CZECH TECHNICAL UNIVERSITY IN PRAGUE

FACULTY OF TRANSPORTATION SCIENCE

Bc. Lukáš Posekaný

DESIGN OF MULTIDETECTOR FOR INTEGRATION OF VARIOUS TRAFFIC SENSORS

Master's thesis

Prague, 2018

CZECH TECHNICAL UNIVERSITY IN PRAGUE
Faculty of Transportation Sciences
Dean's office
Konviktská 20, 110 00 Prague 1, Czech Republic

K620...Department of Transport Telematics

MASTER'S THESIS ASSIGNMENT

(PROJECT, WORK OF ART)

Student's name and surname (including degrees):

Bc. Lukáš Posekaný

Code of study programme code and study field of the student:

N 3710 – IS – Intelligent Transport Systems

Theme title (in Czech):	**Návrh multidetektoru pro integraci různých dopravních senzorů**
Theme title (in English):	Design of Multidetector for integration of Various Traffic Sensors

Guides for elaboration

During the elaboration of the master's thesis follow the outline below:

- Analysis of merging traffic parameters from multiple sensors and detectors
- Requirements for the design of a Multidetector integrating various traffic sensors and detectors, logical functions over data and computing performance
- Design algorithms for data-handling, saving and distribution
- Choosing an appropriate HW platform for designing and realization of Multidetector
- Analysis and design of suitable tools for simulation or modeling of outputs from traffic sensors and detectors
- Verification and evaluation of the designed Multidetector system.

Graphical work range:	according to the supervizor's instructions
Accompanying report length:	55 pages at minimum (including figures, graphs and tables, which are components of the report)
Bibliography:	Traffic Detector Handbook: Third Edition – volume I and II; Federal Highway Research Center; October 2006 technical documentation and manuals of the selected HW platform and used SW solutions

Master's thesis supervisor: **Ing. Martin Langr, Ph.D.**

Date of master's thesis assignment: **July 25, 2017**
(date of the first assignment of this work, that has be minimum of 10 months before the deadline of the theses submission based on the standard duration of the study)

Date of master's thesis submission: **November 30, 2018**
a) date of first anticipated submission of the thesis based on the standard study duration and the recommended study time schedule
b) in case of postponing the submission of the thesis, next submission date results from the recommended time schedule

v.z. [illegible]

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE Fakulta dopravní
L. S.

Ing. Zuzana Bělinová, Ph.D.
head of the Department
of Transport Telematics

[illegible]

doc. Ing. Pavel Hrubeš, Ph.D.
dean of the faculty

I confirm assumption of master's thesis assignment.

[illegible]

Bc. Lukáš Posekaný
Student's name and signature

Prague ....May 30, 2018

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta dopravní
děkan
Konviktská 20, 110 00 Praha 1

K620.. **Ústav dopravní telematiky**

ZADÁNÍ DIPLOMOVÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení studenta (včetně titulů):

Bc. Lukáš Posekaný

Kód studijního programu a studijní obor studenta:

N 3710 – IS – Inteligentní dopravní systémy

Název tématu (česky): **Návrh multidetektoru pro integraci různých dopravních senzorů**

Název tématu (anglicky): Design of Multidetector for Integration of Various Traffic Sensors

Zásady pro vypracování

Při zpracování diplomové práce se řiďte osnovou uvedenou v následujících bodech:

- Analýza sběru dat dopravními senzory s ohledem na jejich odlišné principy.
- Formulace požadavků pro návrh multidetektoru integrující různé dopravních senzory, logické funkce pro práci s daty i jeho výpočetního výkonu.
- Návrh algoritmů pro práci s daty, jejich integraci, ukládání i další distribuci.
- Volba vhodné HW platformy pro návrh a realizci multidetektoru.
- Analýza a návrh vhodných nástrojů pro simulaci či modelování výstupů dopravních senzorů a detektorů.
- Ověření a zhodnocení navrženého systému multidetektoru.

ČVUT
FD

Rozsah grafických prací:	dle požadavků vedoucího práce
Rozsah průvodní zprávy:	minimálně 55 stran textu (včetně obrázků, grafů a tabulek, které jsou součástí průvodní zprávy)
Seznam odborné literatury:	Traffic Detector Handbook: Third Edition – volume I and II; Federal Highway Research Center; October 2006
	technická dokumetace a příručky zvolené HW platformy
	technická dokumentace a příručky využívaných SW řešení

Vedoucí diplomové práce: **Ing.Martin Langr,Ph.D.**

Datum zadání diplomové práce: **25. července 2017**
(datum prvního zadání této práce, které musí být nejpozději 10 měsíců před datem prvního předpokládaného odevzdání této práce vyplývajícího ze standardní doby studia)

Datum odevzdání diplomové práce: **30. listopadu 2018**
a) datum prvního předpokládaného odevzdání práce vyplývající ze standardní doby studia a z doporučeného časového plánu studia
b) v případě odkladu odevzdání práce následující datum odevzdání práce vyplývající z doporučeného časového plánu studia

[illegible]

Ing. Zuzana Bělinová, Ph.D.
vedoucí
Ústavu dopravní telematiky

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta dopravní
3
L. S.

[illegible]

doc. Ing. Pavel Hrubeš, Ph.D.
děkan fakulty

Potvrzuji převzetí zadání diplomové práce.

[illegible]

Bc. Lukáš Posekaný
jméno a podpis studenta

V Praze dne .... 30. května 2018

Acknowledgment

I would like to thank my thesis advisor Ing. Martin Langr, Ph.D. of the Faculty of Transportation Science. He was always willing to give me an objective advice. His overview of the detectors really helped me to form a relation to this topic and his dissertation thesis helped me to understand importance of this topic and I am glad I could have continued at his work.

I would also like to thank all the members of the Department of Transport Telematics and mostly to those from fifth floor of faculty building on Konviktská street for creating a pleasant environment for me to write this diploma thesis. Thanks to their long-term effort, I was able to test principles of this work on professional devices.

Declaration

Abstract

Diploma thesis deals with the possibilities of sensors and detectors integration for transportation environment. Great effort is placed on defining the requirements on sensors and detectors usable in the Multidetector system. The main part is the Multidetector definition, both from the hardware and software point of view. A system built on the principles of this work would be usable both for long-term and short-term transport surveys and for larger telematic systems. The proposed Multidetector requirements are tested in both laboratory and real-world conditions

Key words

sensor, detector, traffic detector, Multidetector, traffic survey, interface, data integration, data processing.

Abstrakt

Diplomová práce se zabývá zkoumáním možností integrace senzorů a detektorů použitelných v dopravním odvětví. Velký důraz je kladen především na definování požadavků na senzory a detektory použitelné v systému Multidetektoru. Stěžejním bodem je definice Multidetektoru a to jak z hardwarového tak softwarového pohledu. Systém vytvořený dle zásad této práce je využitelný jak pro dlouhodobé a krátkodobé dopravní průzkumy, tak i pro rozsáhlejší telematické systémy. Navržené požadavky na Multidetektor jsou ověřeny v laboratorních i reálných podmínkách.

Klíčová slova

senzor, detektor, dopravní detektor, Multidetektor, dopravní průzkumy, rozhraní, integrace dat, zpracování dat.

Content

List of Abbreviations Used

AC	Alternating Current
ADC	Analog-to-Digital Converter
AI	Analog Input
AO	Analog Output
BCM	Broadcom SOC Channel
BLE	Bluetooth Low Energy
CIB	Common Installation Bus
CPU	Central Processing Unit
DAC	Digital-to-Analog Converter
DC	Direct Current
DHCP	Dynamic Host Configuration Protocol
DI	Digital Input
DIN	Digital Input
DO	Digital Output
GIL	Global Interpreter Lock
GND	Ground
GPIO	General Purpose Input/Output
GUI	Graphical User Interface
HDMI	High Definition Multimedia Interface
HW	Hardware
I2C	Inter-Integrated Circuit
ID	Identifier
IEEE	Institute of Electrical and Electronics Engineers
iOS	iPhone Operating System
IP	Internet Protocol
IR	Infra-Red
LAN	Local Area Network
LAP	Lower Address Part
LCD	Liquid Crystal Display
LED	Light Emitting Diode
NTP	Network Time Protocol
OPC	Open Platform Communication

OS	Operating System
OSI	Open System Interconnection
PC	Personal Computer
PLC	Programmable Logic Controller
RF	Radio Frequency
RPi	Raspberry Pi
SCADA	Supervisory Control and Data Acquisition
SPI	Serial Peripheral Interface
SPZ	Státní Poznávací Značka
SSH	Secure Shell
SW	Software
TCP	Transmission Control Protocol
TLC	Triple Level Cell
UART	Universal Asynchronous Receiver-Transmitter
URL	Uniform Resource Locator
USB	Universal Serial Bus
USBIF	USB Implementers Forum
VCC	Voltage at the Common Collector
VDC	Volts DC
VGA	Video Graphics Array

1 Introduction

1.1 Review of the Topic

A role of detectors in transportation is very significant. They are used as the first node in architecture of most telematic systems and to collect data for researches.

Detectors are built in on sensors, in other words sensors are subset of detectors set. Detectors expand functionality of sensors, convert physical quantities to a data streams, save them and provide data when required.

The purpose of this work is to further understand how these detectors and sensors are or could be constructed and more importantly how they could be implemented into system consisting of multiple detectors of different family. Implementation on physical layer (interfaces, power supply, HW design, etc.), logical layer (data streams, algorithms to integrate data, format data, provide data, etc.) and especially how these two layers affect one another.

All known and searched systems using multiple detectors are fusing data in postprocessing, in other words information about vehicles are stored in individual detectors and after survey there is need for manual integration. [1]

This topic needed to be further researched because in Faculty of Transportation Sciences is often a need to measure traffic data for other thesis, researches and semester works. In many cases, more than one detectors need to be used, and then it gets over complicated to process data and often divert attention from the main topic. Traffic is often counted manually in places where automatic system could be used.

This topic has been chosen to extend long term work on "Traffic Models and Traffic Control" project, where part of effort is heading to detectors topic. Apart from others, important for this thesis are dissertation thesis of Ing. Martin Langr Ph.D. "Fúze heterogenních dopravních dat pro odhadování směrových vztahů" and bachelor thesis of Bc. Lukas Posekany "Processing and Evaluation of Traffic Data from Pico Iteris Videodetection System". This work will be directly linked on these two theses.

In the first mentioned work, there is shown that detectors used in telematics systems can be used for traffic survey. This helped to reduce need of human resources when measuring traffic parameters. Only manual work while survey was to periodically check if detectors are running properly. On the other hand, this work demonstrated, that it is not fully possible with current detectors to integrate data from multiple detectors [2]. This was caused because detectors on the market are "one-piece device", in other words, has their own software, data formats and are not interoperable. If there is possibility to add them into existing systems, these systems are used in telematics and are too expensive and too complexed to be used for survey uses.

The "Processing and Evaluation of Traffic Data from Pico Iteris Videodetection System" was first attempt of integrating detector for telematic systems into other device. It was shown, that we are able to collect data from detectors designed for light controllers. In this work the video detector from Pico Iteris will be mentioned and used in final testing. [3]

1.2 The Aim of the Work

The main aim of this work is to design system capable of collecting data from multiple detectors and sensors at once, process this data, save them and provide to a user. System should be fully automated after initialization on boot.

To design such a system, there is need to analyze existing detectors and sensors, find out how they process a data and what format they are using on output. This will help later with integration.

First, there will be need to format this data into one common format. After these data should be integrated with main algorithm, which should be able to merge, format and save all data from all plugged-in detectors.

Important task will be to include into this algorithm how to specify and work with physical and logical connection (how physical layout of detectors influence logical algorithms). This problem will occurs when there will be multiple detectors on the same observation spot, but each of them will create a timestamp for a car in different location on roadway. Some detectors can even create record of vehicle in several meters long area. Although for some detectors accuracy of timestamp is very important (video detector, induction loop, ...), for others doesn't have to be (in case of vehicle identification like Bluetooth identification or SPZ identification)

Big part of effort will be aiming towards designing a first version of multidetector. Not to create sellable product, but to test all algorithms and interoperability of some chosen detectors. Firstly, device will be tested on laboratory conditions and after. Later, some detectors will be used to measure traffic parameters on real conditions.

1.3 Structure of the Work

Structure of this work will have a logical order following design of a new product. There will be two main parts, theoretical and practical. Both parts will be heading towards research, design, develop prototype and test the multidetector system.

In the first part there will be stated topic, the scope of the topic and mentioned depth of research to this problem. There will be mentioned experiences with detectors and sensors describing the points which leaded to this topic, big part will be covering disadvantages of existing detectors and why they are not suitable for integrating data from multiple devices.

Another part will be mentioning problem with data format and more importantly data integration from multiple sensors or detectors. This will be very important to state before the design of algorithms of data integration which will be next chapter. In this chapter will be described connection between real position of detectors on road and logic of data integration.

In this part of work should be already known all requirements on system of multidetector, so there will be need to summarize all points needed for designing multidetector.

Next step is to summarize what the multidetector is, what it should be able to do and what it needs to be constructed. In this chapter will be selected platform on which whole system will be running.

Second part will be for designing and testing prototype of multidetector. The goal is to test designed algorithms and to create first version of multidetector. This first version doesn't have to meet all points stressed in previous chapters. It should be working product, settable by user on terrain and able to test all algorithms.

To choose proper operating system or other software environment will be aim of the first chapter of practical part. It will be necessary to choose suitable one for selected platform. There will be need to choose proper programming language which will be able to run web server or other graphical user interface, read parameters and set multidetector and communicate with individual detectors. Most of the current programming languages can handle data algorithms and files manipulation so this will not be so important parameter.

The structure of next part is not exactly known because it will depend on chosen multidetector platform, operating system and programming language. It should include methods how to access and control multidetector, graphical user interface, time synchronization and program flow or code structure.

Next big part is reserved for individual sensors and detectors chosen to represent inputs for multidetector. There will be shortly mentioned their physical principle and how it will be implemented into multidetector. More time will be dedicated to data format and implementation into multidetector logic of collecting data. For every detector there will be individually run test and evaluated functionality and use for collecting data.

After stating individual sensors and detectors, all parts should be implemented together. Implementation has to be done on both layers, physical solution of connecting multiple detectors and sensors as well as logical algorithms to synchronize and integrated data, save them and provide in correct format with accurate time.

The last task of this work is to test whole designed system and make conclusions of practical part. There should be mentioned how system behaved in laboratory conditions and how in terrain.

