POLITEHNICA University of Bucharest
Faculty of Electronics, Telecommunications and Information Technology

Smart cars for safer traffic

- *The car that doesn't let you drink & drive* -

Diploma thesis

Submitted in partial fulfillment of the requirements for the *Degree of Engineer* in the domain *Electronics, Telecommunications and Information Technology*, study program *Applied Electronics*

Thesis Advisor
Prof. PhD. Corneliu BURILEANU

Student
Loredan Emilio BUCUR

2016

Table of contents

List of figures

List of acronyms

ADAS - advanced driver assistance systems

ADC - analog to digital convertor

BAC - blood alcohol concentration

DEFC - direct-ethanol fuel cell

DUI - driving under the influence

ECU - electronic control unit

EEPROM - electrically erasable programmable read-only memory

FLIR - forward looking infrared

GPS - global positioning system

GSM - global system for mobile communications

HW - hardware

IID - ignition interlock device

IR - infrared

LUT - look up table

MCU - microcontroller unit

PC - personal computer

SMS - short message service

SRAM - static random access memory

USART - universal synchronous/asynchronous receiver/transmitter

USB - universal serial bus

Introduction

Alcohol is a very popular substance used by people in our society to help them get some fun more easily by freeing themselves from the day-to-day stress and also from all kinds of inhibition. Although we have an age limit for its consumption, this also corresponds to the age that one is allowed to drive (with a driver license, of course). So one may experience an increasing temptation to combine the two in a potentially fatal distraction.

Besides offering proper education to our kids, we can protect them with something that is there to think for them when they are unable to do so because of the consumption of alcohol. This is the kind of device that I chose to build and analyse for my diploma thesis: a car breathalyser which generates an alert when you are drunk and should not drive. Not only this, but it also sends an SMS to a trusted contact with the location of the person that is at risk.

Motivation

The sole purpose of this work is reducing the number of car accidents, thus reducing the chances that one gets injured or dies during a road transit. The first thing that had to be done in order to accomplish this mission was the identification of the problem that generated car crashes. After a little bit of research [1] I ended up with a list of top causes of traffic incidents; this was a thing to start from:

1. Fatigue driving
   This usually happens out of cities, on straight roads, where the traffic is monotonous and so it doesn't require much attention. Fortunately, most of the crashes end up in trees or in moats on the roadside so there is a small number of victims.

2. Driving under the influence of alcohol
   Although the penalty for DUI is severe and most of the time the consequences are serious, some persons are that irresponsible and get behind the wheel after a drink or two - they may think that their BAC level is low enough because it's pretty hard to estimate how drunk you really are. That's the whole point of a breathalyser; it provides you with an unbiased feedback. What comes next is easy to predict: one tries to execute a task (i.e. driving) that requires much attention and fast reflexes, but the mind & body cannot operate at full capacity even under the smallest quantity of alcohol. So yes, even after only a small glass of wine the probability of ending up in a tree rises dreadful.

3. Speed
   Speed is ruthless. It always produces the highest number of critical injuries. They could be avoided if people would adapt speed to current weather conditions or their response time. Sadly, there will always be speeding drivers on the roads (even after the adoption of autonomous cars, I'm sure they'll provide a manual operating mode which will be used for fooling the law).

4. Violation of traffic regulations
   Running the red light is plainly stupid, and not granting priority should be treated with the same zero-tolerance as well. The exact situation may get various forms, but the outcome is fatal for the pedestrians, if they have the misfortune to be on the crosswalk at that time.

Objectives

As you can see, the top causes of accidents are related to human error, and not technological faults. Of course, we can't improve our vision capabilities nor even our response time, but what about behaviour?

The simple answer is yes we can. Then why won't more people adopt a cautious conduct so we could enjoy our rides without fearing our dear ones could not make it home? Well, while it is easy to tell the right thing to do, the harder part we encounter when it comes to doing what is right. That way, it may seem proper to drive home anytime even if we are fatigued, it may seem ordinary to ignore the speed limit when we are in a hurry, and it may seem exculpatory to run that traffic light if no one's coming anyway.

These cases present a problem of education which should be applied by all the participants in traffic and cannot be resolved by technology. However, what about the moments when our minds are not thinking straight, and any trace of social compliance is forgotten? What about drunk people who can't tell their right from left but they want to go home? Who should tell them they're not allowed to drive when they're alone with their car and the keys in the pocket? The answer is clear: the car should be able to know and tell if the occupant of the driver's seat is eligible for this task or not.

What I finally accomplished in this work is the practical implementation of this need: enabling cars with the ability to detect drunk drivers and report them. In the next chapters I'll discuss how I achieved this and the results I obtained.

Alternatives

Before starting to build anything, everyone should take a look at the actual alternatives present on the market. Maybe someone is already selling something similar so there is no point in reinventing the wheel, only if you add certain features to that product. So I searched for existing breathalysers and I found that despite the fact that this concept is not new to the automotive industry, there are still a lot of problems to be solved and features to be added.

First of all, there are the so-called ignition interlocks. All states in U.S.A. give permission to the judge to enforce the installation of such a device in cars owned by people who were convicted of DUI. They are breathalyzers which break the ignition circuit if the measured BAC is above a predefined threshold. They usually require the driver to do a test before starting the engine, but there are some which require periodical check-ups while driving too.

Ignition interlocks work very well because they rely on a more expensive type of alcohol sensor than MQ-3, which is actually a direct-ethanol fuel cell (DEFC). A fuel cell uses the input ethanol as a combustible and transforms it into heat and generated current. This current is then intercepted, measured and converted into BAC values.

The difference between these two concepts is that the interlock device is something that is mandatory to install if you were doing DUI, and the other one is just a measure of security initiated by the car's owner for the care of himself or other dear ones that may have that kind of problems. Moreover, the breathalyser proposed in this work can also alert a trusted contact giving the coordinates of the car using a GSM network.

Another problem with ignition interlock devices is that they take for granted that the driver blows into the input hole, and not an electric pump or just a sober friend. It is a real security bug for them because they can be easily fooled (and I have personally seen a video presentation in which a guy fooled such a device with an electric pump). By comparison, the open design of my breathalyser continuously scans the air in the region of the driver for exhaled alcohol vapours. This way no one's required to blow and no one can change seats after verification because this is a continuous process!

I think it's fair to compare a few prices though, and I must say that present breathalysers existing on the market cost somewhere between 80 € - 200 €. By comparison, I've spent ~30 € for my prototype, not including the GSM and GPS module because they are very expensive, but neither are they available on the competing products. Overall, the whole prototype costed me 150 €, a fair price I might say.

1. Theoretical notions

My work is mainly a practical implementation of a safety car gadget, but since it is based on some important theoretical notions one should understand them first in order to get a grasp of how the device is working (and why).

The chemistry behind

Alcohol, as defined by Wikipedia [2] is "any organic compound in which the hydroxyl functional group (–OH) is bound to a saturated carbon atom. The term alcohol originally referred to the primary *alcohol ethanol* (ethyl alcohol), the predominant alcohol in alcoholic beverages".

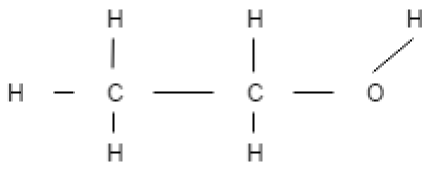


Fig. 1.1 Chemical form of ethanol

Let's see how this relates to ethanol, which is "a volatile, flammable, colorless liquid with a slight chemical odor. It is used as an antiseptic, a solvent, a fuel, and due to its low freezing point, the active fluid in many alcohol thermometers." [3]. The molecule has the following representation:

$$CH_3CH_2OH \tag{1}$$

Therefrom, ethanol is a particular type of alcohol present in alcoholic beverages. This makes it our target gas, the one we should search for in order to determine the sobriety of a person.

Methods used for measuring alcohol concentration

- We measure alcohol concentration by analysing different biological samples like blood, vapours in breath, urine, saliva and perspiration [4]. Each one has its characteristics and applications:

- Blood analysis is the most accurate one, but also the most difficult to perform because of technology, location or subject's diseases

- Measuring BAC from perspiration is a long-term process but has the advantage of continuous testing. This way, people being taken in the view of law enforcers can be monitored 24/7 with some alcohol drug patches. This method is the least accurate because of the subtle quantity of alcohol that ends in human perspiration (most of it is present in blood). In the future, maybe the wheel of the car could extract this information from the hands of the drivers, alongside the pulse and other thermal data. The pulse if useful for the ECU to determine if the driver has drowsiness problems and should take a break.

- Lately, IR spectroscopy devices were used to detect ethanol vapours in a chamber through which a person must exhale air. They are based on the fact that ethanol molecules absorb certain light wavelengths, which don't arrive at the receiver endpoint.

- Another technique based on infrared radiation is thermal screening. Basically, pictures of people's faces are taken in the IR dimension [6], and their blood vessel characteristics are analysed forwarded to a neural network [7] in order to obtain a classification.

- Urine testing can be used when other types are not available or supported by the patient. However, it has low precision results.

- Breath analysis is the most common one, being used by police officers as a first rank classifier of drivers. Their devices are called breathalysers and make use of ethanol cells (DEFC's) which transform the chemical energy from fuels like benzene and alcohol into water and electrical current. Figure 1.2 explains more clearly the way that fuel cells work. This current is then measured, and the BAC can be calculated easily with the help of an LUT. An interesting fact is that new concept cars that run on hydrogen are using fuel cells to convert it into electricity. The energy resulted is then used to power an electric motor.

