**“ALEXANDRU IOAN CUZA” UNIVERSITY OF IAȘI**

**FACULTY OF COMPUTER SCIENCE**

BATCHELOR’S THESIS

# ACCIDeNtS – Automatic Car Crashes and Incidents Detection and Notification System

**Proposed by**

***Alexandra-Florentina Cîtea***

**Session:** *July 2018*

**Scientific Coordinator**

**Lect dr. Vârlan Cosmin**

**“ALEXANDRU IOAN CUZA” UNIVERSITY OF IAȘI**

**FACULTY OF COMPUTER SCIENCE**

# ACCIDeNtS – Automatic Car Crashes and Incidents Detection and Notification System

***Alexandra-Florentina Cîtea***

**Session:** *July 2018*

**Scientific Coordinator**

**Lect dr. Vârlan Cosmin**

**Avizat,**

**Îndrumător Lucrare de Licenţă**

Titlul, Numele şi prenumele _____

Data _____ Semnătura _____

## DECLARAŢIE privind originalitatea conţinutului lucrării de licenţă/diplomă/disertaţie/absolvire

Subsemntatul(a)……

domiciliul în ……..

născut(ă) la data de ……..…., identificat prin CNP ……..……..……..., absolvent(a) al(a) Universităţii „Alexandru Ioan Cuza" din Iaşi, Facultatea de ……. specializarea ……, promoţia ……..,

declar pe propria răspundere, cunoscând consecinţele falsului în declaraţii în sensul art. 326 din Noul Cod Penal şi dispoziţiile Legii Educaţiei Naţionale nr. 1/2011 art.143 al. 4 si 5 referitoare la plagiat, că lucrarea de licenţă cu titlul: _____

_____

_____

elaborată sub îndrumarea dl. / d-na _____, pe care urmează să o susţină în faţa comisiei este originală, îmi aparţine şi îmi asum conţinutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licenţă/diplomă/disertaţie/absolvire să fie verificată prin orice modalitate legală pentru confirmarea originalităţii, consimţind inclusiv la introducerea conţinutului său într-o bază de date în acest scop.

Am luat la cunoştinţă despre faptul că este interzisă comercializarea de lucrări ştiinţifice in vederea facilitării fasificării de către cumpărător a calităţii de autor al unei lucrări de licenţă, de diploma sau de disertaţie şi în acest sens, declar pe proprie răspundere că lucrarea de faţă nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Dată azi, …… Semnătură student ……

# DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licenţă cu titlul „*ACCIDeNtS – Automatic Car Crashes and Incidents Detection and Notification System*”, codul sursă al programelor şi celelalte conţinuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” din Iaşi, să utilizeze, modifice, reproducă şi să distribuie în scopuri necomerciale programele-calculator, format executabil şi sursă, realizate de mine în cadrul prezentei lucrări de licenţă.

Iaşi,

Absolvent *Alexandra-Florentina Cîtea*

_____

(semnătura în original)

# Contents

# INTRODUCTION

Each day, a tremendous number of car accidents occur in the world and unfortunately, a major part of them lead to human victims. According to the Association for Safe International Road Travel[1], the average number of people dying in car accidents is 3,287 per day. 20 to 50 million people are injured every year and most of them eventually die because help does not come fast enough. It is true that this serious problem became a real concern for the world and more and more car manufacturers began developing systems to automatically call for help in case of an accident; other organizations even began developing systems for older cars. I wanted to make such a system on my own. My project is about saving lives, about taking smart decisions and about efficiently centralizing and interpreting data: ACCIDeNtS.

ACCIDeNtS, or more exactly Automatic Car Crashes and Incidents Detection and Notification System is a system where both the software and the hardware are synchronized to reduce the number of victims after an accident. When an accident occurs, the hardware part makes sure to report the data to the main application. The can bus module connected to the car's computer records the unusual data during and after an accident such as suddenly breakings, speed and temperature and sends all of it by bluetooth to a small box under the passenger's seat, where the 'brain' of the system is: the Raspberry Pi module. Using an efficient algorithm where Fuzzy Logic classifies the received data and interprets it to decide if an accident happened, the Raspberry Pi module sends an SMS using a GPS/GSM module containing the coordinates and the priority of the accident. The message is received by another GSM module which is connected to an Arduino module that stores the data in a database. From here, a Django application displays it on a real time worldwide map, filters the accidents by the priority and maintain and guide the units to the place of the accident.

The European Union began developing and using such systems for saving lives. eCall[2] is an initiative to report accidents, mandatory to the cars sold on the EU territory from April 2018. The concept of this system was first presented in 1999 and after many tries, it finally became available on 1 December 2015 in Slovenia. eCall will automatically dial 112 when the information received from the car contains data regarding airbag opening and the impact and

[1] More statistics on http://asirt.org/Initiatives/Informing-Road-Users/Road-Safety-Facts/Road-Crash-Statistics
[2] More about the eCall system on https://ec.europa.eu/transport/road_safety/specialist/knowledge/esave/esafety_measures_unknown_safety_effects/ecall_en

the person answering the call can hear and talk to the passengers. But what about deaf or people with hearing impairments? There are also various phone applications available that try to mimic the functionality of such a system, but many of them use only an accelerometer to detect the accidents and sends emergency notifications to pre-selected contacts. But what about if you drop the phone? Will that be considered a car accident?

ACCIDeNtS tries to do more. Comparing to other systems and applications, the system proposed and developed by me consists of both the physical part connected directly to the car and the software part where the algorithms and the application work together to collect and interpret various data from the car.

The main and most important purposes of the proposed project is to save lives in case of accidents that occur in a remote location where the help might take longer to come. Also, it demonstrates the easiness of designing and implementing such a system for anyone that wants to change the chilling statistics presented in the beginning.

