PCI treatment, and the second equal to 11, because it corresponds to the PWM output pin of timer2, when OCR2A is used to reduce the timer cycle.

# Pin reassignment in Otto

Because of the pin restrictions for the sonar and the buzzer, a little reassignment is necessary for the Otto’s pins. The following schematic precises it with the power improvments, using a specific 9V battery for the servos and a resistance bridge for the RX pin of the Bluetooth module, to reduce the Nano output logic level from 5V to 3.3V[[2]](#footnote-3).



# The RobotSCS class

Now, to add these functionnalities to Otto, just need to define a new class, RobotSCS class, which inherits from the RobotSC, Sonar and Buzzer classes, with a processSonar method.

Here, to illustrate the process, we only consider a permanent distance detection, with alarm sounds when thresholds are underpassed. But more complex implementations are possible : to stop the robot, to follow a limit, etc.

**// RobotSCS.h**

#pragma once

#include "RobotSC.h"

#include "Sonar.h"

#include "Buzzer.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Class RobotSCS

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Add a sonar and a buzzer to RobotSC

\*/

class RobotSCS **:** public RobotSC**,** public Sonar**,** public Buzzer

**{**

public**:**

RobotSCS**(**int pinLL**,** int pinRL**,** int pinLF**,** int pinRF**,** Stream**&** serial**,** int pinT**,** int pinE**,** bool calibrated **=** **true);**

void processSonar**();**

**};**

**// RobotSCS.cpp**

#include "RobotSCS.h"

Note alm1**[]** **=** **{{**NOTE\_B4**,** 250**},** **{**NOTE\_G3**,** 250**}};**

Note alm2**[]** **=** **{{**NOTE\_B4**,** 125**},** **{**NOTE\_G3**,** 125**}};**

#define DIST1 15

#define DIST2 10

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Class RobotSCS

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

RobotSCS**::**RobotSCS**(**int pinLL**,** int pinRL**,** int pinLF**,** int pinRF**,** Stream**&** serial**,** int pinT**,** int pinE**,** bool calibrated **=** **true)**

**:** RobotSC**(**pinLL**,** pinRL**,** pinLF**,** pinRF**,** serial**,** calibrated**),** Sonar**(**pinT**,** pinE**,** 20**),** Buzzer**()**

**{**

**}**

void RobotSCS**::**processSonar**()**

**{**

**if** **(**updateDistance**()** **&&** **!**isPlaying**())**

**{**

double dist **=** getDistance**();**

**if** **(**dist **<** DIST2**)** play**(**alm2**,** 2**);**

**else** **if** **(**dist **<** DIST1**)** play**(**alm1**,** 2**);**

**}**

**}**

Then, to command the robot using the serial port is very simple :

**// TestRobotSCS.ino**

#include <RobotSCS.h>

RobotSCS r**(**2**,** 3**,** 4**,** 5**,** Serial**,** 8**,** 7**);**

void setup**()**

**{**

Serial**.**begin**(**9600**);**

delay**(**100**);**

**}**

void loop**()**

**{**

r**.**processSonar**();**

r**.**processCommand**();**

r**.**update**();**

**}**

For the Bluetooth version :

**// TestRobotBTS.ino**

#include <RobotSCS.h>

#include <SoftwareSerial.h>

SoftwareSerial BTSerial**(**13**,**12**);**

RobotSCS r**(**2**,** 3**,** 4**,** 5**,** BTSerial**,** 8**,** 7**);**

void setup**()**

**{**

BTSerial**.**begin**(**9600**);**

delay**(**100**);**

**}**

void loop**()**

**{**

r**.**processSonar**();**

r**.**processCommand**();**

r**.**update**();**

**}**

But this last sketch produces an error at the compilation because the **SoftwareSerial** library defines the **PCINT2\_vect** ISR in the **SoftwareSerial.cpp[[3]](#footnote-4)** file (lig. 227-244) :

#if defined(PCINT0\_vect)

ISR**(**PCINT0\_vect**)**

**{**

SoftwareSerial**::**handle\_interrupt**();**

**}**

#endif

#if defined(PCINT1\_vect)

ISR**(**PCINT1\_vect**,** ISR\_ALIASOF**(**PCINT0\_vect**));**

#endif

#if defined(PCINT2\_vect)

ISR**(**PCINT2\_vect**,** ISR\_ALIASOF**(**PCINT0\_vect**));**

#endif

#if defined(PCINT3\_vect)

ISR**(**PCINT3\_vect**,** ISR\_ALIASOF**(**PCINT0\_vect**));**

#endif

To solve the problem, one can use another library, adding a serial port, for example **AltSoftSerial**[[4]](#footnote-5) ; but it requires to change again the connections and to uses pins 9 and 8 for the Bluetooth module. A more simple solution consists in undefining the **PCINT2\_vect** in the **SoftwareSerial** library : this is possible because only the Port0 pins 12 and 13 are used here, for a serial communication.

Hence, we just have added the instruction :

#undef PCINT2\_vect

at the begining of the **SoftwareSerial.h** and copied the 2 files in the Robot directory.

1. Here the microphone and the touch sensor are not considered because we just want to illustrate how to complete our library with such tools. [↑](#footnote-ref-2)
2. Cf. « How to command Otto with the voice ». [↑](#footnote-ref-3)
3. Extracted from C:\Program Files (x86)\Arduino\hardware\arduino\avr\libraries\SoftwareSerial\src [↑](#footnote-ref-4)
4. <https://www.pjrc.com/teensy/td_libs_AltSoftSerial.html> [↑](#footnote-ref-5)