Contrary to the popular breathalysers, my device uses sensors that change their conductivity in the presence of alcohol vapours. After that, a voltage divider circuit allows an Arduino to read the voltage across a resistor to determine the amount of alcohol someone exhales. It is notable the fact that instead of requesting a constant stream of air to be exhaled by the person in cause, my device scans for alcohol vapours in the air surrounding it. This has both advantages and disadvantages, so one should choose the appropriate tool for their needs. The positive thing would be that in case of automotive applications, the breathalyser can continuously scan for alcohol, so the driver cannot fool the sensors by asking a sober friend to blow into the breathalyser. At the moment alcohol reaches my device's sensors, an alarm is triggered and sent over the GSM network to the person responsible for the car or the driver. However, there is a huge negative side which must be taken into consideration in law enforcement applications: because of variable factors such as the distance between the driver and the wheel, the way he blow: on the nose or the mouth, what was eaten after the consumption of alcohol, etc the exact value of BAC cannot be calculated. So these types of setups are only suitable for drunk driver detection, and not for BAC estimations. There are other factors which affect the results of the MQ-3 sensors too, but they can be overcome:

- We can install a weight/force sensors in the driver's seat in order to determine its weight
- We can add a temperature / humidity sensor to the device because MQ-3 output varies with the environmental parameters like in Figure 1.4

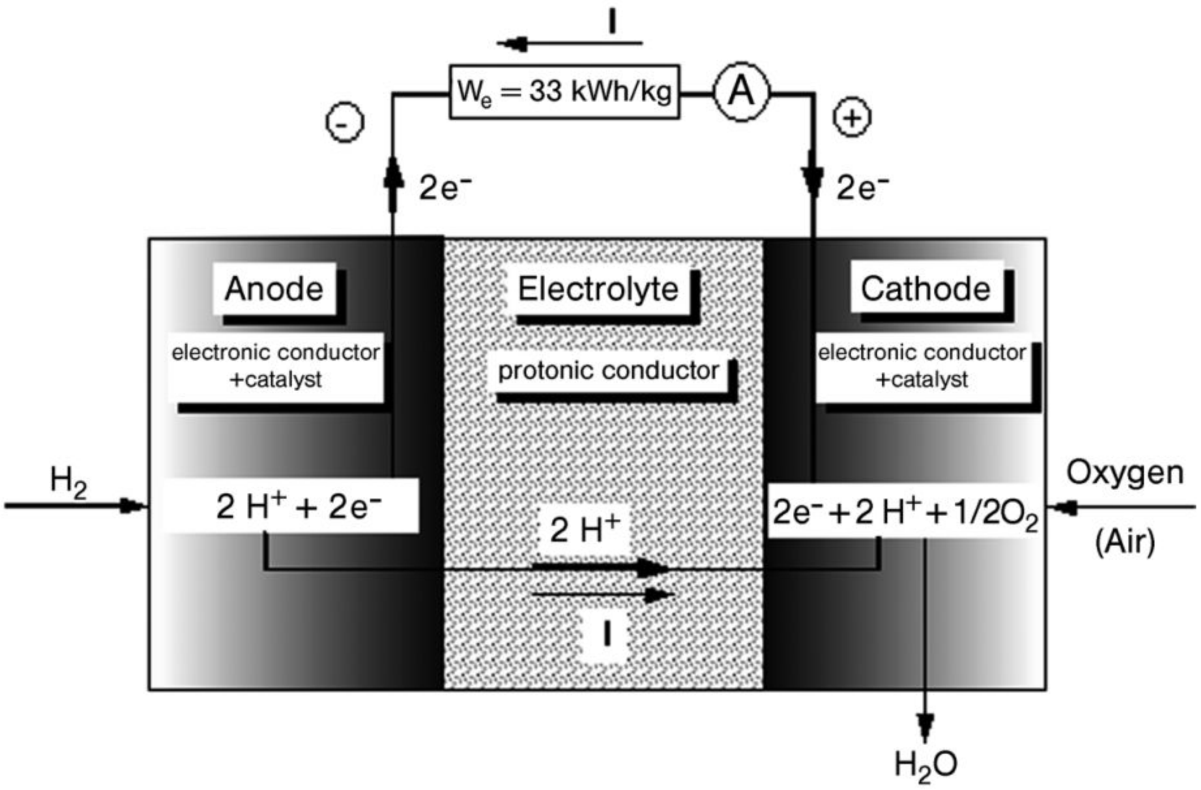


Figure 1.2 An ethanol cell
Source: [3]

Getting to know the hardware

ATmega328P

- ➔ Is an 8-bit microcontroller
- ➔ Needs an external USB-to-Serial converter for communication with a PC
- ➔ Advanced RISC architecture (AVR)
- ➔ Clocked at 8 MHz - because it's on the 3.3V Arduino
- ➔ 32 KB Flash program memory - filled up to 72%
- ➔ 2 KB SRAM - used at 75% for global variables only
- ➔ 1 KB EEPROM
- ➔ 8-channel 10-bit ADC - used to read analog output from two MQ-3 sensors and 3-channel (one for each axis) output from the accelerometer
- ➔ 1 USART - used for computer debugging
- ➔ 1 16-bit timer - used for emulating a software serial port

MQ-3

- ➔ Gas sensor; most sensible to Ethanol
- ➔ 0.5 W as measured
- ➔ Concentration range: 0.05 - 10 [mg/l]
- ➔ Internal resistance drops in contact with alcohol
- ➔ Ideal conditions:
  - ◆ 20 Celsius degrees
  - ◆ 65% humidity in air
  - ◆ 21% oxygen concentration

	Parts	Materials
1	Gas sensing layer	SnO_2
2	Electrode	Au
3	Electrode line	Pt
4	Heater coil	Ni-Cr alloy
5	Tubular ceramic	Al_2O_3
6	Anti-explosion network	Stainless steel gauze (SUS316 100-mesh)
7	Clamp ring	Copper plating Ni
8	Resin base	Bakelite
9	Tube Pin	Copper plating Ni

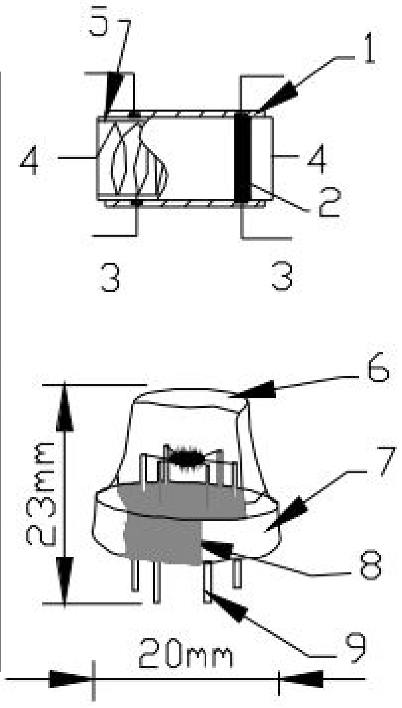


Fig 1.3 Structure of MQ-3 sensor
Source: Hanwei Electronics, LTD

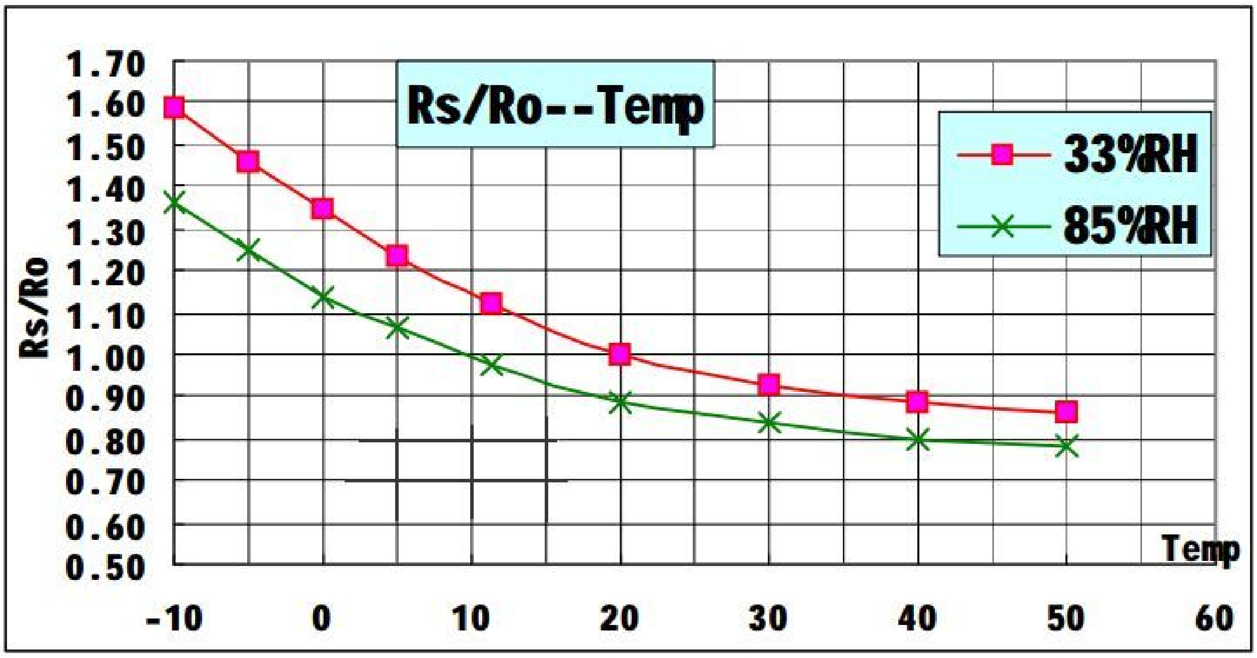


Figure 1.4 MQ-3 output variation with temperature and humidity
Source: Hanwei Electronics, LTD

2. Implementation

I will try to describe in a short paragraph the functionality of the device, and then I will add a few schemes for exemplification. So a possible scenario would look like this: the breathalyser is mounted on a car; when a drunk person enters, the sensors report a sudden rise of ethanol concentration in air and the logical unit (Arduino) takes action. First, it reads the geographical coordinates from the GPS module and then sends a warning to the car's owner via SMS.