I chose this theme because it combines my passions: computer science, cars and helping people. Also, I wanted to put in practice some of the things I've learnt so far in informatics and combine them with the hardware and experiment new things. The things I learnt throughout the development of this project are communicating via AT commands, requesting and receiving data from the car's computer, Django and fuzzy logic, things which cover almost all the project. My main contributions to this thesis is combining these technologies that apparently could not work together to obtain an end to end fully functional system.

The current paper license is composed of two parts: the theoretical part regarding the hardware and the software and the practical part where the implemented project is detailed. Each chapter contains a subchapter presenting its purpose. The first chapter, ACCIDeNtS – A THEORETICAL THINKING, presents the technologies used and implemented in the project. Besides its introduction, it is composed of two subchapters, one detailing the hardware part where the modules and the infrastructure are described and one corresponding to the software part of the project. The second chapter, ACCIDeNtS – FROM THEORY TO PRACTICE, presents more details about the implementation of the project as well as its parts and flows. The in-car component subchapter details the first part of the project, more exactly the receiving of the data from the car, processing and transmitting it to the external component detailed in the second subchapter. The developed application is then presented in the third subchapter.

# 1 ACCIDeNtS – A THEORETICAL THINKING

## 1.1About

This chapter contains a brief presentation of the technologies used for designing and implementing ACCIDeNtS. Here are presented all the parts of the project, both the hardware and the software.

## 1.2Hardware

The hardware components of the proposed system play a very important role in its smooth and complete functioning, being the base of the project. There are mainly two parts consisting of different components linked together in order to communicate with and deliver correct data to the software. In order to obtain real and valid data about the car accident, it must be taken directly from the source, more specifically from the car's computer.

In the automotive industry, the need of connecting to the car and obtaining data from it started before the 1990s when the technology in this field really developed. From that, many car manufacturers developed their own connection tool. The main development center of this technology was California were the need of those systems appeared due to the need of designing reliable emission control systems. For this, in 1991 a basic standard called OBD I (or On-Board Diagnosis) was implemented but, although it served the needs of those times, it had limited capabilities once the industry really began developing. For this reason, California Air Resources Board (or CARB) proposed in 1994 a more advanced standard, known as OBD II. The OBD II computer system became mandatory for the models built in 1996 but it is still in use today because of the variety of data obtained. The connection to this system is made through the Data Link Connector (or DLC), which is a 16-pin connector located under the dash of the car. While every system is the same, there are slightly variations depending on the car manufacturers. Known as protocols, in the automotive industry nowadays there are 5 basic signals protocols in use: SAE J1850 PWM, SAE J1850 VPW, ISO 9141-2, ISO14230-4 and ISO 15765-4/SAE J2480 (or more commonly known as CAN-BUS). To establish the type of

communication protocol used by a car, it is enough to determine which pin is populated in DLC, as seen in Figure 1 - OBD II Data Link Connector and the specific protocols below.

| OBD2 Pin | Description |
|---|---|
| 2 | SAE J1850 Bus + |
| 4 | Chassis |
| 5 | Signal Ground |
| 6 | CAN High |
| 7 | ISO9141 K Line |
| 10 | SAE J1850 Bus - |
| 14 | CAN Low |
| 15 | ISO9141 L-Line |
| 16 | Vehicle Battery Positive |

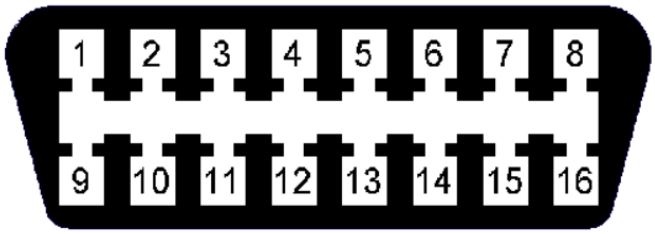


*Figure 1 - OBD II Data Link Connector and the specific protocols*

The communication to the car is made through PIDs (or Parameter IDs) which are binary codes used to communicate to the car.

For this type of communication there are 10 modes of operation described in the SAE J1979 standard, each one being in charge with different types of data. For the ACCIDeNtS project, the used mode is '01' which returns the current data of the car. A basic request command to the car for retrieving a specific data is 01 XX, where XX represents the byte corresponding to the needed data. In our project's case, the data taking into consideration for analyzing is:

- '0C' – Engine RPM, with a value between 0 and 16,383.75 rpm;
- '0D' – Vehicle speed, with a value between 0 and 255 km/h;
- '46' – Ambient air temperature, with a value between -40 and 215 degrees Celsius;
- '5C' – Engine oil temperature, with a value between -40 and 215 degrees Celsius.

For these requests, the car computer responds with byte codes of the form AA BB BB CC DD EE (one or more bytes, depending on the data) FF. The most important bytes are CC DD and EE. The first decimal of the CC byte is 4, meaning the fact that the array of bytes is a response to a query for a mode defined by the second decimal of the byte. DD is the byte that specifies which data was requested (see the XX byte form the request command above). EE (one or more bytes) represent the actual data from the response of the car. For the speed, ambient air temperature and engine oil temperature, the response consists of one byte and for the RPM there are 2 bytes.

There are many tools regarding car diagnosis and data interpreting, but their complexity varies based on the usage. Car services and specialized places for diagnosis use very expensive tools to find and fix critical car issues but for common usage and DIY projects,

alternative and more cheap tools became available on the market, such as ELM 327 that comes in three connection possibilities: Bluetooth, WiFi and USB. For the proposed project, I chose the Bluetooth type because although this device is commonly used with laptops or smartphones, one of the main advantages is that Bluetooth is a universal way of communicating, supported by many components. Also, other advantages of using ELM 327 are the fact that is compatible with all petrol cars manufactured since 2000 and diesel one manufactured since 2004, no matter the protocol used and that it offers a variety of data, such as RPM, the temperature of the cooling liquid, the speed of the vehicle, the state of the power supply system and many more. ELM327 tool has at its base a PIC microcontroller (or Peripheral Interface Controller), a very small controller consisting of electronic circuits programmed to perform different tasks. Its architecture consists of CPU, RAM, ROM, timers, counters and protocols used for interfacing with other peripherals.