At the very end, both theoretical and practical parts will be concluded together, and all successful and unsuccessful assignments will be summarized here.

2 Sensors Used for Traffic Purposes

There are multiple definitions of what the sensor really is. In English dictionary the sensor is "a mechanical device sensitive to light, temperature, radiation level, or the like, that transmits a signal to a measuring or control instrument." [4] or in official Cambridge dictionary "a device that is used to record that something is present or that there are changes in something." [5].

These are not sufficiently accurate enough and therefore for this work will be used " The fundamental transduction mechanism (e.g., a material) that converts one form of energy into another. Some sensors may incorporate more than one sensor element (e.g., a compound sensor)" [6].

Sometimes a sensor needs to be more complicated device with other parts which helps with processing measured values, in these cases, we talk about a sensor system "A sensor and its assorted signal processing hardware (analog or digital) with the processing either in or on the same package or discrete from the sensor itself.". Both from the [6].

Figure 1.: Sensors Example

There need to be set a boundary between sensor and detector. According to dictionary, the detector is "One that detects, especially a mechanical, electrical, or chemical device that automatically identifies and records or registers a stimulus, such as an environmental change in pressure or temperature, an electric signal, or radiation from a radioactive material." [7].

For purposes of this work there will be stated new definitions suited for its purposes. Sensor is "a mechanical device which is able to detect a change in physical property and provide information of this change in electrical current". Detector is "superstructure of sensor (sensor system), which adds to sensor ability to process information about physical property, transform it into a data streams, store and provide these data to a user"

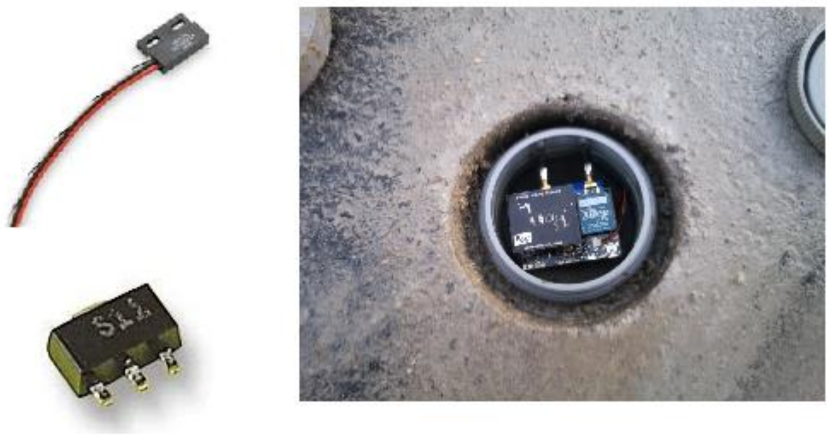


Figure 2.: Magnetic Detector

From the main characteristics table [1], there was few needed for traffic sensors. It is accuracy, distortion, minimum detectable signal, selectivity, sensitivity, threshold, noise, operating range, repeatability and step response. These will be used later in practical part for better understanding of sensors and implementing them into sensor system or detector.

The main logical division of detectors into categories is by energy forms they are working with. The spectrum can be divided into six main energy forms listed in Table 1. From these six forms, for traffic detectors will be important Mechanical, Thermal, Electrical and Magnetic.

Table 1.: Sensor Energy Forms [1]

Energy Forms	**Example Measurands**
Mechanical	Length, area, volume, all time derivatives such as linear/angular velocity, linear/angular acceleration, mass flow, force, torque, pressure, acoustic wavelength and acoustic intensity
Thermal	Temperature, specific heat, entropy, heat flow, state of matter
Electrical	Voltage, current, charge, resistance, inductance, capacitance, dielectric constant, polarization, electric field, frequency, dipole moment
Magnetic	Field intensity, flux density, magnetic moment, permeability
Radiant	Intensity, phase, wavelength, polarization, reflectance, transmittance, refractive index
Chemical	Composition, concentration, reaction rate, pH, oxidation/reduction potential

In traffic, there is need to measure vehicle presence. It could be on one spot or on certain area. All sensors able to capture solid object, movement, heat radiation, sound or presence of metal can be used.

Some of sensors requires more complex system and it would be over complicated for this work to design such, so these will be used in form of detectors or other devices provided by companies. Into this group belongs for example video detection, induction loops, magnetic detectors, microwave and so on.

On the other hand, there are some sensors which are easy to use and can be easy implemented and tested. These are active IR barrier, ultrasonic sensor, simple passive IR detection, Bluetooth tracker.

Every sensor has its own unique mechanism to pass the information about measuring process. Mostly by change in electric current. Working with sensors will require transform this electric current into data streams, to translate information about vehicle into streams of bites the computer will understand. For these purposes, there may be designed device handling interface between sensor and computer or selected such a PLC which will be able to communicate with these sensors.

3 Experiences with Detectors

Detector is enhanced sensor wrapped into product, which extends its functions. Whit this, detector is not only able to measure physical property, but can process it, save data and provide them to user. Detectors are usually supplied as a "ready to go" product provided by company.

On departure of Transportation Telematics, detectors are often used equipment. There are many researches and thesis which used them for measuring traffic parameters. Mostly, they are used as single detector for one purpose, other times there are used multiple detectors of same type in other places at once. In all cases we have got experiences and know strong and weak parts of using these detectors.

There are researches and theses dealing directly with this problematic in order to improve these detectors or help understanding their advantages and disadvantages. The biggest survey in this field was included in Ing. Martin Langr Ph.D. "Fúze heterogenních dopravních dat pro odhadování směrových vztahů". This survey takes a place in city Slatiňany in Czech Republic.

There were few detectors tested in survey mentioned above. Four places each with different detector (magnetic, pneumatic, and two microwave detectors). For each place were used cameras for SPZ recognition, which were automatically recognized in postprocessing with "LPR Reader" application.

The main source for list of detectors will be gathered from the Traffic Detector Handbook which provides a comprehensive resource for selecting, designing, installing, and maintaining traffic sensors for signalized intersections and freeways [8]. In this book, detectors are dividing into categories based on properties such as:

- Sound (acoustic sensors)
- Opacity (optical and infrared sensors and video image processors).
- Geomagnetism (magnetic sensors, magnetometers).
- Reflection of transmitted energy (infrared laser radar, ultrasonic sensors, etc.).
- Electromagnetic induction (inductive-loop detectors).
- Vibration (triboelectric, seismic, and inertia-switch sensors).

There will be need to divide detectors into custom categories for this work. It will be done later in this chapter, for now, division by Traffic Detector Handbook will help with listing detectors used in traffic industry.

3.1 Detectors We Have Experiences With

The detectors have huge impact on traffic control. They are first node in control telematic system. Requirements in this field is to capture presence of vehicle on exact spot on pavement, speed of vehicles or catch vehicles SPZ (license plate). For this, widely used detectors are induction loop, video detection, microwave detector or sound detector. From listed detectors we have most experiences with video detection and microwave detector. Sound detector and induction loop still needs to be further research.

Other used detectors, mostly for "Fúze heterogenních dopravních dat pro odhadování směrových vztahů" [2] work, were pneumatic detector, magnetic detector and video recognition of SPZ. It will be decided later if these detectors will be implemented into developing system.

3.2 Grouping Detectors

The grouping of detectors is important because to connect them, I need to know what they have in common, so whole problem becomes easier. Some grouping, although obvious, won't be used here because for purposes of integrating detectors is not important. For example, divide detectors into groups due to physical (elementary) principles on which their work. In other words, I don't need to know if detector is using IR barrier working on light principle or induction loop which works thanks to magnetic field, important is that both are able capturing vehicle occurrence but have different interface and same data format.

3.2.1 Grouping due to Measured Parameter

First grouping divides detectors into groups due to parameters they are able to capture. By parameters is meant raw physical properties or other raw information about vehicle. Some detectors are counting other parameters from the basic ones straight in measuring process. For example, as shown in [3] it is possible to calculate speed and length of vehicles from two virtual loops of video detector Pico Iteris, but raw parameters, detector is able to capture, are occupancy of virtual loop by vehicle. Therefore, none traffic parameters (intensity, density, etc.).

- **time of vehicle passing point of interest [hh:mm:ss]** is important for multiple reasons. It can be used as referent point for section speed, for a simple vehicle counter or to assign time to ID of vehicle.
- **duration of vehicle spent on detector [s or ms]**. With this parameter can be calculated speed or length of vehicle. This method was tested in [3] on vide detector, but can be applied on microwave detector or others.
- **occupancy [0/1, true/false]** detection of vehicle in detector area. Widely used by light controllers to detect vehicle in lane.
- **speed [km/h]** for section or instant speed of vehicle.
- **vehicle identification [ID (SPZ, Bluetooth address etc.)]** when there is need to assign a ID to a vehicle for section speed or directional survey
- **others**. In this category belongs all other detectors as weather station, pavement temperature and density detectors, noise detection etc.

3.2.2 Grouping due to Interface

There will be needs to solve interfaces later in this work. When dividing detectors into groups by different physical interfaces, it will be easier to deal with these interfaces later. By solving how to connect one interface into system, all detectors from this particular group will be compatible as well. The interface here is physical interface, whit which is detector or sensor connected into system on hardware level. Some of these can even provide electric supply (this will be important especially for sensors), others just communication.

- Simple or multiple electric cable
- Ethernet cable
- USB
- Bluetooth
- Wi-Fi

3.2.3 Grouping Due to Place of Capturing Vehicle

It is important to determinate where exactly detector detect a vehicle. This grouping will be important later when talking about algorithms and processing of data fusion. For saving correct data from detector, we must know what exactly these data means. There is difference between instant and section speed, or when the detector can detect vehicle on certain area (Bluetooth identification) or on one exact spot which is same through the whole time of measurement (induction loop). With this on mind, it was decided to divide detectors into following groups:

- Spot (cross-section)
- Circular area
- Longitudinal area

3.3 Conclusion of This Chapter

In these two chapters has been outlined how will this work handle sensors and detectors, what are differences between them, how they work in general, experiences while working with them and were divided into groups for easier work. By solving problem of group, all detectors belonging in it will be solved at once.

Next task will be to set requirements on sensors and detectors for them to be able to connect into integrating system.

For such purposes, there must be discussed physical interfaces and data integration and chosen set of sensors and detectors for testing.

4 Physical Integration

In other words, hardware integration. It is one of the biggest problems when talking about integration of sensors and detectors and must be solved before trying to integrate data structures or solving algorithms of evaluating data from multiple detectors connected at once.

To solve how to integrate sensors and detectors, it is required to dive into all components and study documentation of detectors constructed in factories, take apart and "hack" these where the required documentation is missing, or it is not obvious from documentation how this could be done and at last design custom components or devices allowing integration.

4.1 Experiences with Physical Integration

That this topic needs to be research comes from experiences with detectors. We have tried besides others magnetic detector, which was almost impossible to solve because this detector lays on pavement and has no inputs or outputs accessible during measurement. And on the other hand, there is video detector Pico Iteris, which is made for integration and its main purpose is to be integrated into other systems (light controller). This is allowed due to outputting pins, which can be wired and are sending simple information about the state of the detector.

4.2 Experience Examples of Electronic Supply

Sensors and Detectors are made of electronic components, so they need electronic supply to work properly. Electronic supply seems to be a small problem, but after testing some detectors, it was obvious, that this topic need to be solved. While researches on field, place of measurement isn't always easy to access and to carry huge and heavy battery boxes could be an issue.

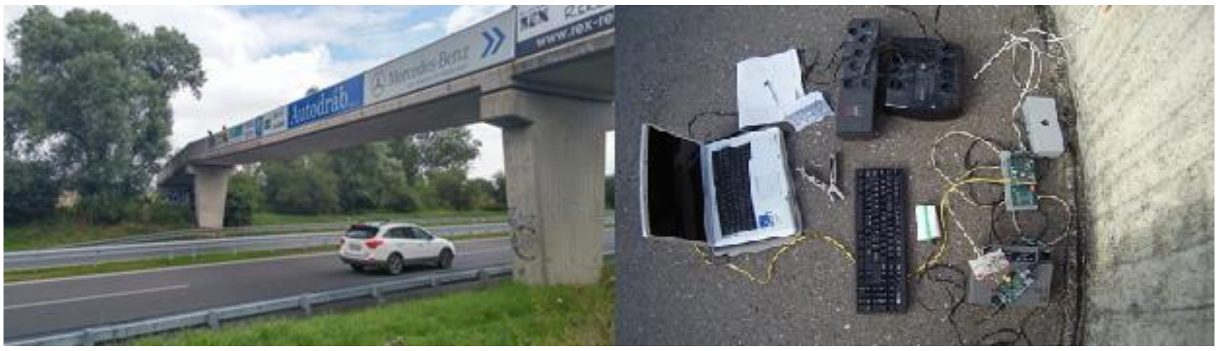


Figure 3.: Situation on testing place of Pico Iteris [3]

in other cases, battery for particular detector is too small and is not made for long term or even permanent researches. During the testing equipment on Slatinany [2], there were used cameras for capturing vehicles SPZ. Although the batteries were able to hold cameras for hh:mm:ss long survey, for survey few ours long, this could become a serious problem.

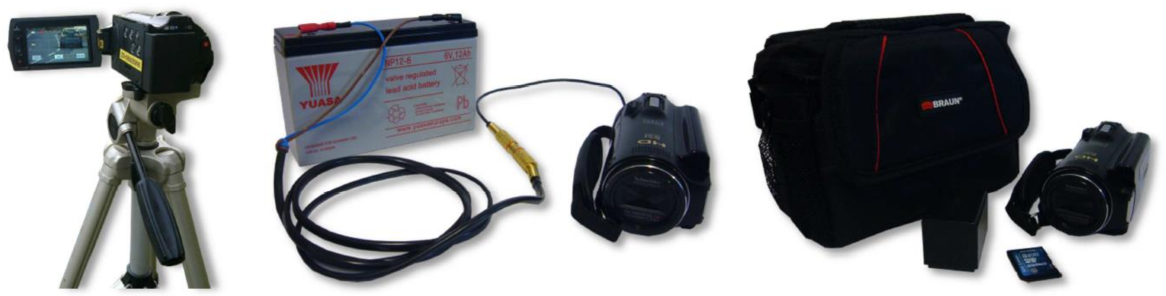


Figure 4.: Necessary accessories for cameras [2]

4.3 Requirements on Electronic Supply

Due to experiences stressed above, it was decided in follow list of requirements. Detector must fulfill at last one of those to be used by multidetector system.