A rough implementation of the prototype looks like this:

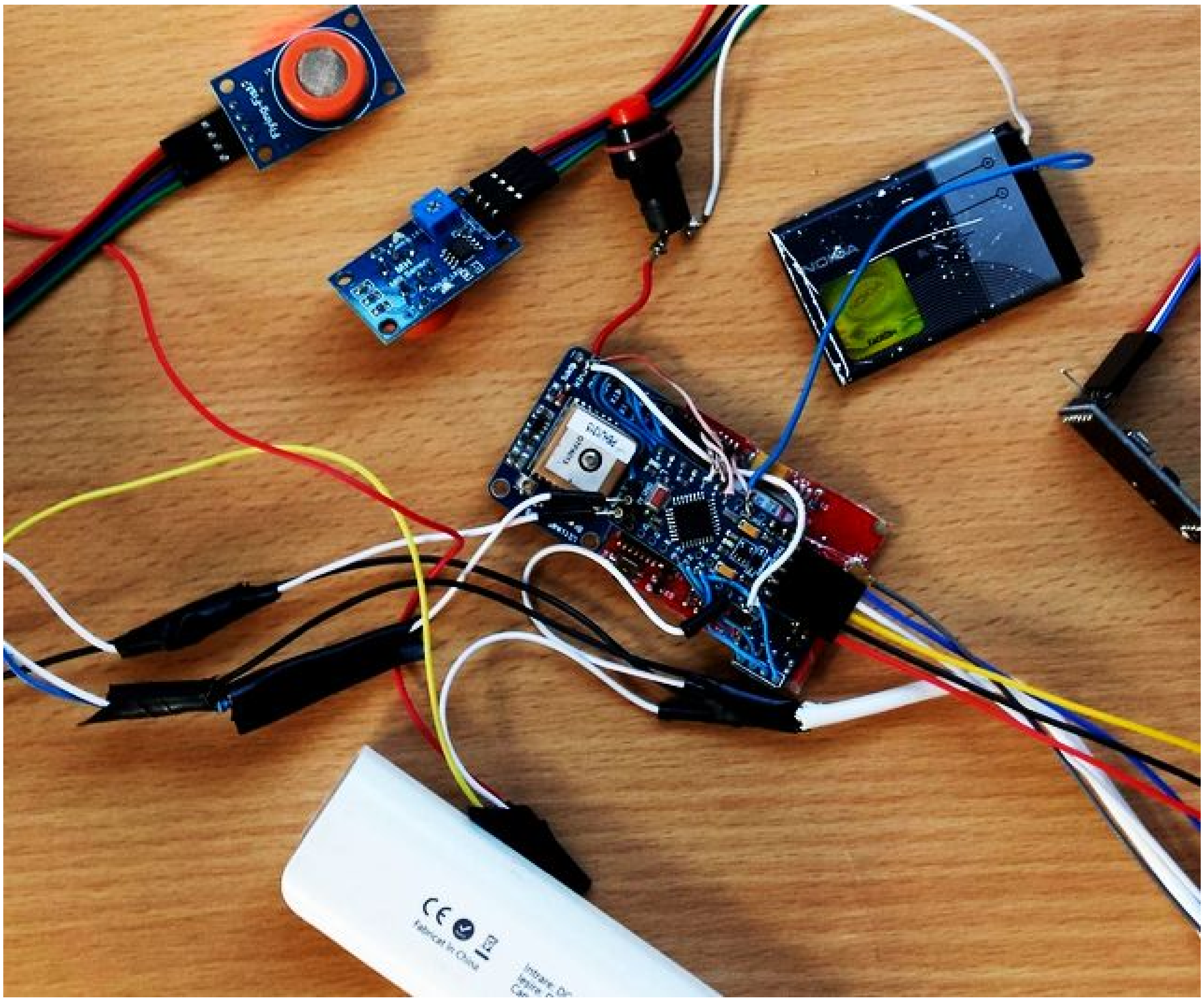


Fig. 2.1 Picture of the breathalyser

This is a closer view of the Arduino wired with a GPS and the GSM module. The accelerometer can be seen in the bottom-left corner, but it has benn cut out in order to meet some size constraints.

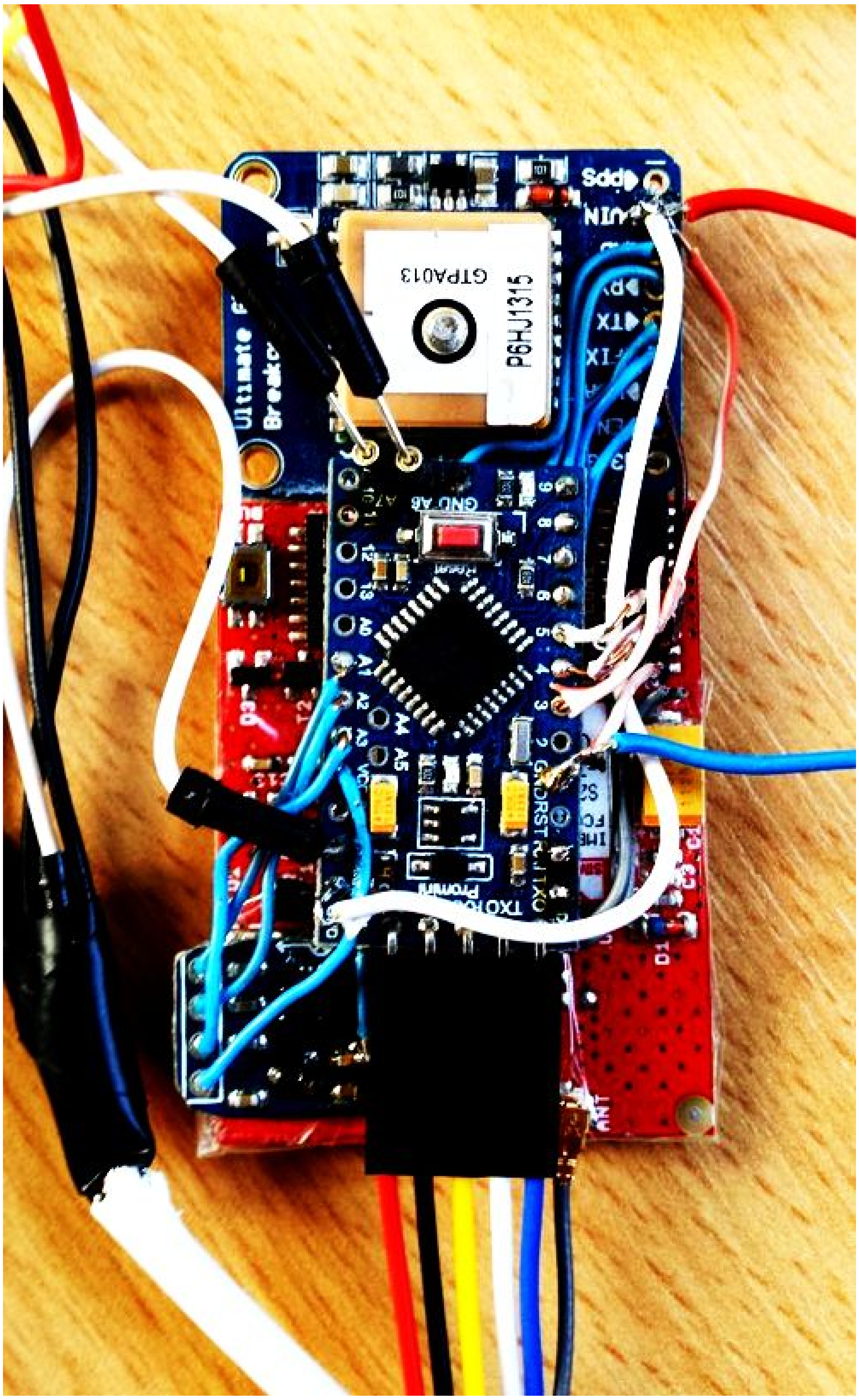


Fig. 2.2 Close-up shot without the sensors

I chose an Arduino Pro Mini as an MCU because it has an ATmega328P at its core, which is a pretty capable AVR for this task. Of course, on the way the GSM and GPS modules were added leading to a short of HW serial ports, but I managed to use two software serial ports as a workaround for this. The downside is that no two software serials may read incoming bytes at the same time, but the project didn't require it anyway so it only justified its low price. This cheap development board also has a sufficient number of analogue inputs, which is important because three are used for accelerometer and other 2 for gas sensors.