Regarding ways of communicating, as said before, the main chosen way of transmitting the data from the car to the Raspberry Pi is by Bluetooth. Bluetooth is in many situations preferred as a way of transmitting data because of the easy way of working, low complexity and low cost, advantages very useful in my project's case.

In order to obtain and process all the data coming from the car, it is not so important the reaction speed but the power to do more complex calculus and run multiple programs. For this main reason, an excellent candidate when it comes to boards is Raspberry Pi. Raspberry Pi is a small general-purpose computer that is usually based on a Linux operating system. Designed for teaching computer science in schools, this board developed in time, being one of the most important components in many projects and applications. Today, there are several models available on the market, each one with different functionalities but very similar. First generations of Raspberry Pi (Raspberry Pi 1 Model A, Model A+, Model B and Model B+) were introduced to the public in 2012 followed by Raspberry Pi 2 released in 2015 and by Raspberry Pi Zero designed to be a smaller Model B. The most complex model of Raspberry Pi is by far The Raspberry Pi 3 Model B, released in early 2016. It features a 1.2 GHz quad core ARM Cortex-A53 CPU, a 300MHz increase in CPU speed over Raspberry Pi 2. Also, the Model B has many new and upgraded functionalities regarding the connectivity, being loaded with 802.11n WiFi and Bluetooth 401 and USB boot capabilities. Because of these upgrades, Raspberry Pi Model 3 became a perfect candidate for sustaining the in-car component of the ACCIDeNtS project.

For the ACCIDeNtS project it is critical for the location of the accident to be known and very precise in order for the units to be sent. For this, it is obvious that GPS coordinates are mandatory.

So far, our project is able to get the data from the car and process it. But how will the output of the processing be transmitted forward?

For this problem I chose to use the GSM standard (or Global System for Mobile communication) because I wanted to make the system as remote as possible. And because some accidents might happen when there is no internet connection, one of the best ways of sending data is by an SMS. On the market there are various modules for GPS and GSM, and some even combine them. Such a system is SIM808 from SIMCOM, a complete Quad-Band GSM/GPRS module combining GPS technology. The module is controlled based on AT commands (or ATTENTION commands), used as a prefix for other parameters in a string. There are two types of such commands: parameter type commands and action type commands, the main difference between them being the purpose of the usage. Syntactically, the AT commands can be split into three categories: basic, S parameter and extended. The base syntax of the commands consists of the AT prefix followed by the command and <CR> at the end. The commands are usually followed by a response of the form <CR><LF>response<CR><LF>. For the basic ones, the format is AT<x><n>, where <x> is the command and <n> are the argument(s) for the command. For S parameter syntax, the commands are of the form ATS<n>=<m>, where <n> is the index of the S register to set and <m> is the value to assign to it. For the extended case there are multiple syntaxes based on the type of the command: AT+<x>=? for testing, AT+<x>? for read, AT+<x>=<…> for write and AT+<x> for execution.

The SMS send from the car is received on the other hardware part of the system where another GSM module is connected this time to an Arduino board. For this, I chose to use also a SIM808. The Arduino board is by far one of the most used micro-controllers on the market, because of its many models that can adapt to a large variety of projects. It was designed to be very fast but with limited processing capacity. The need for the Arduino board here is because the received data transmitted from the accident must be quickly interpreted and displayed in the application, the time being a direct factor in case of a life and death situation.

## 1.3Software

We saw that the hardware is very important for a good functioning of the proposed system, but the sent or received raw data is not so useful if it is not processed and interpreted. For this and many more, the responsible is the software part. In the ACCIDeNtS project, the software plays a role both on the in-car system and on the external system, consisting of algorithms for processing the data from the car, interpreting and classifying, an application to display and manage the accidents and databases to store the raw and processed data.

In the car, after successfully pairing the ELM327 tool with the Raspberry Pi and getting data from the car, the system starts to periodically analyze it and compare it with the previous one in order to detect if an accident occurred. But when there are so many variables and when the data is related to multiple fields, comparing and deciding gets difficult. For this problem, a solution implemented also in the ACCIDeNtS project is by using fuzzy logic.

Fuzzy logic is a many-valued logic where the truth values vary between 0 and 1. More exactly, a situation cannot be just true or false as the Boolean logic works, but can be partially true, partially false, in between or both at the same time.

Fuzzy logic was studied since the 1920s as an infinite-value logic, resembling human reasoning and over the years it has been applied to many different fields, the most important being artificial intelligence. The architecture of the fuzzy logic system is made up of 4 parts: Fuzzification Module, Knowledge Base, Inference Engine and Defuzzification Module. The data from the database which are crisp numbers (basic numbers) is firstly converted to fuzzy sets in the Fuzzification Module. Here, it is split into five steps as follows: LP – large positive, MP – medium positive, S – small, MN – medium negative and LN – large negative. The number resulting are called fuzzy numbers, fuzzy sets with different closeness degree to the given crisp number. The Knowledge Base stores the IF-THEN given rules by

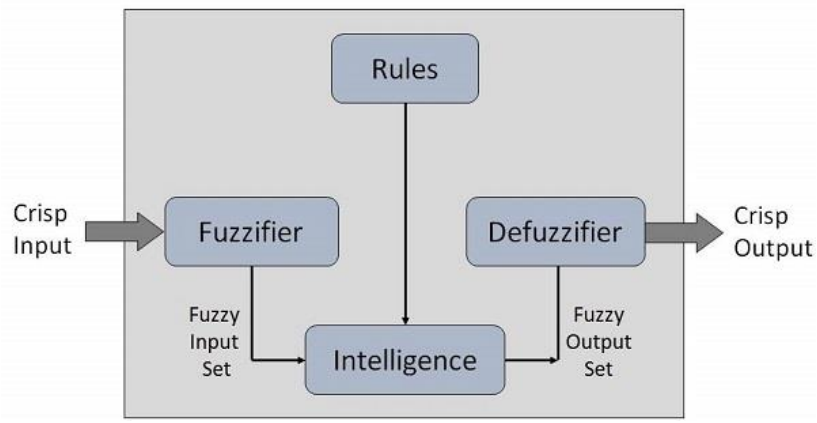


*Figure 2 – The architecture of the fuzzy logic system*

the programmer which are used by the Inference Engine. This engine applies logical rules to the knowledge base in order to obtain new information, simulating the human reasoning process. The Defuzzification Module transforms the obtain set of data back into crisp numbers. One of the most important advantages of fuzzy logic is the fact that can work with imprecise and noisy input data in case it is slightly altered on its way from the car to the system. Also, it is very flexible due to the fact that new rules can be added anytime and old ones can be removed and easy to be understood.