1. Possibilities to connect to 24V power socket directly or through reduction
2. Large enough and light batteries to supply power for whole survey
3. Powering through cable (e.g. 5V, GND) mostly in cases of sensors
4. Other energy source able to hold for long enough (solar panel, etc.)

It needs to be mentioned, that these requirements are necessary, if user will come with other idea how to power devices, it's his choice. These points are just recommended, but there will be always need to somehow power sensors and detectors.

4.4 Experience Example of Connectivity

Each detector and sensor have its own way to provide measured data or information about its state. It would seem, that by this statement this problem has solved, but for integration, there must be set boundaries and rules for sensors and detectors to be able to connect into multidetector system.

First experienced issue was with physical connection itself. Interface between two devices (detector with computer or PLC) has each manufacturer solved separately, and there is no norm, so most of detectors are connecting in different way.

As to the sensors, there will be designed additional circuits which will be suited to conditions set in this chapter.

Best experiences were again with Pico Iteris video detector. It was design to be able to communicate with PLC of light controller on very low level with simple logic. In circuit of this detector there are switchers connecting lines between output pins and common pin (GND). With this, detector is able to send impulse whenever it detects a car on selected virtual loop. Form of impulses can be controlled by changing settings on PLC. These data formats will be discussed later in chapter "Data Integration"

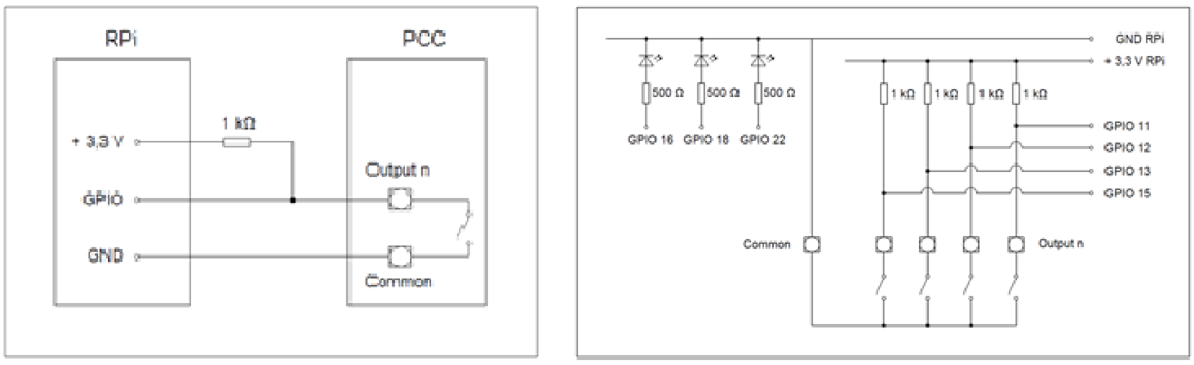


Figure 5.: Realization of connection between Pico Iteris and Raspberry Pi [3]

As a bad example for this category was selected magnetic detector Vaisala NC350. This device and work with it is further described in [2]. It is detector meant for surveys and to integrate it into any "online" system shown to be almost impossible. It has its own interface which is solved with cable ended with standard USB port, but it is meant to be connected after measurement is done. Data are processed inside detector and are provided in one file, online access to raw data would require to heck this device [9]. Thus, this detector is not suitable for multidetector.

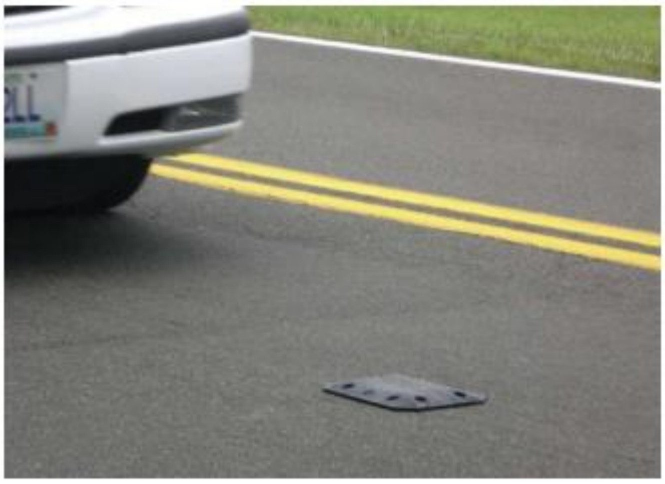


Figure 6.: Magnetic detector attachment on pavement [9]

4.5 Requirements on Connectivity

The first and one of most important requirements on detectors is to be able to be connected into other systems "online", during the measurement. Otherwise it would be same as to process data in postprocessing on computer with Excel, MATLAB or other SW able to manipulate tablelike data. To connect detector in terrain, there must be some medium transferring data from sensor or detector into multidetector.

Because there are too many products on the market, most of them with unique way to access data, it is almost impossible to include all of them. Thus, it needs to be set a few interfaces for which system will be designed and calibrated. Earlier in this work, detectors were divided into groups due to interfaces and this list will be now used. Each sensor or detector must have at last one of these interfaces:

- Simple or multiple electric cable
- Ethernet cable
- USB
- Wireless communication (Wi-Fi or Bluetooth)

The simplest possible connection are single or multiple wires connecting sensor or detector with other system like a computer or PLC). This interface is widely used for light controllers because it is simple and reliable. To connect wire, there must be output, in some cases also input pins or already soldered cables.

Next on list is standard IEEE 802.3 called Ethernet [10]. With this computer networking technology widely used in telecommunications, it is possible to transfer data fast and safe. For our purposes, it is required for detector to have standard ethernet card or other way to plug in ethernet cable and communicate, join network and communicate with other devices on network.

USB ports are often to see in traffic industry. This technology allows to easy download data from media, but there is need to access data online, so it won't be used in this work too often.

Wireless communication is extension of previous. It allows to connect devices without need of physical cable. Most common is standards IEEE 802.11 (Wi-Fi) [11] and IEEE 802.15.1 (Bluetooth) [12]. These standards specify properties and requirements on connection, so there is no need for detailed description.

4.6 Experiences and Requirements with Covers and Attachment Systems

Next requirements for successful product is to be resistant to external elements and harmful traffic and infrastructure environment. This can be done with solid cases able to resist rain, corrosion, wind, cold, sun, cars and others.

In some cases, detectors are meant to be placed onto pavement where vehicle can hit it or run over it. For these types of detectors there are common very strong cases able to hold weight of fully loaded truck as is for magnetic detector mentioned earlier.

For some sensors and detectors in this work, this part won't be so important, because they will be tested in laboratory conditions without harmful elements.

In terrain, detectors need to be fixed into one place. Often there is even necessary to place them into exactly given spot due to pavement and traffic flow. This can be achieved by many possible methods, the best ones showed to be direct attachment to already existing infrastructure.

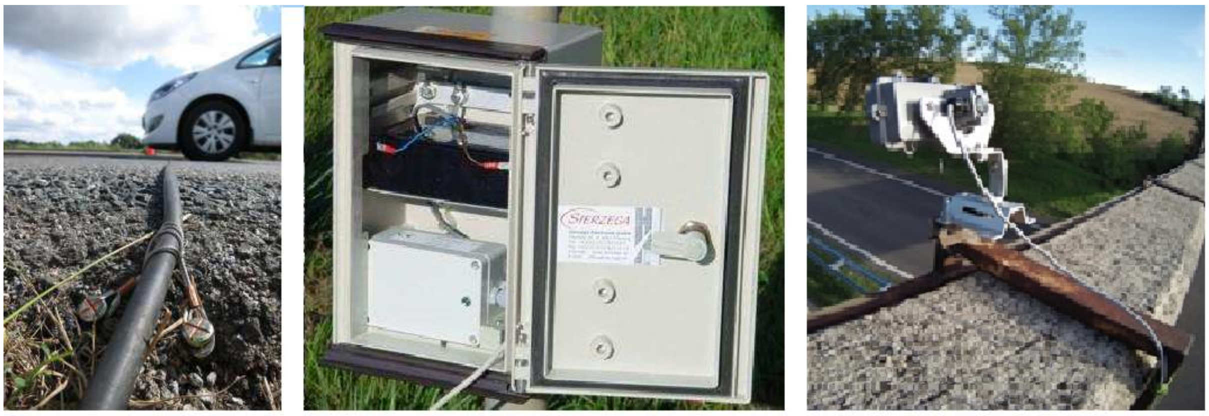


Figure 7.: Attachment of pneumatic, microwave [2] and video detectors [3]

Weaker solution is to use a temporary tripod. These can be unstable and fall due to wind from passing vehicles or stay in way of pedestrians.

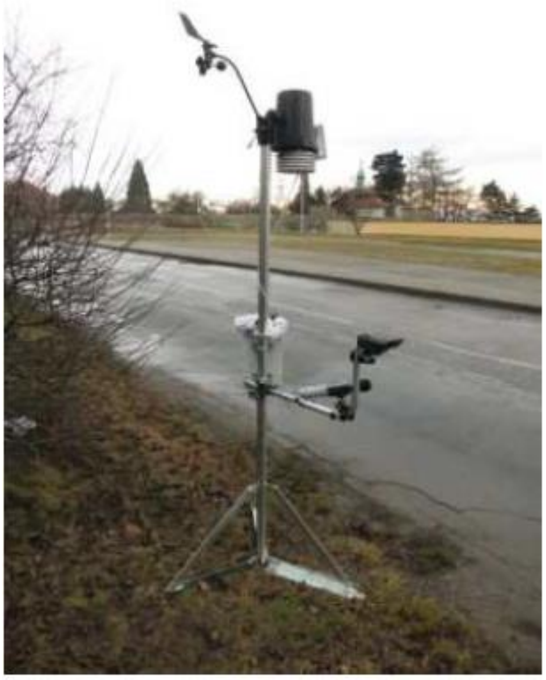


Figure 8.: Stative for weather station [14]

4.7 Sum of Requirements

There were discussed all kinds of requirements on physical properties of sensors and detectors to connect them into the system of multidetector. There are some necessary and some optional which simplifies work on field. Both categories are included in Table 2.

Table 2.: Requirements on sensors and detectors

Necessary physical properties	
El. supply	direct connection into 9V, 12V or 24V power socket
	adapter into power sources mentioned above
	batteries (with enough capacity and light enough to carry)
	Powering through cable (e.g. 5V, GND) mostly in cases of sensors
	other solution suited for particular detector
Connection Interface	simple or multiple electric cable
	Ethernet cable
	USB
	Wireless communication (Wi-Fi or Bluetooth)
Optional physical properties	
covers	protecting and weather prove
attachment systems	construction allowed detector to be fixed into existing infrastructure
	tripods or other temporary structure

5 Data Integration

For the first time in this work, we are moving into software domain and hardware is left behind. The statement “left behind” is not entirely true due to fact, that software layer of computers architecture lays directly on hardware layer and in some parts of this chapter it would be appropriate to mention how software and hardware are linked together and how these two layers affect each other.

5.1 Data and Information

First there is need to state meaning of „data“ in this context. „Computer data is information processed or stored by a computer. This information may be in the form of text documents, images, audio clips, software programs, or other types of data. Computer data may be processed by the computer's CPU and is stored in files and folders on the computer's hard disk.“ [15].

With the definition above, it can be set the difference between information and data. Information for our purposes, is a value of some physical quantity measured by sensor or detector. This information is processed and provide on output of sensor or detector. In some cases, this can be done directly on the device if this device is equipped with a display, simple diode or other output. Mostly, sensors and detectors process information and provide it as data on some communication channel (physical interface) to other device which can translate data back into information and save it or provide to user.

5.2 The Meaning of Data Integration

As was said, we are talking about data as processed information from physical values of sensors, so first data layer is how data looks after this process. There is question whether these data can be directly transported or if they need further processing.

The problem comes with trying to integrate data from multiple sources. Question comes with age of data, in other words, is data which comes out from the detector and were transmitted through medium into collecting system current, or was there any delay?

Some detectors are able to provide data with timestamp which presents real time of record of an information. This case is the best solution and most of the detectors has this ability, this does not apply on sensors, which lack process unit and therefore information has to be processed by integrating system, or there has to be design device for each particular sensor that doesn’t fit these requirements.

5.3 Problems Needs to be Solved

There will be special requirements on data of different physical interfaces during transmission. There are therefore 5 parts to be solved, simple or multiple electric cable, Ethernet cable, USB, Bluetooth, Wi-Fi. It is assumed, that in this state, physical interaction is working so both sides are communicating properly.

5.3.1 Direct Connection with Cable

Simple cable connecting detector with system is the simplest way of passing information. And it is raw information that is send through interface

This method allows to send any form of signal, short, long or periodically repeating. Due to this fact, information doesn’t need to be always transformed into data and after transmission back to information.

This form of communication will be used mostly by sensors, which will use the system as extension to become detectors, and information will be customized to suit our purposes.

But some detectors are using this method as well. These are mostly detectors for light controllers. Again, as an example, video detector Pico Iteris, which can detect vehicle on part of camera field of vision and send this information through cable to PLC of light controller. This information about vehicles is presented here in form of signal which can have multiple shapes. Each shape will represent the same vehicle, but each in different way [3].

- presence
- extension
- delay
- pulse
- addition
- none
- low contrast

Cables will be used not only for one-way communication. Another useful feature is ability to send control signal for sensors.

These signals often behave as prior supply for sensors, as for example ultrasonic sensor, where the time between sent and received signals indicate how long sound wave travelled to an object and back.

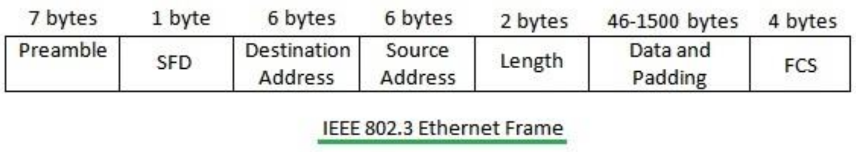
Due to these facts, it is important to harmonize detector with integrating system, so each of them knows what the send information should represents. For sensors and detectors with this interface it is required ability to send signals of known shape and known voltage level. Shape of signal should stay same after initialization of system through whole survey. But for voltage levels, it is important to stay same through whole system life, because changes of voltage could cause harm to device integrating whole system of detectors.

5.3.2 Ethernet Cable

Ethernet cable consists of multiple wires transmitting data from one device to another, so it could seem, this type of interface belongs to previous one, but this type of connection is unique because of norms saying how the interface should look and behave worldwide [10].