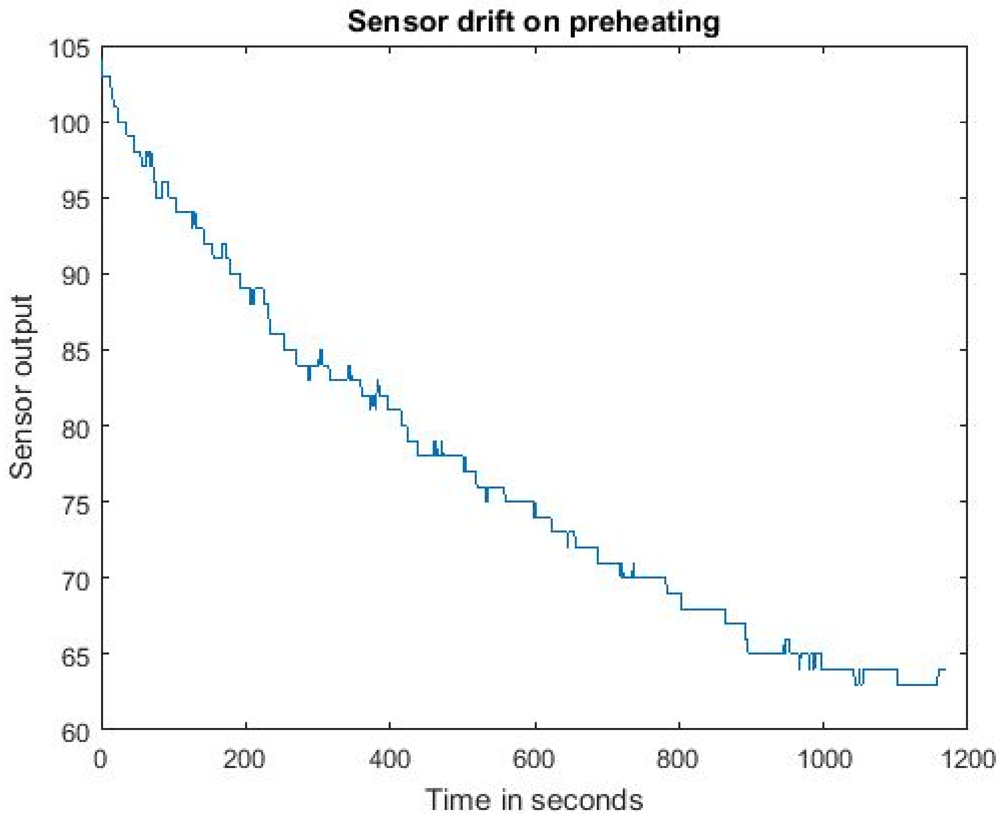


Fig 2.3 MQ-3 drift during preheating
Source: Hanwei Electronics, LTD

The reason I use two sensors is that they tend to produce a drift while being stored in a non-powered state or if the temperature / relative humidity change. This way, I only take into consideration the difference between the two, and not their absolute output value. This is mostly because these cheap sensors have a so-called preheating period after which they start to produce accurate values. This means that they have to stay powered on 48 hours before interrogation (see *Figure 2.3)* due to output drift.

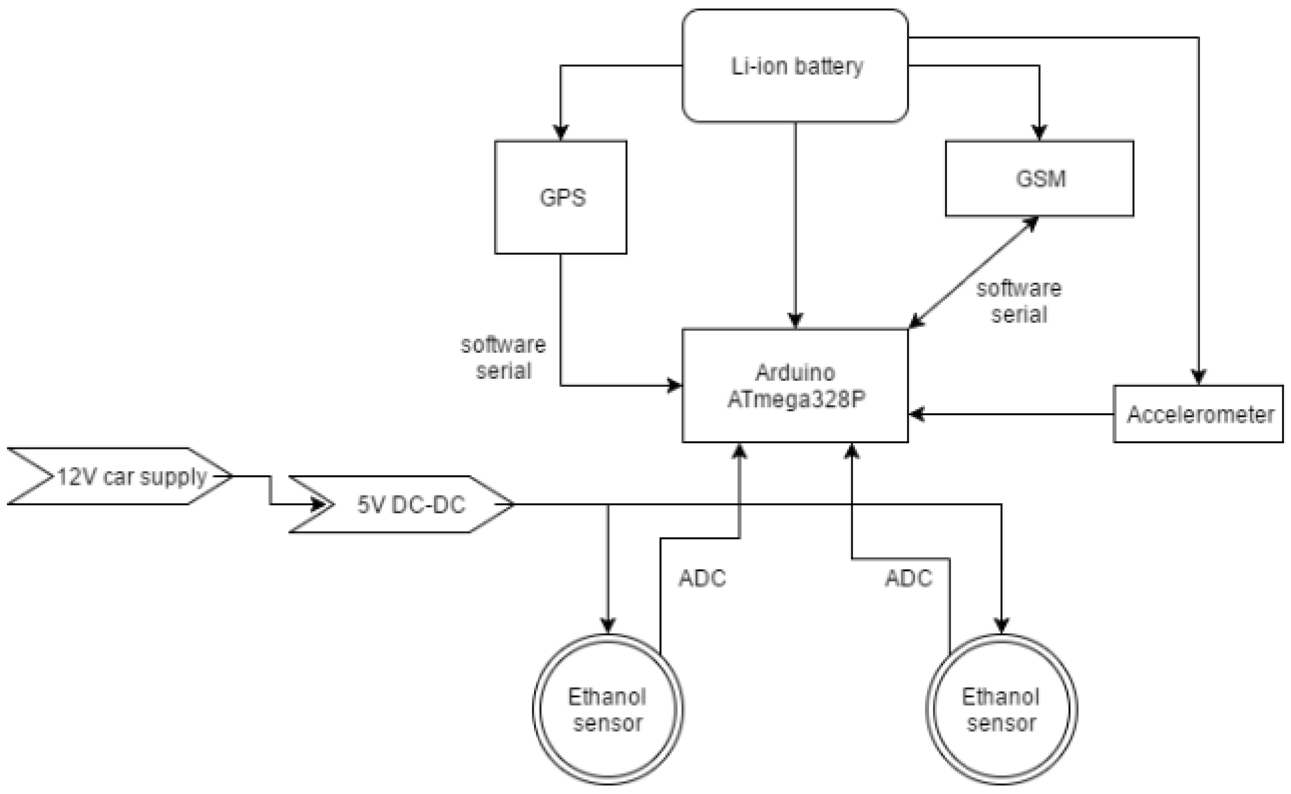


Fig. 2.4 Block diagram

The two sensors are placed like in Figure 2.4 so that the one above acts as a reference sensor which counts for drifts caused by unpredictable environmental conditions and the other one is placed somewhere lower than the exhalatory airways of the driver because ethanol vapours have a density higher than that of air so that they will fall right onto the sensor.

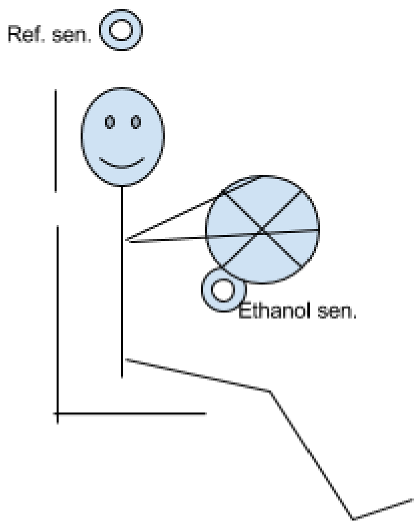


Fig. 2.5 Alcohol sensors setup

I have added an accelerometer to this project in order to detect whether the driver makes an accident (which is a very probable event in DUI cases) and to report it as soon as possible to the local authorities. In the report, geographical localisation information is sent so that intervention personnel are already heading to the place of accident before the victims even get to wake up. I use an ADXL335 module, from Analog Devices and the fact that in car accidents the acceleration vector goes over 20 g [8]. The steps taken in order to transform the three analogue values into an important decision such as reporting an accident are these:

1. Read accelerometer outputs via an ADC with the *analogRead* function (see *get_acceleration()* function in Annex 1)

2. Convert them to g values with the following formulas:

$$accXYZ = \frac{analogReading}{2^{ADC\ bits}} \times referenceVoltage \tag{2}$$

$$accVector = \sqrt{accX^2 + accY^2 + accZ^2} \tag{3}$$

   We use (2) because any rapid deceleration of the car (i.e. crash if g > 20) is going to add up to the acceleration vector, which is the sum of all the vectors on the other axes. That way, the results are not influenced by the way the device is mounted in the car.

3. Compare the result with a threshold value equal to 20 g. If the result is positive, then a crash occurred and we must take action
4. Interrogate the geographical coordinates from the GPS; this is also a very good source of exact date and time, so we should use it whenever we have a fix (there are more than 2 satellites to which the module can reach out)

5. Send all this vital data to a trusted contact. An *eCall* could also be initiated, but this feature is not currently explored in the work.

Powering the device is a bit tricky because of the variety of modules that have to be bound. So we have a 3.3V Arduino and accelerometer, the GPS can deal with signals somewhere between 3.3V and 5V, the GSM module requires to be powered from a battery cell (3.8V - 4.2V) and the gas sensors need to be heated from 5V. Fortunately, I could make use of dc-dc regulators that are present on the Arduino and GPS module in order to power all the components from a USB (5V) cable that goes into a 12-to-5 V car plug but the GSM module still needs a battery for its SIM800H chip.