In order to process the data using fuzzy logic on the Raspberry Pi, I used the skfuzzy toolbox. Skfuzzy or SciKit-Fuzzy is a collection of python algorithms which implements fuzzy logic, developed by the SciPy community. It features an impressive variety of tools and functions, very useful for computation regarding projects that involve such kind of logic, like in ACCIDeNtS's case.

After the accident is identified, the SMS from the car is sent and received in the second component of the project where the transmitted data are stored in a database from which the main application takes it and displays it to the person responsible for sending the units. In order to be smart, fast and simple, the application was designed using the Django web framework.

Both for the Raspberry Pi and for the Django application, I used SQLite3 databases because there are easy to use, flexible, compatible with a large number of technologies and reduced in costs.

# 2 ACCIDeNtS – FROM THEORY TO PRACTICE

## 2.1 About

In this chapter, the ACCIDeNtS project and the way the components work together are more detailed. It specifies the steps the data makes in order to get from the car to the application and the adjacent tools that work together to process, analyze and get it there.

## 2.2 In-car component

### 2.2.1 From the car to the Raspberry Pi

The first connection of the whole flow starts in the car where the OBD II module exchange commands/data with the Raspberry Pi by Bluetooth. For this, an initial step is to enable DBus which is a message-bus system that creates sockets between applications and daemons. Further, another useful tool when it comes to connecting via Bluetooth is Bluez[3], especially in the case when a graphical interface is needed. If the Bluetooth works as expected, when scanning for finding devices, the OBD-II should be available for pairing.

After successfully pairing the components, the Raspberry Pi will start periodically to send commands to the module for asking about the results of the required measurements. As an example, lets analyze a simple request-response case. For querying the car about its speed, the following command must be sent: 01 0D. 01 represents the mode and 0D is the code for speed. A basic response for this command is 83 F1 12 41 0D 00 D4. 83 represents the ECU address, F1 12 represents the protocol, 41 consists of the response code (4) and the mode for which the response is made (01), 0D is the command to which the response is addressed, 00 represents the speed (0 km/h in our case) and D4 the checksum. Beside speed, the RPM, air temperature and oil temperature are also asked for. After the module sends the data, it is firstly stored in the database.

### 2.2.2 Inside the module: processing

The data is then extracted from the table by a script which then identify the bytes corresponding to the values requested, transforms them into readable data and the results are

[3] Official Linux Bluetooth protocol stack (http://www.bluez.org/)

inserted in another table from the same database described in Appendix 1, as it can be seen in Appendix 2. Whenever a new set of data is received and processed, from the database the oldest data will be deleted in order to avoid overflowing the memory. From there, the data is processed and classified in order to look for potentially signs of an accident.

For the example discussed in the previous subchapter, the first thing the script is doing is verifying that the array is a response for mode 1, the only mode used in this project. 41 satisfies the test, so it further verifies if the response is related to the request by comparing the fifth byte of the response with the second byte of the request made. Because 0D from our response corresponds to 0D from the request, the response is valid and the byte corresponding to the needed data (which in our case is 00) is extracted and converted. 0km/h will be inserted in the database among with the response for RPM, air temperature, oil temperature and a timestamp.

At each step, two entries are available in the database, both in the table of raw codes and in the one of processed codes. When another code is inserted, the oldest entry is deleted. In this way, we firstly assure that it will not run out of memory and secondly, the new data can always be compared with an older one. Every time a new set of codes is converted and inserted in the database, an algorithm will compare the data with the previous one to detect a potential accident. This algorithm uses fuzzy logic to verify and, if it is the case, to classify the accident in 3 classes of priority:

- Class '0' – small accident, where there is a difference in speed and RPM
- Class '1' – medium accidents, where the difference is increased
- Class '2' – serious accidents, where there is a very big difference of speed and RPM indicating the fact that the car was going very fast and stopped suddenly and also an increase air and oil temperature, meaning that the car might be on fire.

For the fuzzy logic algorithm succinctly described in Appendix 3, I used the skfuzzy[4] toolbox. The priority of the accident is considered a consequent measure because it represents the output of the program while the speed, rpm and the temperatures are considered antecedent ones, being the input. We then generate the fuzzy membership functions using trimf and create the rules for classifying the accident. If the difference between the previous and the current data is important (after testing, the following intervals were taken into consideration: speed bigger than 50 km/h, RPM bigger than 1500 rpm), using these rules, we calculate the

[4] Fuzzy Logic toolbox for SciPy : https://pythonhosted.org/scikit-fuzzy/

priority given by the input data and round it to an integer. A problem encountered at this step was the installation of skfuzzy tool on the Raspberry Pi because it had dependencies in other packages that needed upgrades. Because it didn't work upgrading all at once, I had to manually upgrade each package, clone the repository of the skfuzzy tool and install it from there.

If an accident was detected, the system will then collect GPS coordinate and will automatically send an SMS to the center containing the calculated priority and the coordinates.

### 2.2.3 Data sending via SMS

The circuit responsible for collecting the location and sending the SMS with data is described in the Figure 3 – Connecting the Raspberry Pi Model B to SIM808 below.