There is norm for hardware (cables and sockets) and for software as well. Hardware part was solved in previous chapter and it would be considered, that each device that meats hardware requirements (cable fits to socket) meets the requirements on software as well.

The payload of Ethernet is called frame and whole message is split into these frames, which can carry maximum size of 1500 bytes. The frame on picture Figure 9.



7 bytes	1 byte	6 bytes	6 bytes	2 bytes	46-1500 bytes	4 bytes
Preamble	SFD	Destination Address	Source Address	Length	Data and Padding	FCS

IEEE 802.3 Ethernet Frame

Figure 9.: Ethernet frame

One example of such an Ethernet connection is an integrated sensorics station Vantage Pro2 from the company Davis able to measure weather parameters. This detector is aa good example of sensors with additional interface providing user with measured information on display with possibility of sending data through Ethernet connection.

In most of the cases, this interface and its logic of frames is “ready to go” in product. This will be important to solve when designing new detector from sensors.

5.3.3 Wireless Connection

When talking about wireless connection, there are two types which are considered, Wi-Fi and Bluetooth. Both cases are similar to the Ethernet. Both have its standards of establishing connection and transform data.

The data of Wi-Fi are organized in packets on an Ethernet link, as well as in Bluetooth where payload is split between packets.

Range of this work doesn’t have space for detailed analysis of how does OSI layers work and other details in telecommunication. For integrating system is important that detectors allowed to connect on this media meet norms and that data will be delivered as bits by packets.

5.3.4 USB Communication

“A Universal Serial Bus (USB) is a common interface that enables communication between devices and a host controller such as a personal computer (PC). It connects peripheral devices such as digital cameras, mice etc.” [13].

“The USB design is standardized by the USB Implementers Forum (USBIF) that is comprised of a group of companies supporting and promoting the USB. The USBIF not only markets the USB but maintains the specifications and upholds the compliance program. Specifications for the USB were created in 2005 with the 2.0 version. The standards were introduced by the USBIF in 2001; these included the older versions of 0.9, 1.0 and 1.1, which are backward compatible.“ [13]. Thus, we can rely on a norm, setup system on its requirements and propose, that each device fulfilling this norm will be able to communicate with system on this interface.

5.3.5 Problems with Special Software

Problem with data integration in many cases fails on software of detector. Software is in most products delivered preinstalled with limited space for customization.

Mostly, detectors are programmed to communicate only with software specially designed for this particular detector, and such a software can be installed only on limited types of operating systems (Windows, iOS). This could be problem for systems with less known operating systems (Linux) or without operating system at all (PLCs).

Another encountered problem when working with software for detectors was, that software process data from sensors online, but provides them in intervals. This is no problem when integrating some detectors (weather, GPS), but can cause big issue with parameters which needs to be as precise as possible (speed, time of vehicle passing)

Therefore, requirements on software are:

1. not to process data on another program installed on computer
2. be able to provide online data for parameters where precision is required

Later in this work, there will be need to marge multiple data, design algorithms to manipulate them and finally save and provide them. Therefore, it is required to know what exactly each piece of information means and what parameter it represents in which time. This condition satisfied most of the detectors. Sensors are fully dependent on designed algorithm, which will be processing information.

Exact data formats, algorithms and saving into files will be designed later in chapter Algorithms and processing of data fusion.

5.4 Problems with "Online" Communication Channels

It is obvious, that when talking about some parameters, time has huge impact and even small deviation can cause big difference in measured data.

When talking about detectors, this problem can be eliminated due to the fact, that most of detectors provides data with timestamp indicating when the occurrence was measured. In this case, it is important that system time of devices on both sides of communication channel was synchronized.

Sensors lack build in clock, so information about time must be processed in integrating device. This can be tricky due to different time interval of detection, place of detection and latency on communication channel.

5.5 Problems Encountered from Listed Above and Conclusion

There are some detectors tested, which aren’t suitable due to conditions stressed above. First problem encountered was with the cameras system recognizing SPZ and its software Ateas LPR Reader. All video footages needed to be processed after survey in software so there is problem with online data access. The same problem was with magnetic detector Vaisala. Although this detector process data online, there is no way to access them until the survey is done and data are downloaded as a file.

To sum all requirements on software and data integration, if communication is solved with simple or multiple cable, shape of sending signals need to be known ahead and voltage levels must stay same after design of circuit is done. USB interface must be supported by drivers and Ethernet, Wi-Fi and Bluetooth must meet norms. Software of detectors must be accessible during measurement and must provide online data.

6 Chosen Sensors and Detectors

In this stage of work, all requirements on sensors and detectors are stated. It is clear now, what physical interfaces they need to successfully connect to system and how their software need to work to ensure proper communication with other parts. There was as well mentioned problem with online access data and with special applications many detectors have, and which can cause problem with integration.

Now it is time to define a starting set of sensors and detectors, so whole system could be tested after its design and realization is done. This set should represent all points of integration, that could be encountered. These problems are listed in previous chapters, so there will be at last one exemplary member with single cable connection, Ethernet, USB, etc., with its own software, with battery, with need of external power supply etc.

Moreover, there will be trend to choose sensors and detectors already available in Department of Transport Telematics. This is due to fact, that the aim of this work is to design and test integrating system, not to learn how to work with all detectors, although some settings and hardware details of some of them will be necessary to discussed. Another advantage is, there will be no need to buy new, mostly very expensive, devices.

6.1 Video Detector Pico Iteris

The most mentioned detector so fare was video detector Pico Iteris. This detector is detailed discussed in work [3], so here will be just touched main principles and explained, why it should be tested on this system.

The detector consists of two main parts. Video camera and PCC module. The main purpose of video camera is to capture footages and send it through twisted pair cable to PCC module. The PCC module is the main brain of this detectors. It processes recorded footages online, detect vehicles and store or send information about these vehicles.

Detector doesn’t search on whole field of view. To detect vehicle, it has to drive into one of the given area of interest. These areas are called virtual loop and one PCC module can work with up to 16 independent loops.

When PCC detects vehicle on virtual loop, it triggers the switcher (transistor) which was assigned by user while setting all system up. This close circuit on connection physically screwed and leading to another electronic device. Usually to PLC of light controller.

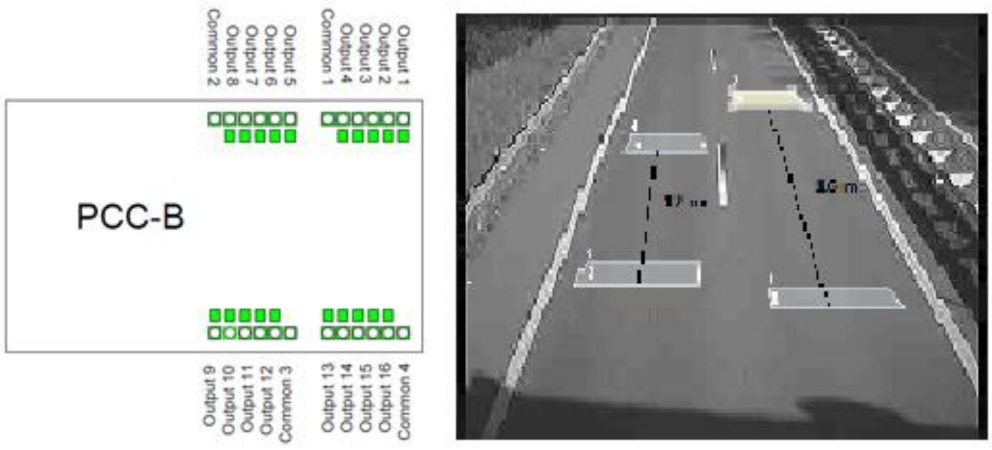


Figure 10.: Pico Iteris system. Pins of PCC [3] (Figure 8) and virtual loops [3] (Figure 21)

This detector will be added to system because it is great opportunity to test simple wire connection which is often used by light controllers.

All settings and options of this detector were already tested, and the device was connected to computer, but there will be need to test algorithms and logic of data integration as well.

Electronic supply is solved by 24VDC connector with need of minimum of 6 W but recommended is 10 W. In field, this is done with batteries, which are very robust and heavy, so this solution is not appropriate to use on hardly accessible places.

Moreover, video camera and PCC module are already equipped with strong and weatherproof cases. The mount system is solved here as well.

6.2 Bluetooth Device

This method was chosen to test working with identification of vehicles. This could be as well SPZ recognition from video footage, but as said before, available system was tested and didn’t meet requirements on software. The video footage was processed in postprocessing and there was no way to get it working online on field.

There will be as well good opportunity to test algorithm, because not each car has handsfree or other device communicating on Bluetooth frequency. That this method isn’t satisfactory to reliably identify vehicles at its own but will be used for designing and testing algorithms.

It shown, that only 22,5% (approximately) of vehicles can be identified by this method. All results are available on [16].

The principle of assigning an ID to vehicle is in sniffing Bluetooth communication. All devices when turned on are searching other devices or are communicating with already paired. Each device has its own address. This address contains of manufacturer-assigned code and organizationally unique identifier, see at Figure 11.: Bluetooth address. The second part of Bluetooth address is not so important for identification and is harder to sniff from signal because it is encrypted due to security reasons. The first part called LAP is unique for each device and devices are not hiding it when communicating with each other. Due to this fact, it is simple to catch this part of address and use it for identification.

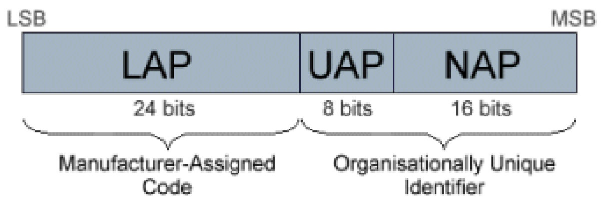

LSB
MSB
LAP
UAP
NAP
24 bits
8 bits
16 bits
Manufacturer-Assigned Code
Organisationally Unique Identifier

Figure 11.: Bluetooth address [17]

The realization is done with Ubertooth One, An open source 2.4 GHz wireless development platform suitable for Bluetooth experimentation [18]. This device has its own programmable chip so is able to process information from RF connector (antenna) into data and therefore falls into the category of detectors due to definitions mentioned earlier.

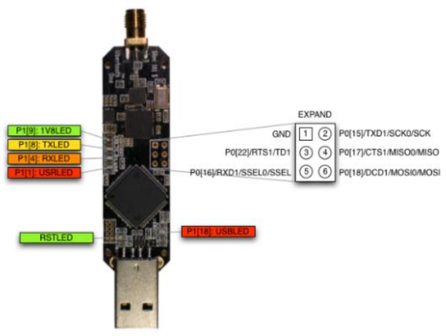


Figure 12.: Ubertooth One parts [19]

Ubertooth One has also USB interface so it will be appropriate to test and comment, eventually adjust requirements on this interface. The USB drivers here are part of the package installed on Linux. This package is able to setup Ubertooth one, turn measurement on and finally read data on interface.

As to a power supply, it is provided through USB port. Device must be connected on USB port and is not able to run on its own. Therefore, there is no need of solving problem with battery or adapter into 220V, whole electronic consumption will be drain from device it is plugged in. Power draw can be seen in Table 3.

Table 3.: Power usage of Ubertooth One in different modes [19]

Description	**Power draw (amps)**
Idle	0.09 A
Receive	0.13 A
Transmit	0.22 A
Firmware upgrade	0.10 A

This would be also a member of group with its own software, able to communicate with systems software. It could be said, that this device is behaving like sensor, because program controllable on chip will be fully under control of another device. There will be need to use open source libraries, but benefits from this are possibility of fully customization of software and easy access to online data.

6.3 Sound Level Detector

There are many traffic applications using noise detection. One of them is to find out decibel levels close to communication for finding noise harmful areas. It is also possible to detect passing vehicle on different of dB level.

Available device on Departure of Transport Telematic is SL-451 from Voltcraft [14]. This device meets a requirements of European norm EN 61 672-1 and is classified as class 2 in accuracy (general measure). It is equipped with LCD display with possibility of display a bar chart to recognize even fast changes of decibel level (single vehicle passing by).

Figure 13.: Noise detector [14]

This device is able to detect a sound wave from 31.5 dB to 8 kHz and in range 30 to 130 dB.

Whole detector is powered by an adapter to a 230V socket or by 9V battery which will be used in a field. All setup can be mounter on stative or other device with fitting screw.

The device is able to store information in the memory, but this solution would not meet the requirements on online data access and therefore will be used the second way of accessing data through analog output. The analog output is able to provide voltage proportional to sound level, from 10 mV DC/dB to max 1Vms AC/130dB. Due to analogue output, there will may be need to construct component between detector and system. By adding an AD converter, the interface will be suitable for any digital computers.

6.4 Infrared Barrier

This will be the first experience with sensors. At first it was thought about simple switcher or button just to test simple interface, but the IR barrier can additionally have real world use.

First attempt to design a IR barrier was with one IR diode on one side and IR receiver. There were problems with reliability of the designed circuit, so it was decided to buy Infrared 2 - 30cm Obstacle Detection Sensor Module FC-51. This module has three pins. VCC for power supply, GND for closing circuit to ground level and out pin on which is indicated state of sensor.

On this device can be adjusted sensitivity of detection, that means on which distance the object is recognize. This won’t be really important in this work, because the testing of sensors will be done in laboratory conditions on really short distance.

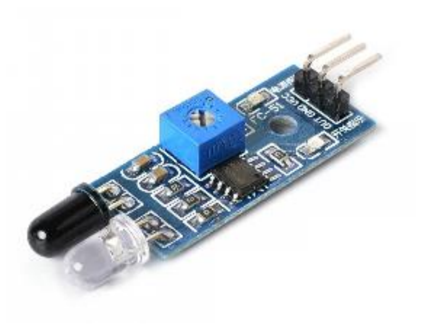


Figure 14.: Infrared sensor Module FC-51 [20]

6.5 Ultrasonic Sensor

This detector can be seen mostly inside garages, where it can detect vehicle standing right on the parking place. This detector works on very simple principle. It sends a sound wave and listen to echo reflected from obstacle.

It is good opportunity to test another sensor already integrated in circuit. The chosen sensor is module HC-SR04. This device can measure distance between 2-450cm and therefore can be tested in laboratory conditions.