3. Results

Here you can see that the sensors are fairly sensible even to small quantities of alcohol intake:

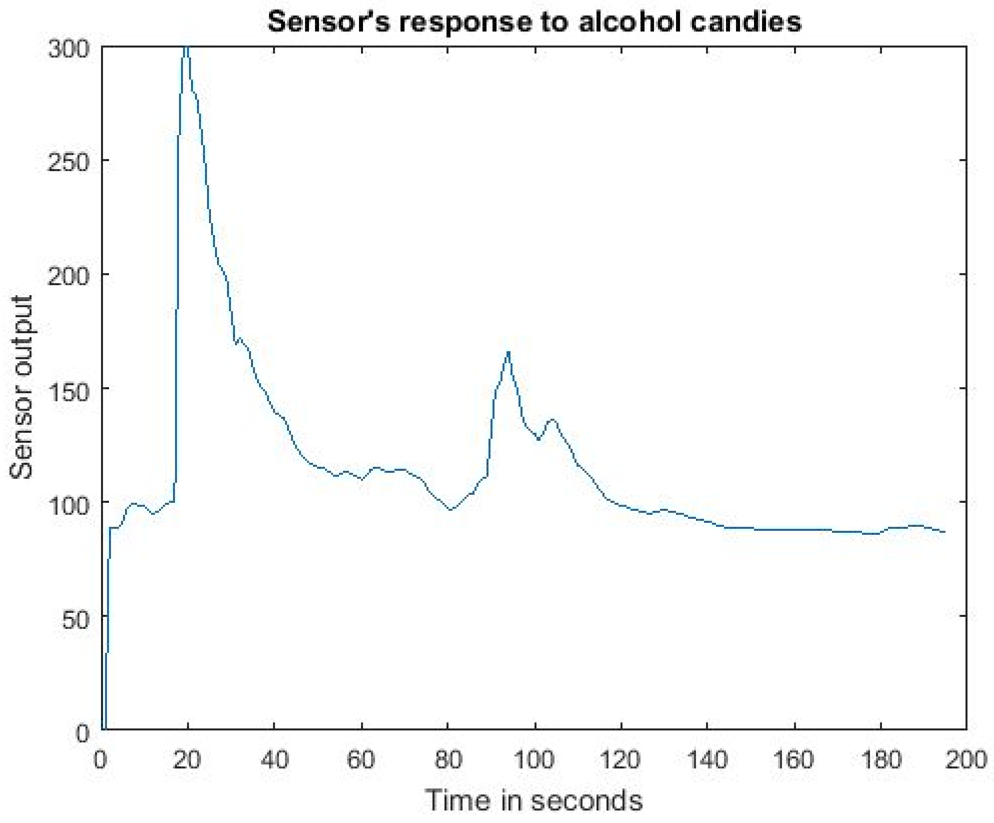


Fig. 3.1 Response to three alcohol candies

What counts more is the position and distance from the sensors. Even when the difference in output is low, with a little bit of processing in MATLAB the response is clear, and a decision can be made (Figure 3.2, 3.3).

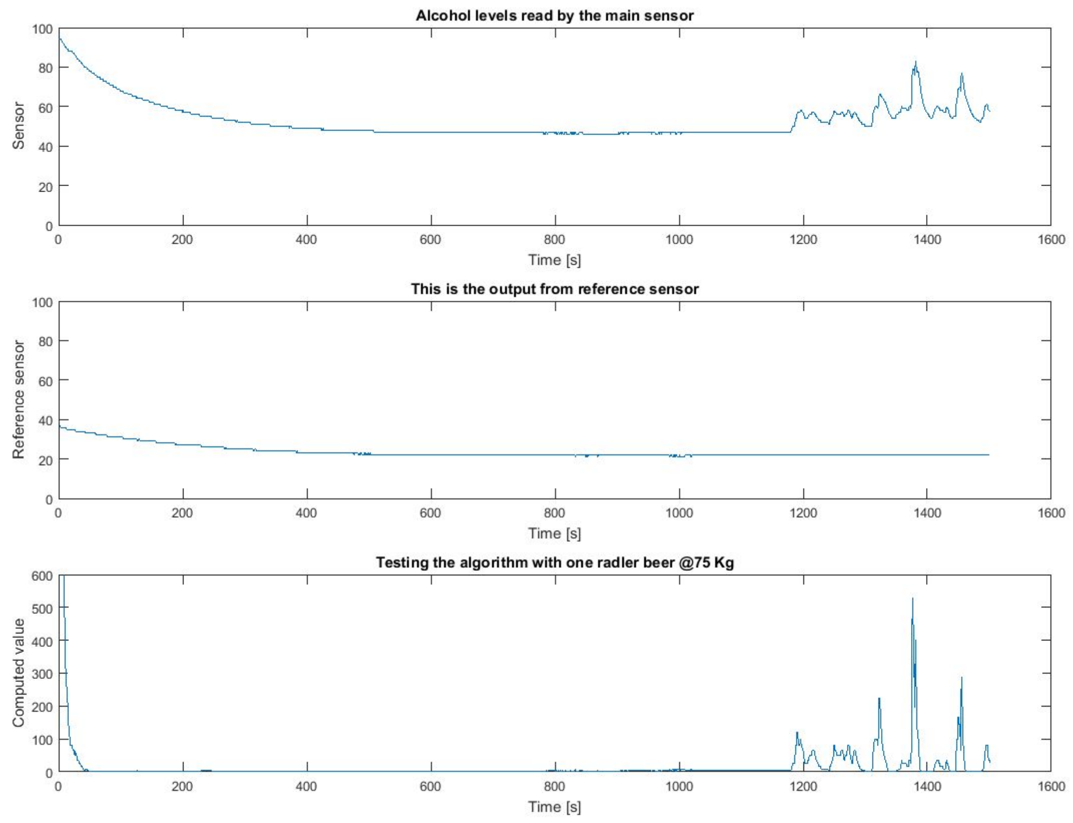


Fig. 3.2 Processing results in MATLAB

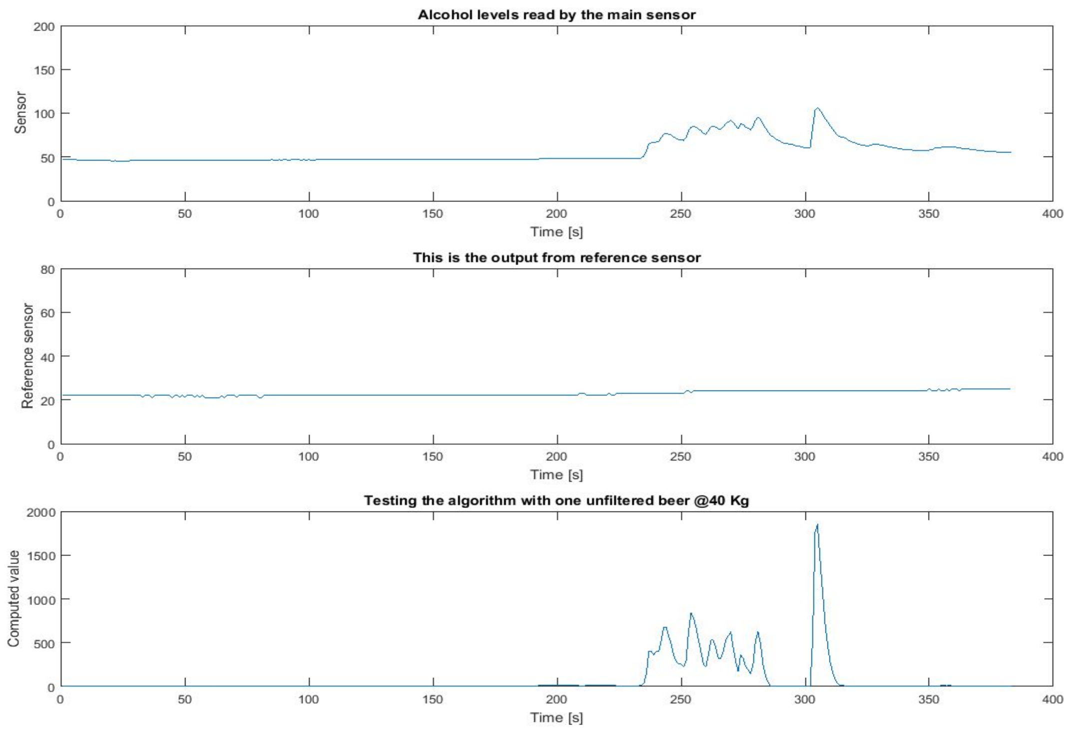


Fig. 3.3 Data processing in MATLAB

Conclusions

From what I managed to implement as a device ready-to-mount and the results that were presented in the previous chapter, one could see that the proposed design is not only feasible but a cost effective way to protect yourself or others from getting drunk behind the wheel. Given the fact that while some people keep a better safe than sorry conduct while they are under the influence of alcohol, for many others it is impossible to control themselves so the need for a gadget that reminds them that they shouldn't drive is real.
Future thoughts
At the end of this work, I want to put some light onto the greater project from which the breathalyser took birth. At the very base of all this effort is a sincere desire of mine to bring an end to deaths caused by car accidents. Despite the fact that in most of the cases the machines are not to blame, but the people, we can use technology in order to make up for our human faults like slow response times, distraction, not obeying traffic rules and so on.

Of course, the first thing that comes to mind in this regard is the autonomous car, which is a trend every big car manufacturer is trying to follow. But until we get to the point that cars will do this job for us, I think that any car could benefit from adding a few features of security, even if that costs a few hundred dollars. After all, it may save your life just once, and it already makes for all the money in the world.

In this regard, I thought of a dashboard that everyone could mount on their car, and this would fulfil the following functions:

- Run on Android tablet (could imitate the dashboard of a Tesla, which is cool)
- Use the camera to recognize in real-time traffic lights, and to ring a bell / apply the brakes if the driver doesn't seem to slow down with red ahead
- Use an online database published by some authority (i.e. the local road administration office) and retrieve from it the traffic signs that are intended to be followed by the driver. Then, show them on a big screen for the times when they are visually blocked by obstacles or they are destroyed. The ECU of the car could also take some intelligent actions depending on what is in front of the driver: a pedestrian crosswalk, a road with priority, etc. It is not very complicated to do this, it is only needed a GPS and a compass, in order to know in which direction the driver is heading.
- On this tablet a *parental control application* can be installed by the parents, and this would report driving style metadata of the teens that use the car to their parents, could limit some safety functionalities like music volume, speed on rainy weather, belt use, etc. Also, allowing certain driving hours and geographical areas is possible and even setting an upper speed limit for when the visibility is relatively low, or the road section has less grip than usual.

- An application could also have multiple users registered, and could recognize each one of them based on their weight combined with their height or voice / face traits. After recognizing a registered driver, the car could automatically adjust mirrors, seats, radio stations, etc according to the preffered / last used configuration
- People that can spend more on their car gadgets could connect this dashboard to the output of a termographic (FLIR) camera. That way, the driver can easily spot living beings in nighttime, like wild animals or pedestrians.