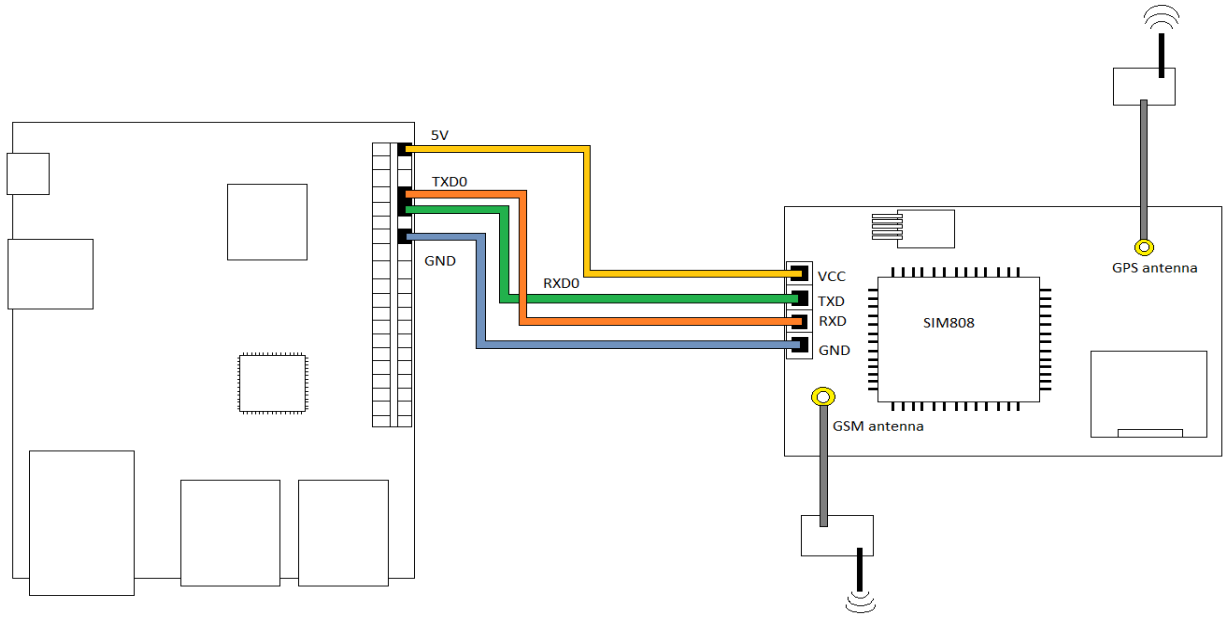


*Figure 3 – Connecting the Raspberry Pi Model B to SIM808*

The SIM808 is powered from the Raspberry Pi at 5V and connected to the GND (ground) of it. For serial communication, TXD0 and RXD0 from the Raspberry Pi are connected to RXD and TXD from the GSM/GPS module. The data byte is sent via TXD0 pin which is the output of one of the serial devices on the Raspberry Pi. Through this pin, the byte is serially shifted at the baud rate which is specified by the user. The form of the transmitted frame and the size in bits of each part can be seen in Figure 4 – Transmitted frame between TXD0 and RXD and between RXD0 and TXD.

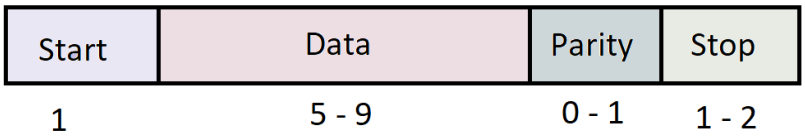


*Figure 4 – Transmitted frame between TXD0 and RXD and between RXD0 and TXD*

In our circuit, TXD0 from the Raspberry Pi is connected to RXD of the SIM808 module in order for the sent bytes to be received by the module. RXD0 is the opposite of TXD0, the serial peripheral listening for the data received from the TXD serial of the module and shifting it to the device.

The first thing to do is to enable the serial communication defined on port /dev/ttyS0 on 115200 baud rate. The communication is made via AT commands described in the previous chapter which are transmitted to the modem that responds with various codes or errors. An example of code can be found in Appendix 4. After the beginning 'AT' command, the signal is checked using 'AT+CSQ'. We successfully connected to the GSM function, but in order to report the accident we need the location from the GPS. For this, we first turn on the power supply for the GPS by sending 'AT+CGPSPWR=1'. We then check the status of the GPS with 'AT+CGPSSTATUS?' and can successfully retrieve the coordinates with 'AT+CGPSINF=0', coordinates which are saved into a variable. We set the GSM modem to SMS text mode with 'AT+CMFG=1' and select the procedure for message reception from the network by 'AT+CNMI=<mode><mt><bm><ds><bfr>', where <mode> controls the processing of unsolicited result codes, <mt> sets the result code indication routing for SMS-DELIVERs, <bm> set the rule for storing received Cell Broadcast Message types depend on its coding scheme, <ds> is used for SMS-STATUS-REPORTs and <bfr> which is 0 as default. In our case, the command looks like this: 'AT+CNMI=2,1,0,0,0'. We then specify the phone number to which the SMS will be sent via 'AT+CMGS="+40XXX-XXX-XXX"', set the message containing the coordinates and the priority of the accident and then sent it by sending a command containing just a new line character. For this part, one of the problem encountered during the implementation was capturing the response of the modem at the right time but this aspect was solved by reading each time each response of the commands.

The SMS is sent to the specified number from the code which actually refers to the SIM inside the GSM module connected to the Arduino board from the external component.

## 2.3 The external component

### 2.3.1 New accident - receiving the data

The SMS sent in the previous subchapter is received by the SIM inside the GSM module from the external component, component formed by the module and the Arduino.