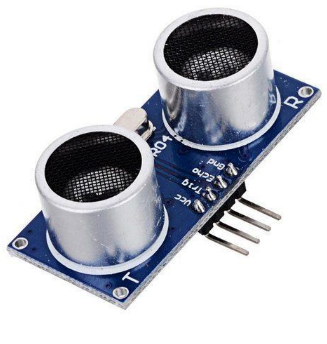


Figure 15.: Ultrasonic sensor [20]

The circuit has 4 pins. VCC and GND for providing an electric current of 5VDC, Trigger which triggers whole system and the speaker starts to transmit and Echo, on which circuit sends a current when there are detected a sound waves on microphone of same frequency as a speaker sends. The output voltage on Echo is 5VDC, which needs to be considered when connecting to interface.

6.6 Table of Interfaces Needed to be Tested

To sum all sensors and detectors and what interfaces and requirements they have, the table was created. This helps to see if all the interfaces will be tested.

Table 4.: Table of tested sensors and detectors

Interfaces	
simple wire	Pico Iteris, SL-451 noise detection, module FC-51, module HC-SR04
Ethernet	Weather station Vantage Pro2
USB	Ubertooth One
Wi-Fi	NONE
Bluetooth	NONE
Electric supply	
supplied external	Pico Iteris, Ubertooth One, module FC-51, module HC-SR04, SL-451 noise detection
supplied with battery	SL-451 noise detection
Software	
own software	Ubertooth One
without software	module FC-51, module HC-SR04,

From the table, we can see, that the Bluetooth and Wi-Fi interfaces have no member here. This is not issue due to strict norms defining how the interface and communication should behave. It is assumed, that each device fulfills these norms can be allowed into the system without any problems.

7 Definition of Multidetector

Now, when all requirements on sensors and detectors are set, it is time to specify integrating device. The main reason of this device will be to physically connect with input nodes into system (sensors and detectors), control them if required, provide power if needed and most importantly collect, marge and synchronize data and information.

7.1 Schema of Multidetector System

There were two main views on how to solve the problematic of system components and system schema.

7.1.1 Multidetector as Sensor Network

First Idea was to design sensor network defined as "A sensor network comprises a group of tiny, typically battery-powered devices and wireless infrastructure that monitor and record conditions in any number of environments" [22]. In this case, each node could communicate with each other and whole system would be decentralized.

This form of system would be good at situations when one node break. In this case network would be functional and other nodes would take over connection and communication.

Disadvantages is, that this system would be very complicated to design, setup and collect data. Another issue could be, that almost each sensors and detectors would need additional components allowing them to connect into the system and communicate on network. These components are expansive and for most detectors would need customization.

7.1.2 Centralized System Schema

Centralized system can be disabled when main component controlling each other is disabled. In this situation, whole system is shut down and central node needs to be repaired or changed. As for input nodes, their failure would not cause any harm to other system components.

Another strong point for centralized system is good adaptability of system when adding new input nodes. In this case only central device would need adjustments, not all nodes.

When only one device will be handling management of whole system, it would be easier and clear as well as data, which would be stored on one place.

Due to points stressed above, it was decided to use centralized system in arranged in star shape. Whole system will be consisted of one main component mastering others. The nodes acting in this system as slaves will be sensors and detectors.

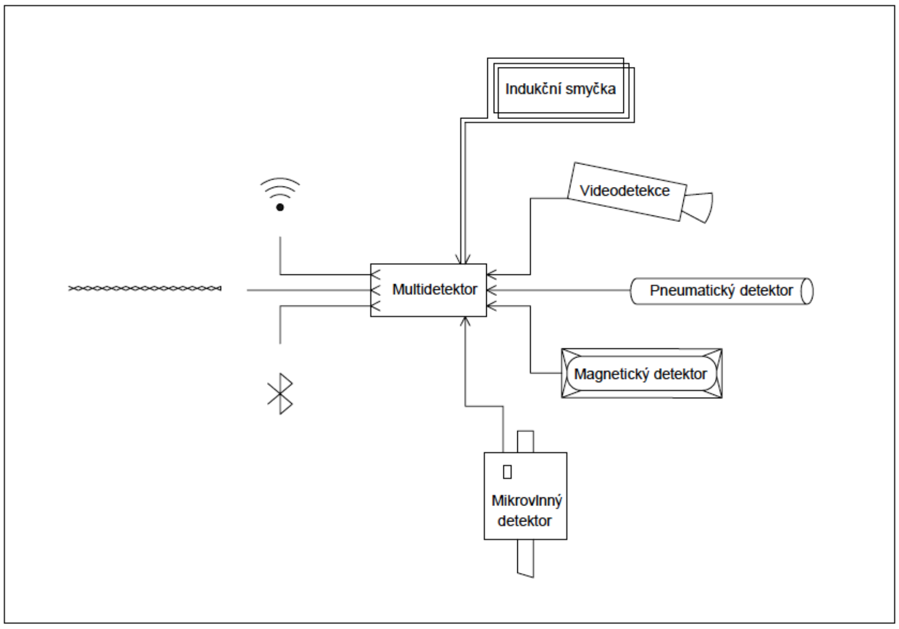


Figure 16.: Schema of system

We have already detailed discussed input nodes and their interfaces. As graphically shown on Figure 16.: Schema of system, the inputs into integrating device will be same interfaces as in output interfaces of sensors and detectors.

The last part on picture is outputs in form Wi-Fi, Bluetooth and twisted pair wire (Ethernet). These interfaces were as well already included into previous “interface” parts. The interfaces will be meeting same norms so there will be no need to discuss them further.

7.2 The Multidetector

Until now, it is not exactly obvious what the Multidetector is. There were two possibilities on how to define it First is that Multidetector is whole system including sensors, detectors, integrating device, connections, data storage or output interface.

Second possible definition is that the Multidetector is only the integrating device and all needed parts to function (processor, data storage, interfaces, etc.) but all input nodes are not included. For better terminology, the second variant was chosen. Therefore, the definition is “Multidetector is a device able to connect on physical and logical defined interfaces, format data, process them, store and provide to user on defined output interfaces”.

The definition is not quiet done yet due to fact, that interfaces will be defined later in this chapter.

If there will be needing to talk about whole system, it will be called Multidetector system defined as “The system including all sensors and detectors installed into system, connections with interfaces on both sides and Multidetector”.

7.3 Requirements on Multidetector

7.3.1 Physical Requirements

7.3.1.1 Requirements on Connection

One of the big topic of this work so far was how to connect sensors and detectors to the Multidetector. All physical interfaces on both sides of the connection will be considered as same and thus all interfaces of Multidetector were already discussed.

There are five main interfaces the Multidetector must have. The Multidetector must be able to read changes of voltage on input pins for single wire connection. The voltage range is not strictly given because there is no problem to change input voltage levels with voltage divider later.

Another two interfaces are Ethernet and USB sockets. For Ethernet, there is norm IEEE 802.3 [10] and for USB, there must be available drivers for operating system of Multidetector which will be chosen later in this work.

There are two members of wireless category, Wi-Fi and Bluetooth. Both has its own norm. IEEE 802.11 for Wi-Fi [11] and IEEE 802.15.1 for Bluetooth [12]. All detailed requirements of these two interfaces are detailed described in the norms so there is no for further definition.

To access the Multidetector, there are planned to be used two main interfaces. The Ethernet and the Wi-Fi. Both of these can be used from input interfaces for sensors and detectors. In this case, it must be considered, that communication channels might be closed for sensors and detectors in time of communication with Multidetector.

7.3.1.2 Requirements on Power

There are two branches of power requirements to be solved. First will define power supply for Multidetector itself and second power supply provided by Multidetector to sensors and detectors.

Power supply for Multidetector should be solved the way, that it could be plugged into 230V socket through adapter or powered with internal or external batteries. Power drain should be as low as possible for light battery pack in field. The bed example is the Pico Iteris detector which needs about 30W to run so battery needs to be huge and heavy to provide enough power through whole measurement process.

Second branch is mostly done by connection interfaces. The most important to discussed is power provided by simple cable connection. This connection is using input/output pins on Multidetector, so there must be available pins for power as well. It was experienced, that most used voltage levels in sensors technology are 3.3VDC and 5VDC so Multidetector must provide both of them. Additionally, there must be a grounding pin GND to close the circuit.

Some devices require power supply through USB which is usually 5V max 2A. Other option is to use PoE (Power over Ethernet) which is normalized on level between 44VDC and 57VDC [23] which is usually too high for sensitive devices this work will be working with, so this method probably won’t be used.

7.3.2 Software Requirements

The software layer of Multidetector will be fully custom. That means, hardware part needs to be programable with any of standard programming language (C, C++, Java, Python, etc.).

There are no specific requirements on operating system. It could be one of widely used (Windows, iOS, Linux), new one developed specially for Multidetector or proprietary operating system provided with the hardware by company (like systems running on PLC).

7.3.2.1 Controlling Data Flow

The chip of computer must have access to the physical I/O pins, so the program can read or change voltage levels on these pins. The pins usually have its own address in logic of computer and there are various libraries helping user to access them.

The program should have as well direct access to other interfaces. Usually, there are pieces of software handling control over these interfaces, so programmer can work on higher level with predefined functions. These could be OSI layers on Ethernet and Wi-Fi or drivers of USB.

When the data are collected by any of mentioned principles, they need to be processed. More on this topic in chapter about algorithms of data integration, merging and synchronization.

The obvious fact is, that whole software must enable access and work with variables of various types as well as to read and write files.

7.3.2.2 Controlling Sensors

Some input nodes of Multidetector system will need a special control to work properly. This fits mostly for sensors, which are bare circuits with special parts to measure change on physical quantities. These sensors need external brain to work and this will be provided by Multidetector. The process power needed for these tasks must be taken to account.

7.3.2.3 User Interface

From previous experiences with implementation of similar systems, it is highly required to have at last some user interface. This interface will be used to setup system, turn on measuring process, control state and if it runs properly, collect data and halt the whole system.

The way of how the user interface will be designed will be specifies later when the platform is chosen and operating system working. There are few possible methods: programmed special application, added display straight into Multidetector, communication through internet with SSH or web server, etc.

7.3.2.4 Data Formats to Save

The measured, collected and processed data should be saved. The best way to preserve data is to save them into files.

Most files we can see inside computers are compressed in some form. When Multidetector will be creating and saving data into files, these files won't be compressed during measurement. It is assumed, Multidetector will be saving data into files continuously during measurement and to compress and decompress file every time new data is added would take a lot of computing capacity.

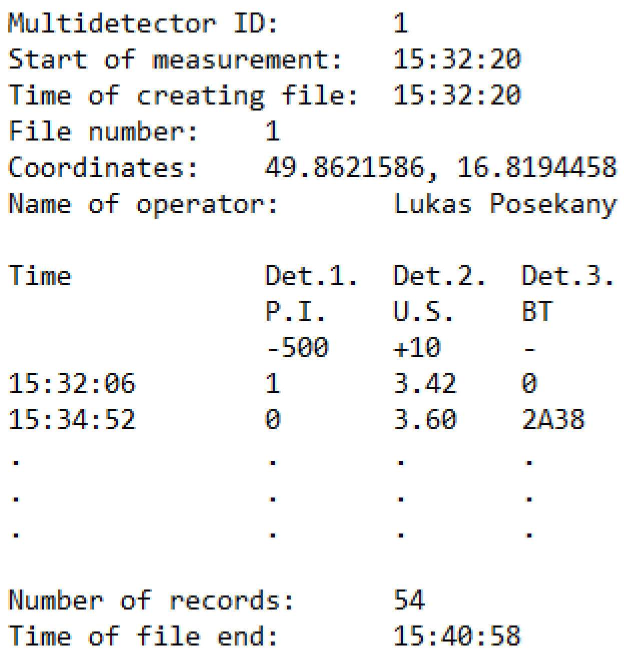
When the measurement is done, or after a given period when new file is created, the completed file can be compressed to save space on storage unit.

Each file will contain a header with important information about Multidetector like an ID of device (in situation when more than one Multidetector are used), time of starting measurement, time of creating a file, number of file in one measurement, position coordinates, name of user and additional information.

The records will be saved into matrix. The first column is time of event occurrence. Following files will indicate state of all detectors at this time. The first line will be header of table indicating which sensor or detector belongs to relevant column, second row will be keeping information about exact position of sensor or detector on a pavement relative to chosen fixed spot (recommended position of Multidetector). This value will be in centimeters.

The position on pavement can be virtual position on which detector spots vehicle. This can be handy when working with virtual loops of video detectors, where triggering spot is not in same position as detector itself.

At the end of the file, there will be added a footer containing information about number of records and time of end of this file.



```
Multidetector ID:        1
Start of measurement:    15:32:20
Time of creating file:   15:32:20
File number:    1
Coordinates:    49.8621586, 16.8194458
Name of operator:        Lukas Posekany

Time            Det.1.   Det.2.   Det.3.
                P.I.     U.S.     BT
                -500     +10      -
15:32:06        1        3.42     0
15:34:52        0        3.60     2A38
.               .        .        .
.               .        .        .
.               .        .        .

Number of records:       54
Time of file end:        15:40:58
```

Figure 17.: Example of data file

8 Platform of Multidetector

This will be one of the most important parts of this work. To choose a good piece of hardware is important for all electronic systems.

The strict requirements on hardware are given from previous chapter. Most important are interfaces. These requirements must be fulfilled to create a device capable to integrate detectors with physical properties listed in this work.

The main piece of Multidetector representing the brain of the whole system is CPU (central processing unit) with other additional parts keeping it alive (power supply, crystal oscillator, cooling system, memory, etc.). “The CPU is the heart and brain of computer. It receives data input, executes instructions, and process information. It communicates with input/output devices, which send and receive data to and from the CPU” [24].

Next logical step in choosing platform will thus be to check input/output possibilities. Some interfaces can be connected directly to chip, in which case interface must be handled carefully so manipulation won’t hurt chip. Others have electronic parts, circuits or other microchips between chip and interface. The second group are usually defined (often in norms), widely used interfaces like USB, Ethernet, HDMI, VGA, etc.

All these interfaces wont necessary be a part of hardware chosen for Multidetector, but it must be possible to add them later. This can be done permanently (by welding them to circuit) or temporary by connecting them with wires and headers or through some reduction to other, already connected interface.

With this knowledge the platform can be chosen. There will be listed few known possibilities and from them only one device will be selected for implementation and testing.

- designing own HW from individual parts
- Arduino
- PLC Tecomat Foxtrot
- Raspberry Pi

8.1 Designing from Scratch

This option is the hardest one to achieve. It requires to completely design and build custom computer on low level. When the building is done, the computer should be tested redesigned and rebuild all over again until the systems performance and functionality won't reach desired level.