Bibliography

[1] Abdul-Rahaman Haadi, Identification of Factors that Cause Severity of Road Accidents in Ghana: a Case Study of the Northern Region, Tamale Polytechnic

[2] Alcohol. https://en.wikipedia.org/wiki/alcohol (accessed Jan 19, 2016)

[3] Ethanol. https://en.wikipedia.org/wiki/ethanol (accessed Jan 19, 2016)

[4] Catalysis for Sustainable Energy Production. Edited by P. Barbaro and C. Bianchini Copyright 2009 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

[5] Kate B. Carey and John T. P. Hustad, *Methods for determining blood alcohol concentration,* Center for Health and Behavior at Syracuse University, 2006

[6] Georgia Koukiou,1 M.Sc.; and Vassilis Anastassopoulos, *Drunk Person Screening using Eye Thermal,* Department of Physics University of Patras, 2016

[7] Georgia Koukioua and Vassilis Anastassopoulosa, *Intoxicated person discrimination using infrared signature of facial blood vessels,* Australian Journal of Forensic Sciences, *2015*

[8] Augustus Chidester, John Hinch, Thomas C. Mercer, Keith S. Schultz, *Recording Automotive Crash Event Data, 1999*

Annex 1

```
/*     PROJECT: Breathalyzer
 *
 *        FILE: terra-drive.ino
 *
 * DESCRIPTION: This is the program for an Arduino breathalyzer.
 *              It's main purpose is to discourage drunk driving.
 *
 *      AUTHOR: Loredan Emilio Bucur (lebucur)
 *
 *     LICENSE: 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.
 *
 *   LAST_EDIT: June 2016
 */

//DEFINE
#define DEBUG 1
#define MEAN(a, b) ((a + b) / 2);
#define CRASH_G 1.5f //20-40 usual values
#define FAIL -1
#define TASK_ACC 1
#define TASK_ACC_INT 50L
#define TASK_MAIN 2
#define TASK_MAIN_INT 1000L
#define TASK_LOC 4
#define TASK_LOC_INT 5000L
#define BAUD_RATE 9600L

//INCLUDE
#include "./Debug.h"
#include <SoftwareSerial.h>
#include <AltSoftSerial.h>
#include <Adafruit_FONA.h>
#include <TinyGPS.h>
#include <ADXL335.h>

//CONSTANTS
struct {
    static const byte mq3     = A6;
    static const byte mq3_ref = A7;
    static const byte led     = 13;
    static const byte acc_x   = A1;
    static const byte acc_y   = A2;
    static const byte acc_z   = A3;
    static const byte gsm_pwr = 3;
    static const byte gsm_tx  = 4;
    static const byte gsm_rx  = 5;
    static const byte gps_tx  = 8;
    static const byte gps_rx  = 9;
    static const byte gps_en  = 6;
    static const byte gps_fix = 7;
} Pins;

AltSoftSerial gps_ser;
SoftwareSerial gsm_ser = SoftwareSerial(Pins.gsm_tx, Pins.gsm_rx);
Adafruit_FONA gsm = Adafruit_FONA(Pins.gsm_pwr);
ADXL335 accel(Pins.acc_x, Pins.acc_y, Pins.acc_z, 3.3); //3.3V is analog reference
TinyGPS gps;

//GLOBALS
byte active_tasks;
unsigned sen, sen_ref; //0..1023 ADC sensors
unsigned diff_stable, diff;
unsigned long difp, sen_threshold = 1000L;
unsigned long last_task_Acc, last_task_Main, last_task_Loc;

struct GSM_data {
    char contact[11] = "0741405354";
    float latitude, longitude; //, datetime
} GSM;

struct GPS_data {//reduce scope to temporary
    long latitude, longitude;
    //unsigned long fix_age, time, date; //last_update?
    const char * cardinal;
    float speed; // ?
} GPS;

void setup()
{
    DBG_begin(BAUD_RATE); //same as SS, or twice
    gsm_ser.begin(BAUD_RATE); //as high as possible on SS
    gps_ser.begin(9600); //as small as possible on ASS
    DBG_printsln("SETUP");

    //declare output/input_pullup pins if any
    pinMode(Pins.led, OUTPUT);
    pinMode(Pins.mq3, INPUT_PULLUP); //an alert should be generated
    pinMode(Pins.mq3_ref, INPUT_PULLUP); //if sensors are disconnected
    pinMode(Pins.gsm_pwr, OUTPUT);
    pinMode(Pins.gps_en, OUTPUT);
    digitalWrite(Pins.gps_en, HIGH); //enable==HIGH
    gsm.begin(gsm_ser);
    DBG_prints("Battery: ");
    DBG_print(get_battery()); DBG_printsln("%");

    active_tasks |= TASK_ACC;
    active_tasks |= TASK_MAIN;
    active_tasks |= TASK_LOC;

    print_commands();
}

void loop()
{
    unsigned long current_time = millis();

    //Acceleration thresold check
    if ((active_tasks & TASK_ACC) && (current_time - last_task_Acc > TASK_ACC_INT)) {
        if (get_acceleration()) {
            report_accident();
        }
        last_task_Acc = current_time;
        current_time = millis();
    }

    //Main task: alcohol monitoring
    if ((active_tasks & TASK_MAIN) && (current_time - last_task_Main > TASK_MAIN_INT)) {
        if (process_data()) {
            report_dui();
        }
        last_task_Main = current_time;
        current_time = millis();
    }

    //Location interogation
    if ((active_tasks & TASK_LOC) && (current_time - last_task_Loc > TASK_LOC_INT)) {
        get_location();
        last_task_Loc = current_time;
        //current_time = millis();
    }

    if (get_free_ram() < 100) { //bytes
        DBG_printsln(" ** NOT ENOUGH RAM ** ");
    }
}

bool get_acceleration()
{
    accel.update();
    if (accel.getRho() > CRASH_G) {
        DBG_println(accel.getRho());
        return true;
    }
    return false;
```

```
}

unsigned get_battery()
{
    unsigned vbat = 303;
    gsm.getBattPercent(&vbat);
    return vbat;
}

bool process_data()
{
    sen = analogRead(Pins.mq3);
    delay(1);
    sen = MEAN(sen, analogRead(Pins.mq3)); //average readings; LPF
    delay(1);
    sen_ref = analogRead(Pins.mq3_ref);
    delay(1);
    sen_ref = MEAN(sen_ref, analogRead(Pins.mq3_ref));
    delay(1);

    diff = sen > sen_ref ? sen - sen_ref : 0; //unsigned, for now
    difp = diff > diff_stable ? diff - diff_stable : 0;
    difp *= difp;
    diff_stable = 0.9 * diff_stable + 0.1 * diff; //testez diferite valori

    DBG_prints("sen:"); DBG_print(sen);
    DBG_prints(" ref:"); DBG_print(sen_ref);
    DBG_prints(" diff_stable:"); DBG_print(diff_stable);
    DBG_prints(" diff:"); DBG_print(diff);
    DBG_prints(" difp:"); DBG_println(difp);

    return difp > sen_threshold;
}

void report_dui()
{
    DBG_printsln("Taking action:");
    char message[141];
    sprintf(message, "Hello! You have a drunk driver at: %ld lat, %ld lon heading %s.",
            GPS.latitude, GPS.longitude, GPS.cardinal);
    send_sms(message);
}

void report_accident()
{
    DBG_printsln("Reporting crash:");
    char message[141]; //make sure you don't put here more than 140 chars
    sprintf(message, "Your car has just crashed at latitude %ld and longitude %ld!",
GPS.latitude, GPS.longitude);
    send_sms(message);
}

bool send_sms(char * message)
```

```
{
    if (gsm.sendSMS(GSM.contact, message)) {
        DBG_prints("SMS sent to "); DBG_print(GSM.contact);
        DBG_prints(": "); DBG_println(message);
        return true;
    }
    else {
        DBG_printsln("SMS failed to send!");
        return false;
    }
}

//bool get_location(const char ptr
void get_location() //takes a few secs
{
    bool not_fix = true;
    //try to get GPS location
    unsigned long start_time = millis();
    do { //should be done asynchronously
        if (gps_ser.available()) {
            char c = gps_ser.read();
            if (gps.encode(c)) { //feed gps object with bytes
            //got fix
            //if done processing a sentence
                gps.get_position(&GPS.latitude, &GPS.longitude); //+/- lat/long in
degrees
                //gps.get_datetime(&GPS.date, &GPS.time, &GPS.fix_age); //time in
hhmmsscc, date in ddmmyy
                GPS.cardinal = gps.cardinal(gps.course());
                GPS.speed = gps.f_speed_kmph();

                //DBG_prints("age:"); DBG_print(GPS.fix_age);
                DBG_prints("lat:"); DBG_print(GPS.latitude);
                DBG_prints(" lon:"); DBG_print(GPS.longitude);
                DBG_prints(" speed:"); DBG_print(GPS.speed);
                DBG_prints(" direction:"); DBG_println(GPS.cardinal);
                //DBG_prints(" date:"); DBG_print(GPS.date);
                //DBG_prints(" time:"); DBG_println(GPS.time);

                not_fix = false;
                //return true;
                break; //enough
            }
        }
    } while (millis() - start_time <= 1000L); //1Hz update rate

    if (not_fix) {
        DBG_printsln("Querying location from GSM cells");
        //need to turn on GPRS for GSMLoc
        toogle_gprs(true);
        if (gsm.getGSMLoc(&GSM.latitude, &GSM.longitude)) {
            DBG_prints("Latitude: "); DBG_print(GSM.latitude);
            DBG_prints(" Longitude: "); DBG_println(GSM.longitude);
            //return true;
        }
        else { //toogle GPRS, then try again
            toogle_gprs(false);
            toogle_gprs(true);
            if (gsm.getGSMLoc(&GSM.latitude, &GSM.longitude)) {
                DBG_prints("Latitude: "); DBG_print(GSM.latitude);
                DBG_prints(" Longitude: "); DBG_println(GSM.longitude);
                //return true;
            }
            else DBG_printsln("Could not get GPRS location data!");
        }
    }
    //return false;
}

void flush_serial()
{
    while (Serial.available()) {
        Serial.read();
    }
}

void wait_serial()
{
    while (!Serial.available());
}

void toogle_gprs(bool state)
{
    //only if not corresponding to given argument
    if (state != gsm.GPRSstate()) {
            if (gsm.enableGPRS(state)) {
                DBG_prints("GPRS mode set to ");
                DBG_println(state);
            }
            else {
                DBG_printsln("Could not toogle GPRS state!");
                //return false;
            }
    }
}

void gps_debug_mode()
{
    //gps_ser.listen();
#if DEBUG
    DBG_printsln("\n** GPS debug mode started **");
       char c = 100;
       do {
             if (gps_ser.available()) {
                    Serial.write(gps_ser.read());
             }
```

```
              if (Serial.available()) {
                     c = Serial.read();
             Serial.write(c);
                     gps_ser.write(c);
              }
       } while (c != '`');
       DBG_println("\n** Exit debug mode ** ");
    print_commands();
#endif
}

void gsm_debug_mode()
{ //here the execution stops @ gsm_debug() @ serialEvent() @ loop()
#if DEBUG
       DBG_println("\n** GSM debug mode started **");
       char c = 100;
       do {
              if (gsm_ser.available()) {
                     Serial.write(gsm_ser.read());
              }
              if (Serial.available()) {
                     c = Serial.read();
             //Serial.write(c);
                     gsm_ser.write(c);
              }
       } while (c != '`');
       DBG_println("\n** Exit debug mode ** ");
    print_commands();
#endif
}

bool is_printable(char c)
{
    if (c >= 32) { //that's where printable chars begin in ASCII table
        return true;
    }
    return false;
}

bool is_figure(char c) {
    if (c >= '0' && c <= '9') {
        return true;
    }
    return false;
}

void print_commands()
{
    DBG_println("Commands:");
    DBG_println("  g: enter GSM debug mode");
    DBG_println("  G: set GPRS mode #");
    DBG_println("  p: enter GPS debug mode");
    DBG_println("  m: show free RAM");
    DBG_println("  n: change contact phone number");
    DBG_println("  s: send a test SMS");
    DBG_println("  z: toogle task #");
}

void serialEvent()
{ //called at the end of each loop() execution if new bytes arrive in RX
  //it takes a little while until it is executed again
  //does not respond at events in other functions like gsm_debug
#if DEBUG
    unsigned char c = Serial.read();
    switch (c) //one ASCII character command
    {
        case 'g':
            gsm_debug_mode();
        break;

        case 'p':
            gps_debug_mode();
        break;

        case 'm':
            DBG_prints("Free RAM: ");
            DBG_print(get_free_ram());
            DBG_println(" bytes");
        break;

        case 'n':
            flush_serial();
            DBG_prints("New contact phone number: ");
            for (int i = 0; i < 10; i++) {
                wait_serial();
                char f = Serial.read();
                if (is_figure(f)) {
                    GSM.contact[i] = f;
                    DBG_print(f);
                }
            }
            flush_serial();
            DBG_println();
        break;

        case 's':
            char message[141];
            sprintf(message, "Hello! I\'m at %ld lat %ld lon heading %s.",
                    GPS.latitude, GPS.longitude, GPS.cardinal);
            send_sms(message);
        break;

        case 'G':
        {
            wait_serial();
            char s = Serial.read();
            if (is_figure(s)) {
                if (s == '0') { //disable GPRS
                    toogle_gprs(false);
                }
                else if (s == '1') {
                    toogle_gprs(true);
                }
            }
        }
        break;

        case 'z':
        {//this creates a scope, so we can create vars in this case w/o errors
            wait_serial();
            char t = Serial.read();
            if (is_figure(t)) {
                if (t == '9') {//toogle all tasks
                    active_tasks = active_tasks ? 0x00 : 0xFF;
                }
                active_tasks ^= 1 << ((t + 1) % 10); // '1'==49
                DBG_prints("Active tasks: ");
                DBG_printfln(active_tasks, BIN);
            }
        }
        break;

        case '?':
            print_commands();
        break;

        default:
        break;
    }
#endif
}
```