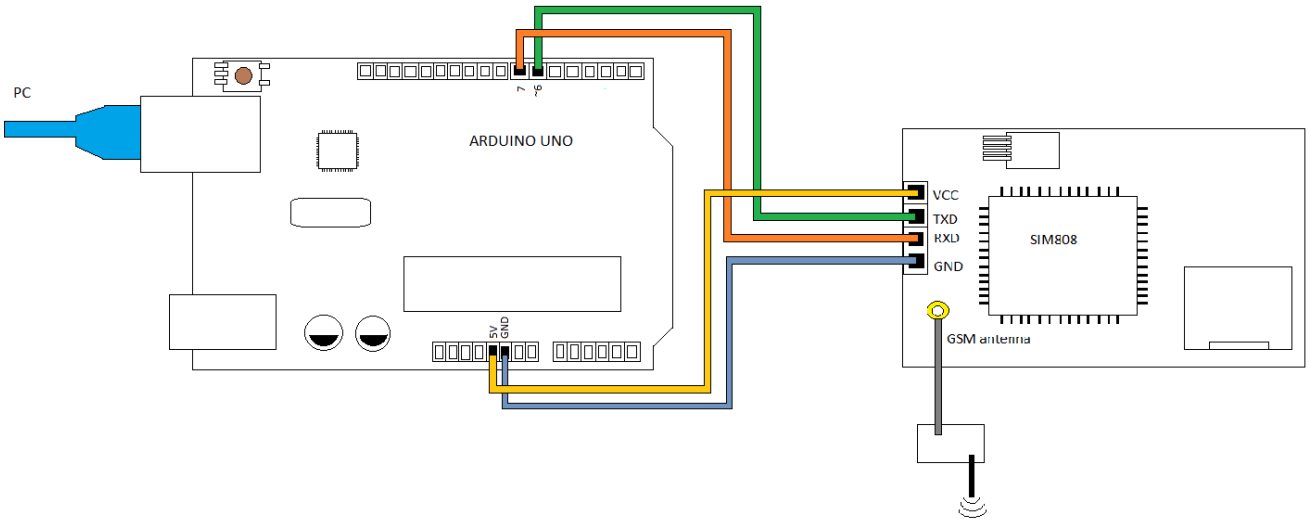


*Figure 5 – Connecting the Arduino Uno to SIM808*

The connection between the Arduino and the SIM808 is similar to the one from the other component. The SIM808 module is powered from the 5V of the Arduino and connected to one of its GND. The TXD pin of the GSM module is connected to pin 6 of the Arduino which acts as its RX and RXD is connected to pin 7 which is considered the TX of the Arduino. The Arduino is powered from and connected to the PC through a USB A/B type cable.

### 2.3.2 Arduino: from an SMS to database

The main role of this component is to listen for SMSs related to accidents and when one is received, to capture the data and insert it into the database of the Django application. The first program of this part exemplified in Appendix 5 is uploaded into the Arduino. It consists of an initial part in the setup() of connecting to the SIM from the module and a second part in the loop() function where it waits for an SMS. The form of a received SMS is +CMT: "+407XXXXXXXX","","26/06/18, 23:57:21" 1 47.5268 26.5789. At this point I faced a really challenging problem because I has to connect the Serial of the Arduino with the application's database. What I considered to be a good solution to this problem was to write

the data from the Arduino into a file and then read it through a script, connect to the database and insert it there. But writing in a file placed on the PC from the Arduino is not possible so after a research in this area, I came across a program that does exactly this. On the port used for connecting the PC to the Arduino (COM6 at 9600 baud rate in my case) I also connected an application called CoolTerm[5] which listens for new received data. From the AT response from above, if the phone number is the expected one, the priority and the coordinates are selected and sent to the Serial, data which is also detected by the CoolTerm application and captured in a file which is updated every time a new data is received. From this file, the data is read and inserted into the Django database. After the connection between the python script and the Django database is made, the first one reads the first SMS received and remains in an infinite loop to wait for others by continuously comparing the previous list of SMSs with the one read. If a new SMS is received, a new Incident() model is created, the data is set and the entry is saved. In this way, the data is inserted in the Django database and is ready to be displayed. Connecting an external script to the database of the application was another problem because there are separate systems but setting DJANGO_SETTINGS_MODULE resolved it.

## 2.4The Application

### 2.4.1 Data: from database to map

The web application of the ACCIDeNtS project is designed for informational purposes and can be used to track the incoming accidents, sort them by priority and send the closest units to check them. The main page of the application displays the world map and the accidents based on the coordinates from the database. The second page contains the accidents sorted by their priority, with their ID and the timestamp of the accident. For further details regarding the coordinates and some of the closest units, the user can press the button related to the specific accident and the information is displayed in an iframe.

Because the application is based on Django framework, its structure is organized and easy to be understood. The database is constructed in models.py where the Incident model is described: it contains an id which plays the role of a primary key, the timestamp of the received message in timestamp column, the coordinates under latitude and longitude columns and the priority of the accident under the accidentsPriority column. The migrations which are

[5] CoolTerm is a serial port terminal application: http://freeware.the-meiers.org/

basically considered as versions of the database can be found inside the migrations subfolder and although they are automatic, some cases require manual processing.

The URLs of the pages are described in the urls.py file where for each designed URL it is specified the name of the function from the views.py file responsible for it. These functions return an HttpResponse specifying that for the request received as input it is necessary a context described inside that function. The contexts vary based on the needed data on the specific page and represents the entries from the database that can be ordered or filtered as needed.

The HTML files responsible for the design of the pages can be found inside the templates subfolder. For displaying the data, I chose to use on the main page the Google Maps API as described in Appendix 7. For this, in a function, I specified first the coordinates of the center of Romania in order for the map to be centered and to display the entire country in order for the accidents to be seen easily. Then, for all the incidents in the database, a marker is created and displayed on the map based on its coordinates. The function is then called in a script that uses a mandatory API KEY[6] in order to connect and display the map.

## 2.5Testing the system

Every system that involves human lives must be tested in order to identify flaws or errors in calculating the data or how the syncing of the components is made. Also, it is very important to precisely calibrate the system in such a way that it reflects the real live scenarios. After successfully completing the flow of the project, it was time to test it in real life.