To build such a complicated electronic system from scratch would take a lot of time, effort and knowledge such a system would probably never reach a quality of computers build by big companies with lot of experiences and suitable equipment.

Only advantages could be fully custom device made for only one purposes, to become a Multidetector. It would mean, that even software would have to be fully custom and written from scratch. This would also take a lot of time and effort and result wouldn't be necessary the best functional system. There are two possible ways to go with software, custom OS or design HW in such a way, that it could handle already existing OS (Linux, Windows, iOS).

8.2 Arduino

Arduino is from following options closest to the custom designed computer. In its basic version it is simple programmable chip with input/output pins which program can control/read. This device can be programmed in C, C++ or other low-level language where compiler translate written code into binary code which Arduino understands.

There are few versions of Arduino available, between them two very similar, mostly used products: Arduino Nano and Arduino UNO. Arduino Nano is smaller "lighter" and cheaper. On the other hand, Arduino UNO is more robust and most used, therefore most documented member of Arduino family and will be considered as better suited for Multidetector [25].

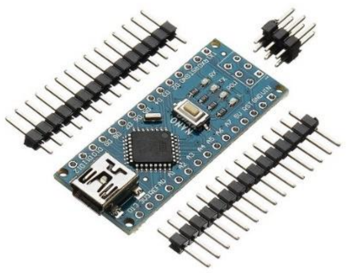


Figure 18.: Arduino Nano [25]

Figure 19.: Arduino UNO [25]

The Arduino UNO has an USB interface and I/O pins header which allows Arduino to communicate with other devices. This can be done with serial UART TTL, I2C or SPI communications. These are useful interfaces, but it is required from Multidetector to have more than that. Although this could be solved externally or with reduction as mentioned at start of this chapter. This solution has disadvantages in practicality or speed of communication.

As to specifications of Arduino UNO, it has a Microcontroller ATmega328P, operating on voltage 5V, is capable to handle on power input voltage up to 20V. As main interface it has 14 digital and 6 analog input pins. Clock speed is 16 MHz, which is for purposes of Multidetector fast enough. Dimensions are 68.6 x 53,4 mm and it weights 25g (Arduino Nano 18 x 45 mm and weight 7g).

Although Arduino is great for smaller projects, it could lack some functionality for more complex systems like the one of Multidetector. It also lacks a GUI of operating system, and for almost all kinds of required communication with user, except of USB (thus Wi-Fi, Bluetooth, Ethernet), must be added additional parts. There are also no internal clock keeping information about time when Arduino switch off or reboot, therefore time must be set every time Arduino is turned on and so communication with user would be necessary every time Multidetector is used.

8.3 Tecomat Foxtrot

There were PLCs mentioned in few places of this work, so it is a logical step to add one into list of possible technologies for Multidetector.

The Department of Transport Telematics on Faculty of Transportation Science have communicated with company Tecomat and although it is company specialize on intelligent building systems, these can be used even for a telematic systems because requirements, sensors, interfaces and processing requirements are almost the same.

To further understand PLCs and especially Tecomat products, meeten have taken the place with a Business Manager Ing. Jaromír Klaban to discussed possibilities of Tecomat device as platform for Multidetector. On the meeting, it was decided that the best combination for Multidetector project is to use Tecomat Foxtrot with additional modules for extension its functionality and number of input and outputs as well as other interfaces and serial buses.

“Tecomat Foxtrot is a compact modular control and regulation system with powerful processor, mature communications, original two-wires and wireless connection with intelligent electroinstallation elements and peripherals.” [26].

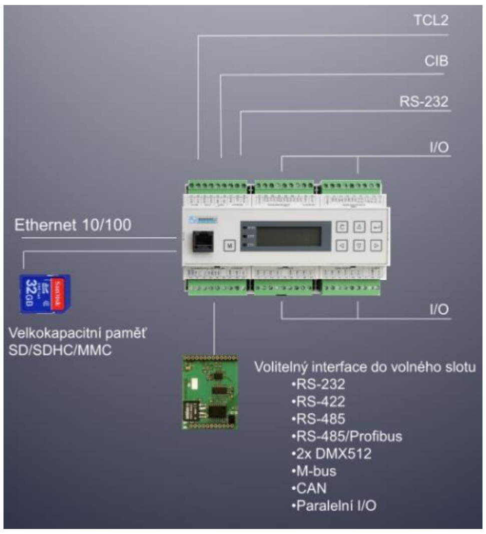


Figure 20.: Foxtrot CP-1000 [27]

Foxtrot is PLC which meet almost all requirements and thanks to the serial busses and additional modules made especially for this device it fulfill all interfaces.

“There is integrated FAST ETHERNET 100Mbit/s with RJ-45 connector and it enables direct integration of PLC Foxtrot into standard and industrial Ethernet network. On this fast communication line Foxtrot holds up to 6 connections, for instance with software Mosaic, SCADA visualization system, OPC server, graphical touch panel and concurrently answers to 6 web clients questions to built-in web pages” [26]. With this built-in interface, it is also possible to run a web server and serve built-in or custom web pages, which could be used as a GUI which is in requirements on Multidetector. There would probably be no need of connecting 6 users because only one person will operate Multidetector at a time.

“Foxtrot is perfectly designed to control intelligent electroinstallations and buildings. For this Foxtrot uses electroinstallation bus CIB – Common Installation Bus with guaranteed response between sensor and actuator to 150 ms. For this bus there is available still expanding amount of modules for interiors in the wall switch and sockets leading manufacturers designs, further modules for the installation into electroinstallation boxes, into lights and under covers of other device as well. Finally, there are modules for installation on DIN rail into switchboards like circuit breakers etc.” [26].

8.3.1 Programming Environment of Foxtrot:

“Tecomats are freely programmable controllers of the PLC category which are controlled by IEC / EN / ČSN 61131. Tecomat controllers are also harmonized standards in the European Union. The programming itself applies to IEC 61131-3. It defines two text languages IL (Instruction List) and ST (Structured Text), two Ladder Diagram and FBD, as well as a SFC (Sequential Flow Chart) program with an emphasis on the definition of program transitions between individual states.” [28]. The Tecomat Foxtrot is required to be programmed in Mosaic which is development environment for designing programs for Tecomat PLCs. “The integrated programming, engineering and service tool Mosaic is used to program Tecomat. It is executable under Windows. Allows PLC programming, simulation and stepping without a PLC connected. If the Mosaic computer is connected to the LAN with Internet access, it can be connected to any Tecomat in the world via TecoRoute communication service, and also can be diagnosed or PLC reprogrammed. Mosaic contains a number of other tools to help and accelerate the work of programmers, such as WEB Maker, Panel Maker, Graph Maker, and other powerful debug utilities.” [28].

8.3.2 Foxtrot Interfaces

There are few versions of Foxtrot from which, we will consider CP-1000 as best one considering the interfaces it has on its own without additional modules. CP-1000 has 6xDI, 2xDO, ETH100/10,2x serial bus, 2xCIB, 1xTLC2. This would be not enough for detectors to communicate with Multidetector so there has to be added modules to extend its functionality.

The modules can be added through common lines, serial, TLC or CIB. The last one is the bus designed by Tecomat and is able to support up to 270 inputs outputs / modules on CIB bus

Another useful interfaces are SCDA or other HMI connectable through TCL2. Such an interface would serve as GUI and control panel for Multidetector.

"The Foxtrot has also a GSM/GPRS/EDGE/UMTS/3G/HSDPA" [26] and although this interface was not considered before, it could get useful as remote controller of Multidetector.

8.3.3 Foxtrot Modules:

The main task for modules will be to serve as an interface extension. Tecomat has various types of modules, some of them has already built in sensors so Multidetector could be easily extended with these later. [29]. There were chosen two modules which would be necessary to have so Foxtrot could become the Multidetector (in its basic configuration). C-IB-1800M and C-HM-0308M.

C-IB-1800M is module with 4xAI/DI, 14xDI and 1xCIB and thus would be the main gate between simple wire connection and Multidetector. It could be used for switch, button, IR sensor, Pico Iteris or Volcraft sound level meter. There could be plugged in up to 18 sensors or detectors which is number of devices that will probably newer be reached in real situation. [29].

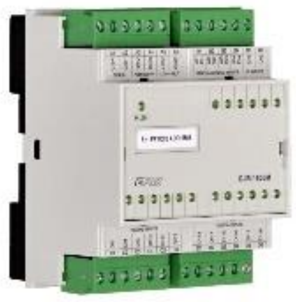


Figure 21.: C-IB-1800M [29]

C-HM-0308M is second module which provides, except for inputs, 6DO and 1AO. Outputs are required as well because many sensors require control or power supply. With this module, ultrasonic sensor HC-SR04 could be used (C-IB-1800M doesn't provide outputs so it wouldn't be possible) [29].

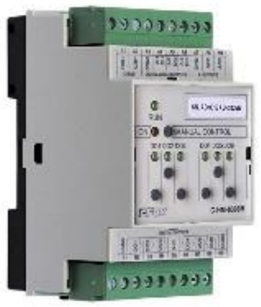


Figure 22.: C-HM-0308M [29]

The only interfaces which are still not solved are Wi-Fi, Bluetooth and USB. Wi-Fi and Bluetooth were promised by Tecomat to be designed for needs of Multidetector. USB is in current Multidetector system configuration used only for Ubertooth detector and because Ubertooth is made exclusively for computers with Linux OS, it would be overcomplicated to include this detector into system with Foxtrot (although Tecomat is now designing functions for Foxtrot which will serve as Linux fibers, so Ubertooth could be added into system later).

By this Tecomat Foxtrot possibilities to become a platform for Multidetector are discussed and will be compared with other technologies in the end of this chapter.

8.4 Raspberry Pi

The last chosen option for Multidetector is a Raspberry Pi. This device is from listed platforms closest to the computers as most people would define them. In fact, it can support operating systems used in desktop computers, laptops or servers.

Raspberry Pi was used as a device collecting data from Pico Iteris in [3] and in this work are described properties in chapter “Hardware řešení vyhodnocovací jednotky” so this text will be linked to it and main properties will be just briefly touched here. The reason to open this topic again is due to fact, that Multidetector has much wider requirements, and especially in interfaces and therefore it is necessary to research it more into depth.

“Raspberry Pi is a universal computer of small dimension. Although Raspberry Pi is less powerful than big desktop computers, it can achieve everything the desktop computers can do, it is especially good at controlling hardware. Big advantage is GPIO ports (General Purpose Input/Output), which allows to connect computer Raspberry with sensors, actors, lights or microcontrollers” [3].

Figure 23.: Raspberry Pi Model B+ [30]

The main properties of Raspberry Pi wouldn’t be described to detail. They will be just listed for better comparison with other technologies (platform build in from scratch, Arduino, Tecomat Foxtrot). As an Arduino or Foxtrot, even Raspberry Pi has several versions. They work all alike but have slightly different composition of interfaces. As a most suitable product for Multidetector it was considered Raspberry Pi 3 model B+ so the rest of this work will be talking about this product. The properties are:

- Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz
- 1GB LPDDR2 SDRAM
- 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE
- Gigabit Ethernet over USB 2.0 (maximum throughput 300 Mbps)
- Extended 40-pin GPIO header
- Full-size HDMI
- 4 USB 2.0 ports
- CSI camera port for connecting a Raspberry Pi camera
- DSI display port for connecting a Raspberry Pi touchscreen display
- 4-pole stereo output and composite video port
- Micro SD port for loading your operating system and storing data
- 5V/2.5A DC power input
- Power-over-Ethernet (PoE) support (requires separate PoE HAT) [31]

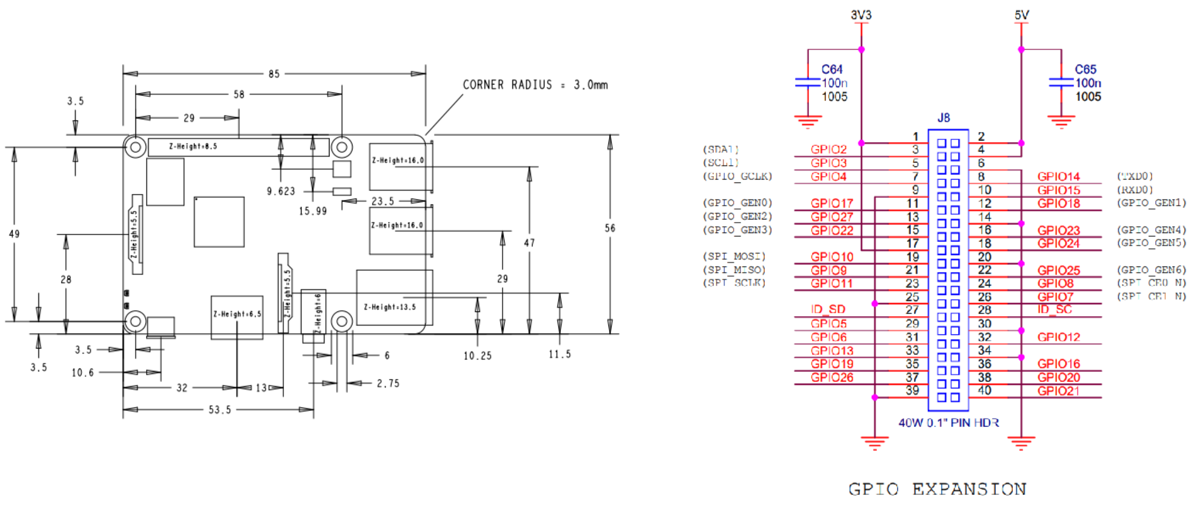


Figure 24.: RPi 3 mechanical drawing [30] and schematic of GPIO header [32]

In Properties list it can be seen, that Raspberry Pi is meet all requirements on interfaces. It has header with I/O pins, 4xUSB, Ethernet and Wi-Fi. Bluetooth can be added through USB sockets and there are various modules for Raspberry Pi for analogue I/O pins. analogue I/O pins will be important for work with Vantage sound level meter.

Like the Arduino, even Raspberry Pi has no inner offline clock, so it has no possible way to keep track of time while device is turned off. This already shown to be a problem in [3] to it will need to be solved if this technology will be chosen for Multidetector.

As was showed above, Raspberry Pi meets all requirements on platform of Multidetector and therefore can be used for this project.

8.5 Choosing Platform

Now, when all listed technologies are discussed it is time to decide which one will be best for implementation of Multidetector and therefore on which one will be while system tested.

The following Table 5.: Table of advantages and disadvantages of different technologies, was created to help with visualization of advantages and disadvantages. In this table we can see, that the less suited solution is to build fully custom computer from scratch. Arduino is not the bad solution but is very similar technology as Raspberry Pi and it has better parameters. So, the final choice will be between Tecomat Foxtrot and Raspberry Pi.