Annex 2

```
/*
 *        FILE: Debug.h
 * DESCRIPTION: Header for a library useful for debugging
 *      AUTHOR: Loredan E. Bucur
 *   LAST_EDIT: Jan 2016
 */


#ifndef DEBUG_H_INCLUDED
#define DEBUG_H_INCLUDED

int get_free_ram();

#if DEBUG //no problem if not defined at all
#define DBG_begin(baud_rate) while(!Serial); Serial.begin(baud_rate)
#define DBG_print(msg) Serial.print(msg)
#define DBG_println(msg) Serial.println(msg)
#define DBG_prints(msg) Serial.print(F(msg))
#define DBG_printsln(msg) Serial.println(F(msg))
#define DBG_printf(msg, format) Serial.print(msg, format) //format = {DEC, HEX, OCT, BIN,
float_precision}
#define DBG_printfln(msg, format) Serial.println(msg, format)
#define DBG_write(msg) Serial.write(msg)

#else //not defined or zero
#define DBG_begin(baud_rate)
#define DBG_print(msg)
#define DBG_println(msg)
#define DBG_prints(msg) Serial.print(F(msg))
#define DBG_printsln(msg) Serial.println(F(msg))
#define DBG_printf(msg, format)
#define DBG_printfln(msg, format)
#define DBG_write(msg)
#endif //DEBUG

#endif //DEBUG_H_INCLUDED
```

Annex 3

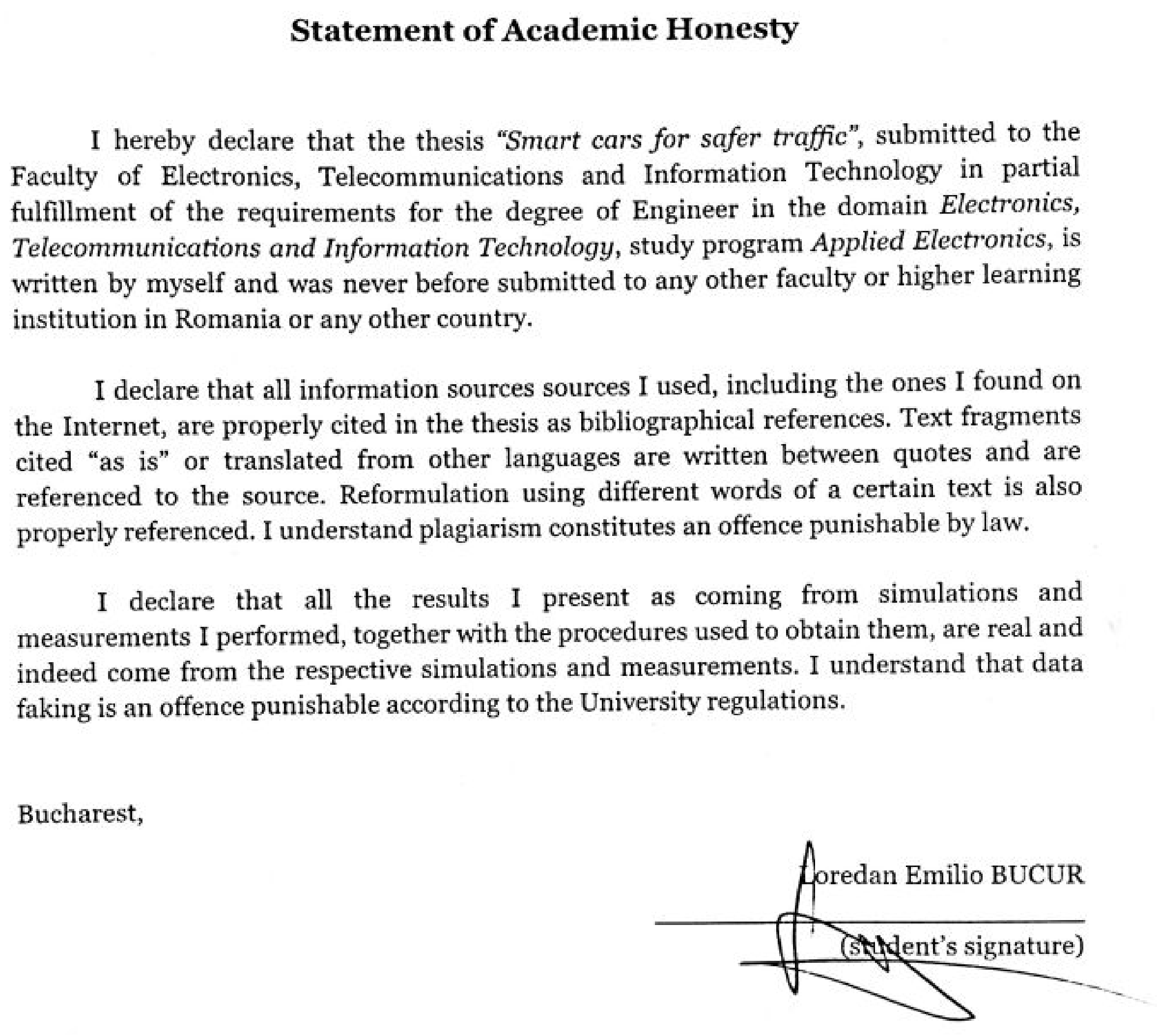
```
/*
 *        FILE: Debug.cpp
 * DESCRIPTION: C++ source code for a debugging library
 *      AUTHOR: Loredan E. Bucur
 *   LAST_EDIT: Jan 2016
 */


#if defined(ARDUINO) && ARDUINO >= 100
       #include "Arduino.h" //for Serial and other stuff
#else
       #include "WProgram.h"
#endif

#include "Debug.h"

int get_free_ram()
{
    extern unsigned __heap_start;
    extern unsigned * __brkval;
    int a;
    return (int) &a - (__brkval == 0 ? (int) &__heap_start : (int) __brkval);
}
```