The car used for testing is a Skoda Octavia 2, model from 2009, 1.9 TDI. The tests were performed on a flat portion of the road and the data used in the fuzzy logic for detecting an accident was set to 60 km/h for the speed and 1500 RPM for rpm. For each test a specific speed or RPM was reached then suddenly breaking mimicked the occurrence of an accident or a false alarm. The results of the ACCIDeNtS system are presented in Appendix 8. As it can be seen, the system was accurate in proportion of 100%. The GPS location was also correctly calculated in a proportion of 94%, with an error range between 300 m and 850 m.

[6] An API KEY for Google Maps can be easily obtained from https://developers.google.com/maps/documentation/javascript/get-api-key

# CONCLUSION

ACCIDeNtS is the system which proves that some well-chosen components and good technologies can lead to very important results. Designed to be an independent system, this project may lead to a new product in its domain because of its flexibility, cheap but very effective components and the easiness of implementing. Using everyday technologies such as GSM, GPS and Bluetooth and combining them with performing modules and simple software, ACCIDeNtS can really help saving lives. I consider ACCIDeNtS an awareness raised to the automotive industry because it shows that with the help of some research, everyone can contribute to a smarter way of detecting and resolving accidents.

The project can and should be modified in order to increase its capacity for saving passengers. A suggestion regarding the data to be taken into consideration from the car could be also the airbags, which is a direct consequence of an accident. ELM327 does not have access to the Airbag system but some other tools do. Other sensors like pressure sensors and a gyroscope can also help to a more precise identification of a situation where an accident took place. Besides fuzzy logic, machine learning and artificial intelligence might be use to "learn" the habits of the driver when driving so the data from the car to be analyzed with respect to human factors.

ACCIDeNtS is just the beginning, just a proposal, but can be the base of a smart and independent system. We just have to try new things, because "any time you create a technology that has the potential of saving 20 to 30 thousand lives in a year, one has to sit up and take notice"[7].

Many thanks to Bucur Irinel Ion and Crăciun Vlad for providing a significant part of the hardware components used in this project.

[7] Raj Reddy

# BIBLIOGRAPHY


Adrian Holovaty, J. K. (2009). *The Definitive Guide to Django: Web Devlopment Done Right.* Apress.

Barnes, R. (n.d.). *The Official Raspberry Pi Projects Book.* London: Liz Upton.

Bartlett, D. (2008, March). *The future of GPS.* Retrieved from Ingenia Online: http://www.ingenia.org.uk/Ingenia/Articles/473

Can Pan, J. G. (2014, December 22). Modeling and Verification of CAN Bus with Application Layer using UPPAAL. *Electronic Notes in Theoretical Computer Science, volume 309*, pp. 31-49.

Dennis, A. K. (2013). *Raspberry Pi Home Automation with Arduino.* Birmingham: Packt Publishing Ltd.

*ELM Electronics.* (2017). Retrieved from www.elmelectronics.com: https://www.elmelectronics.com/wp-content/uploads/2017/01/ELM327DS.pdf

Hanzo, D. L. (2008, August 04). *Global system for mobile communications (GSM) (University of Southampton).* Retrieved from Scholarpedia: the peer-reviewed open-access encyclopedia: http://www.scholarpedia.org/article/Global_system_for_mobile_communications_(GSM)

Kristian Ismail, A. M. (2015, April). Design of CAN bus for research applications purpose hybrid electric vehicle using ARM microcontroller. *Energy Procedia, volume 68*, pp. 288-296.

Puy, I. (2008). *Bluetooth.* Furtwangen im Schwarzwald.

Smith, A. G. (2011). *Introduction to Arduino - A piece of cake!* CreateSpace Independent Publishing Platform.

team, T. s.-i. (2016). *The scikit-fuzzy Documentation Release 0.2.*

# APPENDIX

## 1. Appendix 1

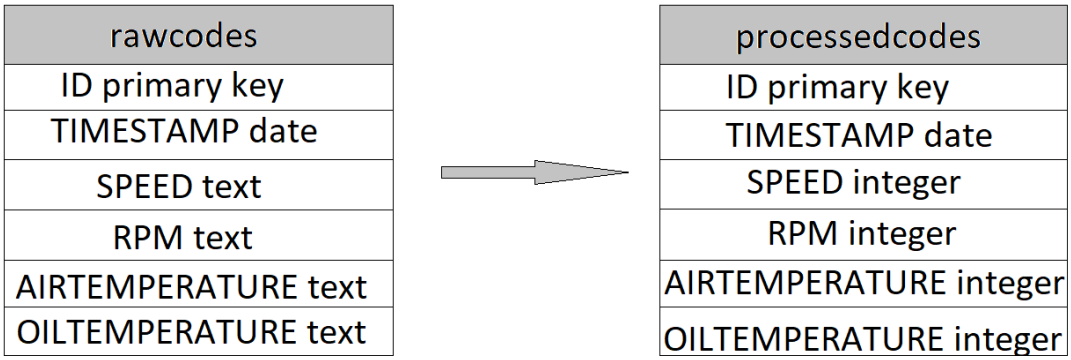


*Figure 6 - Database on Raspberry Pi*

## 2. Appendix 2

```
connection = sqlite3.connect('codes.db')
cursor = connection.execute('SELECT id, currenttime, speed, rpm, airtemperature,
oiltemperature FROM rawcodes') # select the entire entry
for row in cursor:  # for the raw code from the output of the query, the data is
extracted, the bytes are extracted and transformed
    speed = str(row[2]).split()
    speedToBeInserted = int(speed[5], 16)
    rpm = str(row[3]).split()
    rpmToBeInserted = int(rpm[5] + rpm[6], 16)
    airtemp = str(row[4]).split()
    airtempToBeInserted = int(airtemp[5], 16)
    oiltemp = str(row[5]).split()
    oiltempToBeInserted = int(oiltemp[5], 16)
     # the real data is inserted into the processedcodes table
    connection.execute("INSERT INTO processedcodes(speed, rpm, airtemperature,
oiltemperature) VALUES(" + str(
    speedToBeInserted) + ", " + str(rpmToBeInserted) + ", " +
str(airtempToBeInserted) + ", " + str(
    oiltempToBeInserted) + ")")
    connection.commit()
```