Table 5.: Table of advantages and disadvantages of different technologies

	price	dimensions and weight	interfaces	customization	robust	complexity
Custom	-	-	+++	+++	-	---
Foxtrot	-	+	+	++	+++	+
Arduino	++	++	-	+	+	+
RPi	+	++	+++	+	++	++

Although the Foxtrot would be better in many ways the Raspberry Pi has more required interfaces in its basic configuration without need of modules. It would be also better to supply power due to fact that it requires only 5V standard micro USB, so this could be solved with power bank.

Raspberry Pi is additionally smaller, lighter and support common operating systems (most importantly Linux). It has also better processing power, so it could perform even video processing if this task would be required later.

Although there are good relations between Faculty of Transportation Sciences and Tecomat company, and there was promised cooperation from side of Tecomat, the configuration and purchase conditions have not been made in time. Due to this face, it wasn’t possible to learn how to work with this technology and especially with programming environment of mosaic and therefore it was decided to use Raspberry Pi 3 model B+ as a platform for Multidetector.

9 Algorithms of Data Fusion

9.1 Logic Schema of Multidetector

Now, the hardware part of multidetector is chosen, it is time to dive into soft layer of problem. There is need to consider which processes will take a part in layer schema, the way the data will flow and specify algorithms which will control behavior of Multidetector.

The logic schema will be deduced from the physical one and from the image of how system should behave. The process starts in input nodes (sensor or detector) where information is created. This information needs to be changed into data meets the required data format. These data need to be collected, processed and saved. Whole logic schema is illustrated in Figure 25.: Logical schema of Multidetector

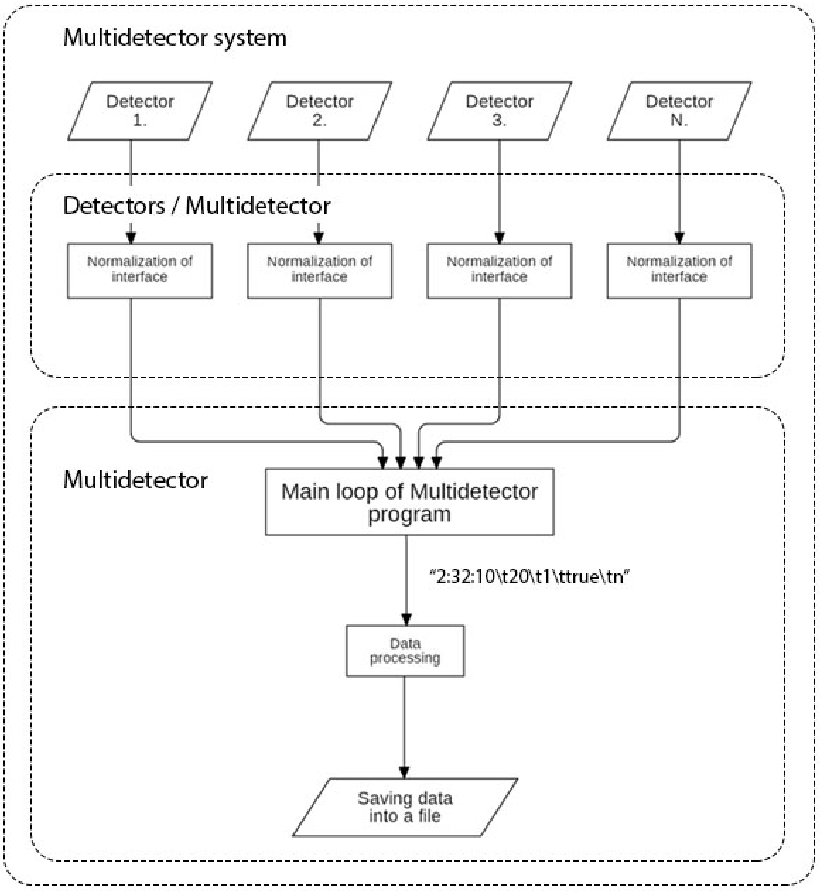

Multidetector system
Detector 1.
Detector 2.
Detector 3.
Detector N.
Detectors / Multidetector
Normalization of interface
Normalization of interface
Normalization of interface
Normalization of interface
Multidetector
Main loop of Multidetector program
"2:32:10\t20\t1\ttrue\tn"
Data processing
Saving data into a file

Figure 25.: Logical schema of Multidetector

In this schema is not seen, which part is happening in detectors and which in Multidetector. It is because it depends on detector, some are programmable, and all processing is happening in their program running on separate hardware.

In other cases, sensors or detectors needs a piece of external software to be able to provide required information in defined format. There will be programmed and implemented modules as a part of multidetector and if needs, new one can be added for new type of sensor.

In logic schema, we can see that there are multiple detector blocks. These blocks will be often running in multidetector device and thus will be sharing processing power with main program.

There are usually multiple possibilities to solve this problematic. Almost each programming language has its own way to handle multiple piece of code running at once. Even though not all computers have more than one core, problem can be solved concurrently, with processes, threads or other programming methods.

Problematic of processes and how will be structured will be further discussed later when multidetector platform is selected and is known if program will run on one or multiple cores. As well as will be known programming language.

9.2 The Main Algorithm

Now, that logical schema is known, it is time to start with designing algorithms of program controlling the main functionality and processes of Multidetector.

Diagrams of algorithms showed in this chapter will serve just for visual purposes. These will not be final program algorithms because it is not decided which programming language will be used. There will be used programming language conventions if required but that doesn’t mean, algorithms are ready to go into production.

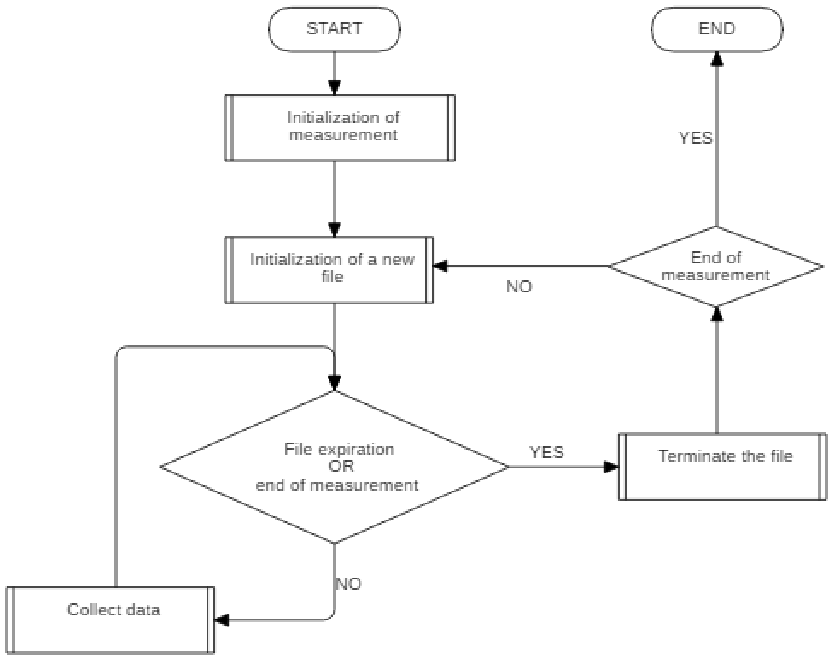

START
END
Initialization of measurement
YES
Initialization of a new file
NO
End of measurement
File expiration OR end of measurement
YES
Terminate the file
NO
Collect data

Figure 26.: Main program of Multidetector

At the start, its needed to get current time and read a configuration file which will contain all information about Multidetector settings and current measurement. User will be able to change this file directly or by user interface of Multidetector. After this, measurement will be initialized, controlling variables loaded and selected processes of detectors.

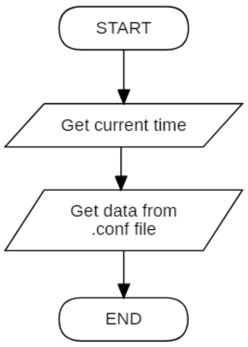


Figure 27.: Initialization of measurement

At the start of measurement, it is also required to create first file in which data will be stored. Each of these files need to have a header to hold information do file can be recognized later. This will be done begore main loop. In order to prevent data loss, the file will be closed and opened every time data needs to be written in. User will have choice to periodically terminate file and start the new one. In this case, new header will be written with the same time of start measurement and new time of creating file.

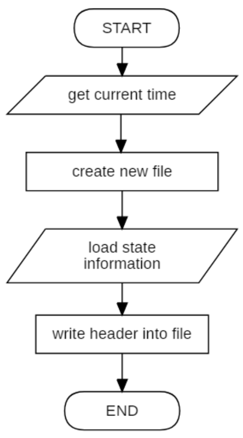


Figure 28.: Initialization of file

The main part of whole program will be infinite loop. This loop will make sure that data from all active detectors are collected, checked and eventually saved. It will be periodically checking state of Multidetector and if needed, it will terminate whole program.

At the start of each loop, there will be created two variables. The first, array type same length as number of active detectors, with empty values prepared for strings coming from detectors and the second one of bool type indicating if new values aren't exactly same as iteration before. The checking equality of detector values in two consecutive iterations will prevent file from repeating same rows, which would have no additional information (the only value of repeating same rows would be in checking if multidetector is still working, but as long as values won't be printed on terminal or streamed by some user interface, there is no need of this).

Once the value of the first detector process will be harvested, converted to string type, it will be checked against the value from the array stored from previous iteration. If the value is new, program will save it into same array position as old one and set checking variable to True (indicates that yes, I do want to save values in this iteration).

Then, the second detector process will be processed, third, fourth and so on up to Nth detector process.

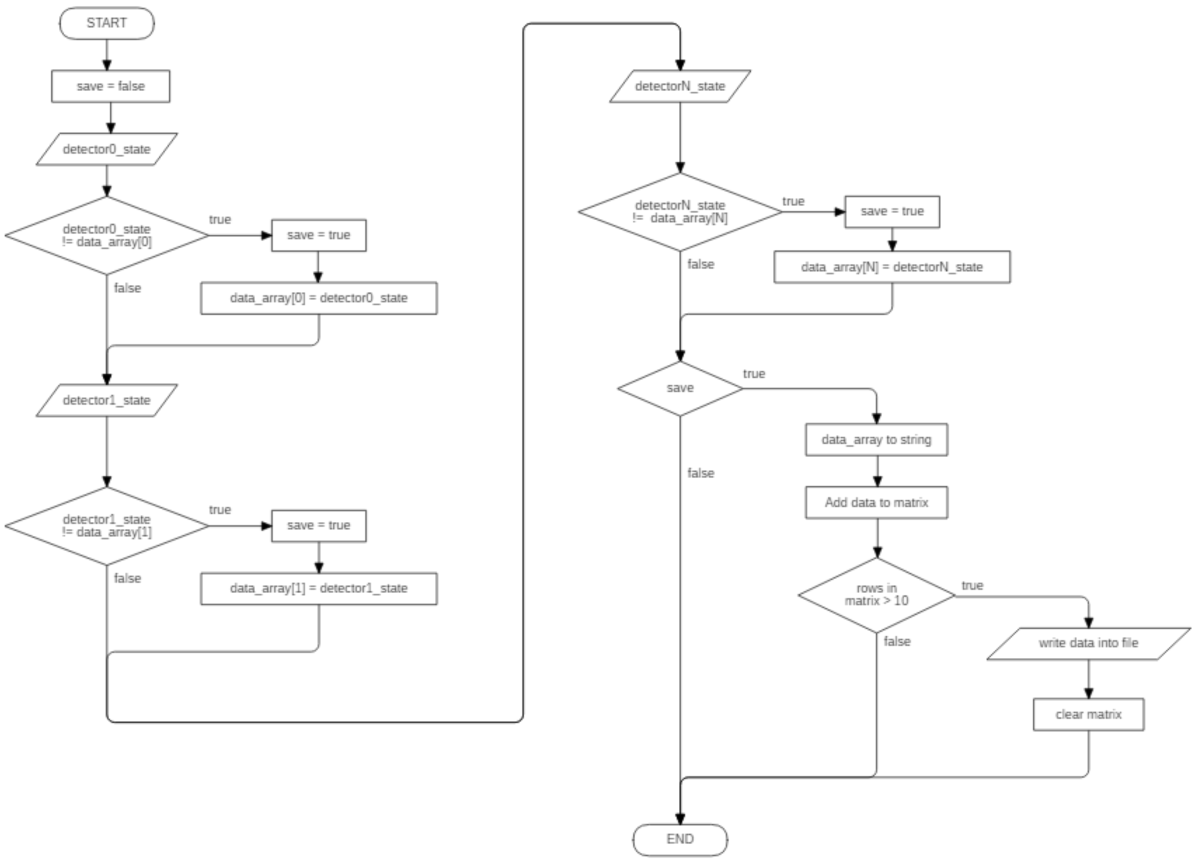


Figure 29.: Collecting data from detectors

At the end of the detectors loop, values from array will be stored into list of lists of values if the "save" variable is True (at last one of values have changed). Periodically (every time the tenth element is added into array), this matrix will be written into the file and the file will be closed again. This will prevent data loos in case the program would fall while stayed open. Next, there is need a check if there isn't need of terminating file or measurement program. If so, file will be closed and in first case, measurement will continue all over again.

9.3 The Logic in Data Selection

With this algorithm, it is guaranteed that every change will be written. Additionally, each detector process will have its own null value so there are only useful values written into file and to refinement data of additional information (0 won't necessary means null value).

First designed algorithm has been writing only not null values, which would lead to leaks of information about changes from real value and null and back again. The values would be same through whole file. Next version has been writing non-null values with additional first null value after row of non-nulls. This kept information about state change but has redundant values (row of same values with no change). The last algorithm recording only changes seems to be the best one as to performance and storage memory.