Annex 4

```
%        FILE: logger.m
%     PROJECT: Breathalyzer
% DESCRIPTION: MATLAB code used for real-time analysis
%   LAST EDIT: June 2016

clear
format shortG

loopDelay = 1; %[s]
comPort = 'COM11';
saveDir = './results/';
scroll = 100;
yMin = 20;

if scroll > 0
    it = scroll;
else
    it = 0;
end
time = 0;
data = [0 0];
figure
plotGraph = plot(time, data, 'LineWidth', 1);
title('Live data from 3 sensors', 'FontSize', 18);
xlabel('Time in seconds');
ylabel('Ethanol concentration');
legend('driver', 'reference', ...
    'orientation', 'horizontal', 'location', 'south')
grid on
drawnow

ser = instrfind('Type', 'serial', 'Port', comPort);
if isempty(ser)
    ser = serial(comPort);
else
    fclose(ser);
    ser = ser(1);
end
fopen(ser);

tic
while ishandle(plotGraph)
    if ser.BytesAvailable
        [values, inCnt, ret] = fscanf(ser, '%d %d');
        if ~isempty(ret)
            continue % move on
        end

        it = it + 1;
        time(it) = toc;
        data(it, :) = values;
        Stats.max = max(data);

        if scroll < 0
            set(plotGraph(1), 'XData', time, 'YData', data(:, 1))
            set(plotGraph(2), 'XData', time, 'YData', data(:, 2))
            axis([time(1), time(it)+10, ...
                yMin, 1.1*max(Stats.max)]);
        else
            set(plotGraph(1), ...
                'XData', time(it - scroll : it), ...
                'YData', data(it - scroll : it, 1));
            set(plotGraph(2), ...
                'XData', time(it - scroll : it), ...
                'YData', data(it - scroll : it, 2));
            axis([time(it)-scroll, time(it), ...
                yMin, 1.1*max(Stats.max)]);
        end
        pause(loopDelay)
    end
end

Stats.min = min(data);
Stats.median = median(data);
Stats.peak = Stats.max - Stats.min;
Stats.mean = mean(data);
Stats.variance = var(data);
Stats.deviation = std(data);

clearvars -except data Stats time saveDir

fileName = input('If you want these vars saved to a mat file, type a name for it: ',
's');
if isempty(fileName) || strcmp(fileName, 'no') || strcmp(fileName, 'n') ||
strcmp(fileName, 'N')
    disp('Okay, no save this time. Bye!')
else
    fileName = strcat(saveDir, fileName);
    msg = input('Any comments about this session?\n', 's');
    save(fileName)
    disp(['You can find your work in ' fileName])
end

open_connections = instrfindall; %find all open connections

if ~isempty(open_connections)
    fclose(open_connections); %close them
    disp('Session closed.')
else
```

```
    disp('No open connections.')
end

delete(open_connections)
clear open_connections
```

Annex 5



Statement of Academic Honesty

I hereby declare that the thesis *"Smart cars for safer traffic"*, submitted to the Faculty of Electronics, Telecommunications and Information Technology in partial fulfillment of the requirements for the degree of Engineer in the domain *Electronics, Telecommunications and Information Technology*, study program *Applied Electronics*, is written by myself and was never before submitted to any other faculty or higher learning institution in Romania or any other country.

I declare that all information sources sources I used, including the ones I found on the Internet, are properly cited in the thesis as bibliographical references. Text fragments cited "as is" or translated from other languages are written between quotes and are referenced to the source. Reformulation using different words of a certain text is also properly referenced. I understand plagiarism constitutes an offence punishable by law.

I declare that all the results I present as coming from simulations and measurements I performed, together with the procedures used to obtain them, are real and indeed come from the respective simulations and measurements. I understand that data faking is an offence punishable according to the University regulations.

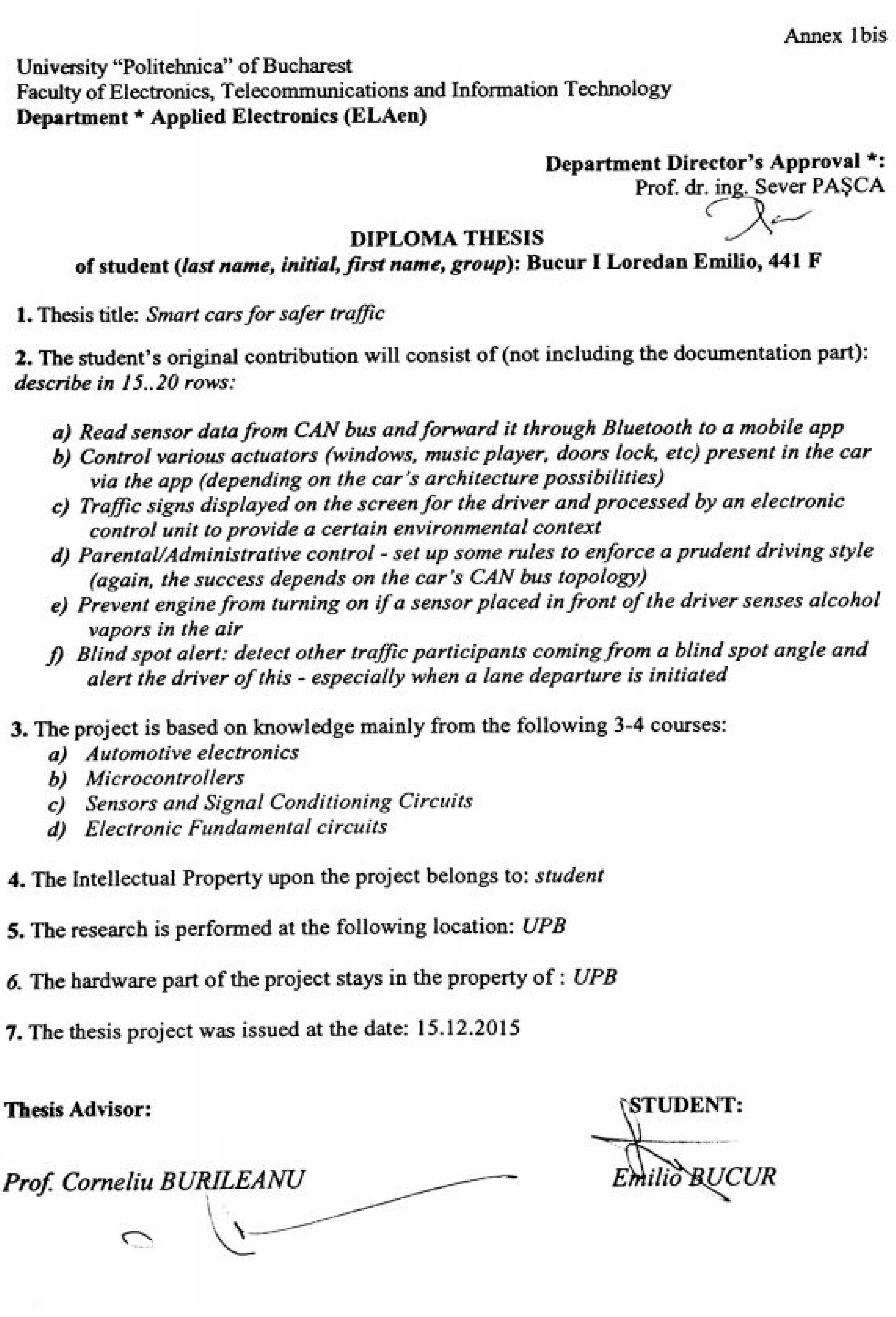
Bucharest,

Loredan Emilio BUCUR

[illegible]

(student's signature)

Annex 6



Annex 1bis

University "Politehnica" of Bucharest
Faculty of Electronics, Telecommunications and Information Technology
Department * Applied Electronics (ELAen)

Department Director's Approval *:
Prof. dr. ing. Sever PAŞCA
[illegible]

DIPLOMA THESIS
of student (*last name, initial, first name, group*): Bucur I Loredan Emilio, 441 F

1. Thesis title: *Smart cars for safer traffic*

2. The student's original contribution will consist of (not including the documentation part):
describe in 15..20 rows:

a) Read sensor data from CAN bus and forward it through Bluetooth to a mobile app
b) Control various actuators (windows, music player, doors lock, etc) present in the car via the app (depending on the car's architecture possibilities)
c) Traffic signs displayed on the screen for the driver and processed by an electronic control unit to provide a certain environmental context
d) Parental/Administrative control - set up some rules to enforce a prudent driving style (again, the success depends on the car's CAN bus topology)
e) Prevent engine from turning on if a sensor placed in front of the driver senses alcohol vapors in the air
f) Blind spot alert: detect other traffic participants coming from a blind spot angle and alert the driver of this - especially when a lane departure is initiated

3. The project is based on knowledge mainly from the following 3-4 courses:
a) Automotive electronics
b) Microcontrollers
c) Sensors and Signal Conditioning Circuits
d) Electronic Fundamental circuits

4. The Intellectual Property upon the project belongs to: *student*

5. The research is performed at the following location: *UPB*

6. The hardware part of the project stays in the property of : *UPB*

7. The thesis project was issued at the date: 15.12.2015

Thesis Advisor: **STUDENT:**

Prof. Corneliu BURILEANU [illegible] *Emilio BUCUR* [illegible]

Annex 7