## 3. Appendix 3

```
# setting the limits of the input/output variables
speed = ctrl.Antecedent(np.arange(0,255,1), 'speed')
rpm = ctrl.Antecedent(np.arange(0,5000,1), 'rpm')
airtemp = ctrl.Antecedent(np.arange(-40,215,1), 'airtemp')
oiltemp = ctrl.Antecedent(np.arange(-40,215,1), 'oiltemp')
priority = ctrl.Consequent(np.arange(0,3,1), 'priority')

priority.automf(3)
speed.automf(3)
rpm.automf(3)
airtemp.automf(3)
oiltemp.automf(3)
```

```
priority['0'] = fuzz.trimf(priority.universe, [0,0,1])
priority['1'] = fuzz.trimf(priority.universe, [0,1,2])
priority['2'] = fuzz.trimf(priority.universe, [1,2,2])
# creating the rules for the priotity codes
rule1 = ctrl.Rule(speed['good'] & rpm['good'] | airtemp['good'] |
oiltemp['good'], priority['2'])
rule2 = ctrl.Rule(speed['average'] & rpm['average'], priority['1'])
rule3 = ctrl.Rule(speed['poor'] & rpm['poor'], priority['0'])
# computing the priority
prioritizing_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])
prioritizing = ctrl.ControlSystemSimulation(prioritizing_ctrl)
prioritizing.compute()
```

## 4. Appendix 4

```
GPIO.setmode(GPIO.BOARD)
port = serial.Serial("/dev/ttyS0", baudrate=115200, timeout=1)
port.write('AT'+'\r')
port.write('AT+CSQ'+'\r')
port.write('AT+CGPSPWR=1'+'\r')
port.write('AT+CGPSINF=0'+'\r')
s= port.readline()
data = s.split(",")
port.write('AT+CMGF=1'+'\r')
port.write('AT+CNMI=2,1,0,0,0'+'\r')
port.write('AT+CMGS="+40752700802"'+'\r')
port.write(str(priorityOfTheAccident)+" "+str(data[1])+" "+str(data[2])+'\r')
port.write("\x1A")
```

## 5. Appendix 5

```
# setting the RX and TX pins of the Arduino
SoftwareSerial gsm(6,7); // RX, TX

#the setup function is executed at the start of the program when it connects to the
SIM card
void setup() {
  Serial.begin(9600);
  gsm.begin(9600);
  gsm.println("ATE0");
  gsm.println("AT");
  gsm.println("AT+CMGF=1");
  gsm.println("AT+CNMI=1,2,0,0,0");
}

# the loop function constantly listens for SMSs received
void loop() {
  if(gsm.available()) {
    Grsp = gsm.readString();
    Serial.println(Grsp.substring(48));
 }
}
```

## 6. Appendix 6

```
# connecting to the Django database
Sys.path.append("C://Users//Alexandra Citea//Desktop//Licenta//accidents")
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "accidents.settings")
django.setup()
application = get_wsgi_application()

# reading the first line from the file
file_msg = open("C://Users//Alexandra Citea//Desktop//Messages.txt", "r")
previous_line1 = file_msg.read().splitlines()
previous_line=[]

for elem in previous_line1:
if elem != "":
              previous_line.append(elem)
file_msg.close()
while True:
       # each time a new SMS is received and a new set of data is inserted in the
file, the entry is inserted in the database of the application
       file_msg = open("C://Users//Alexandra Citea//Desktop//Messages.txt", "r")
       line1 = file_msg.read().splitlines()
       line = []
       for elem in line1:
              if elem != "":
                  line.append(elem)
       if previous_line != line:
              previous_line = line
              new_accident = line[-1]
              print("New accident info: "+new_accident)
              elements = new_accident.split(' ')
              model = Incident()
              model.timestamp = timezone.now()
              model.accidentsPriority = elements[0]
              model.latitude = elements[1]
              model.longitude =elements[2]
              model.save()
       file_msg.close()
```

## 7. Appendix 7

```
<script>
function myMap() {
       # the map is centered
  var myCenter = new google.maps.LatLng(45.983611,24.695278);
  var mapCanvas = document.getElementById("map");
  var mapOptions = {center: myCenter, zoom: 7};
  var map = new google.maps.Map(mapCanvas, mapOptions);
       # each accident is displayed on the map
    {% for incident in all_incidents %}
        var pointFromDatabase = new google.maps.LatLng({{ incident.latitude }}, {{
incident.longitude }});
        var marker = new google.maps.Marker({position:pointFromDatabase});
    marker.setMap(map);
    {% endfor %}
}
</script>

<script
src="https://maps.googleapis.com/maps/api/js?key=API_KEY&callback=myMap"></script>
```

## 8. Appendix 8

| Test no. | Max speed | Max RPM | Results | Priority |
|---|---|---|---|---|
| 1 | 40 | 1500 | Not accident | - |
| 2 | 40 | 2500 | Not accident | - |
| 3 | 50 | 1300 | Not accident | - |
| 4 | 50 | 2000 | Not accident | - |
| 5 | 60 | 1000 | Not accident | - |
| 6 | 60 | 1500 | Accident | 0 |
| 7 | 80 | 1500 | Accident | 0 |
| 8 | 80 | 2500 | Accident | 1 |
| 9 | 100 | 2000 | Accident | 1 |
| 10 | 100 | 3000 | Accident | 2 |
| 11 | 120 | 2500 | Accident | 2 |
| 12 | 120 | 3500 | Accident | 2 |