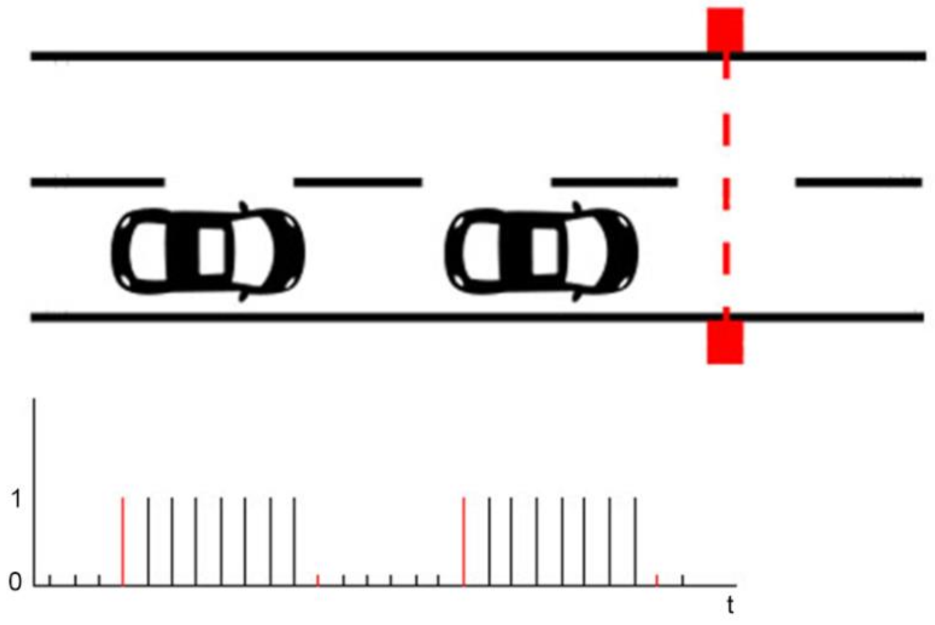


Figure 30.: Logic in data collecting

By this, the main view on algorithms is complete. There may be some small changes when real program code will be in process of making, but its structure will be following mentioned principles.

There are still more algorithms waiting to be solved, for example user interface, communication with Multidetector, algorithms of individual sensors and detectors, etc. but these are not so complex, so they will be solved during programming software part of Multidetector.

10 Software Environment of Raspberry Pi

It is finally time to design, build, program and test first version of Multidetector. All HW and SW parts and layers will be designed the way to fully fir Raspberry Pi and the selected operating system although it would be possible to use implemented methods and functions for different platform if the change would be required for later versions or similar projects.

As mentioned above, first task will be to choose proper OS, install it and run on RPi, configure environment of system and solve problems with communication and how to control Multidetector.

10.1 Operating System of Multidetector

Raspberry Pi can run mostly all operating systems (Linux, Windows, Android, iOS) but Linux is the best suited. Linux distributions are fully controllable, there are ways to fully customize environment, fully controllable from other machine through Linux terminal and is appropriate for running different kinds of custom services.

As to Linux distributions, there are few recommended for Raspberry Pi like Raspbian, Fedora, Gentoo, ROKOS, Arch Linux, Ubuntu. Raspbian was developed by Raspberry Pi developers [33] and after previous attempts, the Raspbian showed to run smoothly and with less errors and need of troubleshooting than other Linux distributions.

There are two versions of Raspbian, Raspbian Stretch with Desktop and Raspbian Stretch Lite. The desktop version is robust with lot of software installed in it. It has bigger image size, use more electricity and resources of raspberry Pi. On the other hand, Raspbian Lite is light, small, economical version. For this project was installed Lite version. Although there will be needs to install additional software later, it will be fully custom from start of project. There is also no graphical interface in Lite version, so all settings on Raspberry Pi will be controlled via terminal.

10.1.1 Installing, Write and Run Raspbian

Raspbian can be installed with NOOBS (New Out of Box Software) which is an operating system installer containing Raspbian and LibreELEC. After installing OS with this installer, there were some problems with running Raspbian in the past, so for this project was Raspbian downloaded and installed fully manually. Manual installation isn't that simple, but it allows fully customize all settings and version of Raspbian.

Raspbian Stretch Lite can be downloaded from official web page https://downloads.raspberrypi.org/raspbian_lite_latest. So, to download image into current directory is used command "wget" which will download content of URL. Next the folder needed to be unzipped and write into formatted SD card with:

```
sudo dd bs=4M if=2018-06-27-raspbian-stretch-lite.img of=/dev/sdb status=progress
  conv=fsync
```

After completing installation, SD card was mounted into the Raspberry Pi and USB power cable was attached, that will turn whole system on.

10.2 Communication and Control of Multidetector

There is the standard way how to safely communicate with Linux OS. This standard way is SSH protocol on port 22 and this will be used as a main communication channel with Multidetector.

There will be need to solve problem with connect Multidetector into same network and how to find the IP address. Later, the Multidetector will have its own network through Wi-Fi interface and on which other devices will be able to connect and communicate directly with it.

10.2.1 Command Line via SSH

SSH is cryptographical network protocol for operating network services securely over an unsecured network. It runs on TCP port 22 [34]. For this work it will be used to login into Raspberry Pi and controlling it entirely over network.

SSH can be allowed straight from Raspberry Pi, which requires monitor and keyboard connected. To connect to bare Raspberry Pi, blank file named ssh must be added into boot folder of SD card with Linux after installation. Path to SD card is /media/user. In other words, with Linux terminal:

Access to network can be done via Ethernet. When connected to LAN, DHCP server will give IP which can be hard to find from other computers. Best way showed to be connecting Raspberry Pi straight to computer with Linux distribution.

On connection settings was selected IPv4 Method "Shared to other computers", which will create network into which is Raspberry Pi able to connect. DHCP server will provide IPs in range 10.42.0.2-255. To find Raspberry Pi IP, the nmap method was used. This will return all active IPs in range 10.42.0.0 - 10.42.0.255.

```
nmap -T5 -sP 10.42.0.0-255
```

IP 10.42.0.1 belong to current computer, second one to Raspberry Pi. To used Raspberry Pi was given IP 10.42.0.206

to SSH to Raspberry Pi use Linux terminal

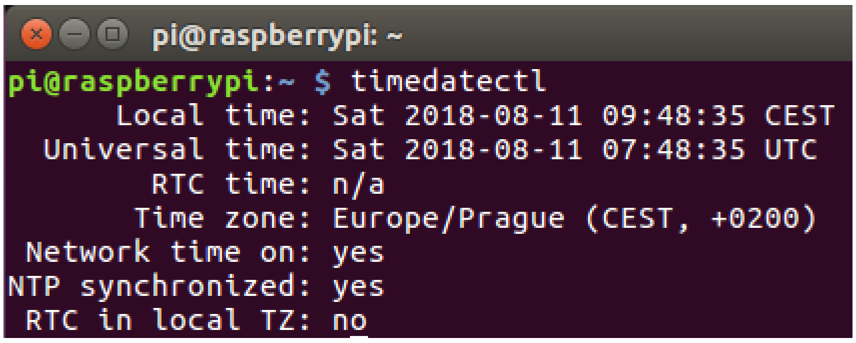
```
sudo ssh pi@10.42.0.206
```

or Putty on PC. Default password is raspberry.

10.2.2 Time of Multidetector

The first thing to be done with SSH connection will be set current time. Raspberry Pi has no real time clocks; thus, it is not able to keep current time while turned off. When Raspberry Pi turns on, it will try to get current time from NTP server according to time zone. To start controlling time on Raspberry Pi, there was need to find current settings of time date controller:

```
timedatectl     Figure 31
```



```
pi@raspberrypi: ~
pi@raspberrypi:~ $ timedatectl
      Local time: Sat 2018-08-11 09:48:35 CEST
  Universal time: Sat 2018-08-11 07:48:35 UTC
        RTC time: n/a
       Time zone: Europe/Prague (CEST, +0200)
 Network time on: yes
NTP synchronized: yes
 RTC in local TZ: no
```

Figure 31.: Timedatectl command output

Raspbian is after installation set on American time zone. To set it on real time zone, there was need to find and set the name of the correct one.

```
timedatectl list-timezones
  => Europe/Prague
sudo timedatectl set-timezone Europe/Prague
```

Now, Raspberry is in the correct time zone and it is able to get correct data from internet if connected.

Time can be as well set manually with set-time flag. It takes a string in format “years-months-days hours:minutes:seconds”:

```
sudo timedatectl set-time 'YYYY-MM-DD HH:MM:SS'
```

If the Raspberry Pi is connected to internet, it fetches date and time automatically from NTP server. Therefore, it was impossible to set date time manually. To solve this problem, automatic connection to NTP server can be stopped at the beginning of program (0 for disconnect, 1 for connect).

```
sudo timedatectl set-ntp 0
sudo timedatectl set-ntp 1
```

10.2.3 Wi-Fi Hotspot

To easy access Multidetector on the field without need of creating a local network on computer, there was decided to setup Wi-Fi hotspot build in Multidetector.

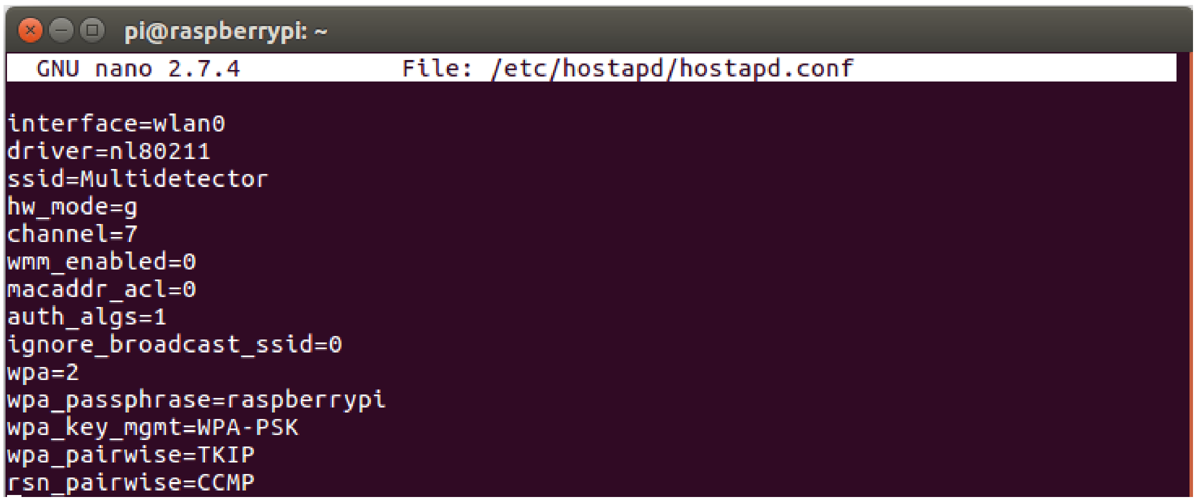
The access point was created with hostapd software. This software will create local network sitting on selected interface and allow other devices to connect to this network. The hostapd was installed with command:

```
sudo apt-get install hostapd
```

[35]

10.2.3.1 Net Configuration

For setting network parameters was created hostapd.conf file. Into this file were written information about interface, ssid (wifi name), password, security and so on. Final configuration file is showed on Figure 32.



```
pi@raspberrypi: ~
  GNU nano 2.7.4              File: /etc/hostapd/hostapd.conf

interface=wlan0
driver=nl80211
ssid=Multidetector
hw_mode=g
channel=7
wmm_enabled=0
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=raspberrypi
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```

Figure 32.: Hostapd.conf file

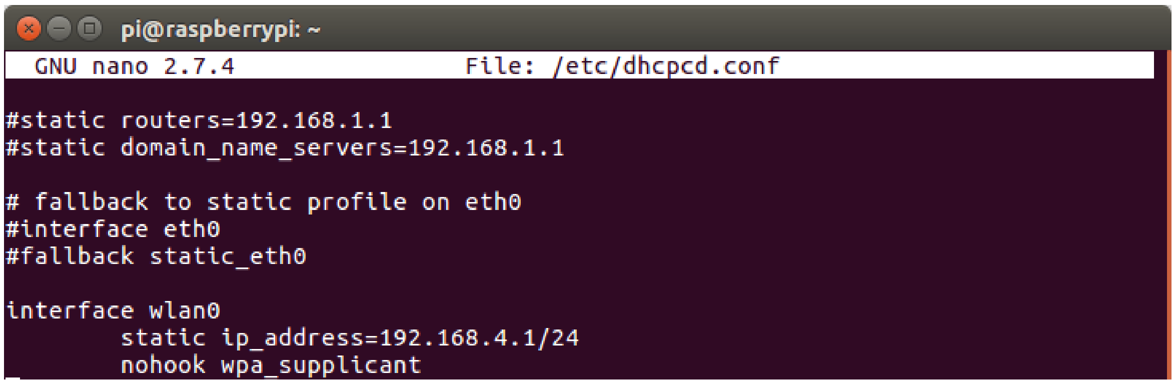
To tell the hostapd program where to find this configuration file, there was added line: DAEMON_CONF=”/etc/hostapd/hostapd.conf” into hostapd file located at /etc/default/hostapd

10.2.3.2 DHCP Server

After devices are connected into Raspberry Pi network, they need an IP addresses to communicate on that network. IP addresses are provided by DHCP server. For this case was selected dnsmasq program.

```
sudo apt-get install dnsmasq
```

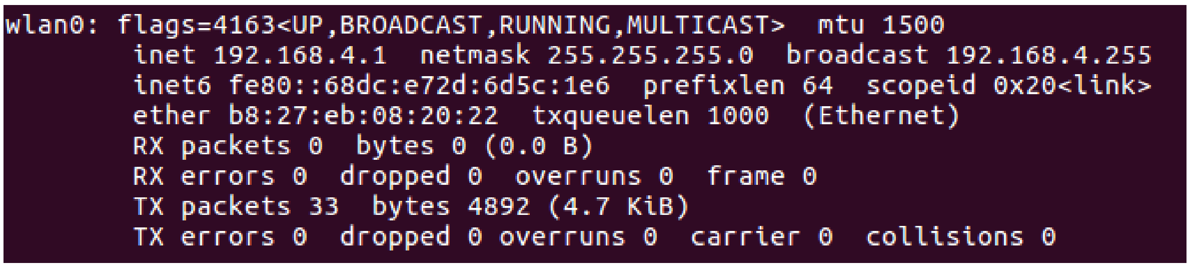
Raspberry Pi needs to get the static IP from DHCP server on wlan0 interface. In this case dnsmasq acts as DHCP server and dhcpcd as a client. For static IP, the dhcpcd.conf file was edited Figure 33.



```
pi@raspberrypi: ~
  GNU nano 2.7.4                 File: /etc/dhcpcd.conf

#static routers=192.168.1.1
#static domain_name_servers=192.168.1.1

# fallback to static profile on eth0
#interface eth0
#fallback static_eth0

interface wlan0
        static ip_address=192.168.4.1/24
        nohook wpa_supplicant
```

Figure 33.: Dhcpcd.conf file

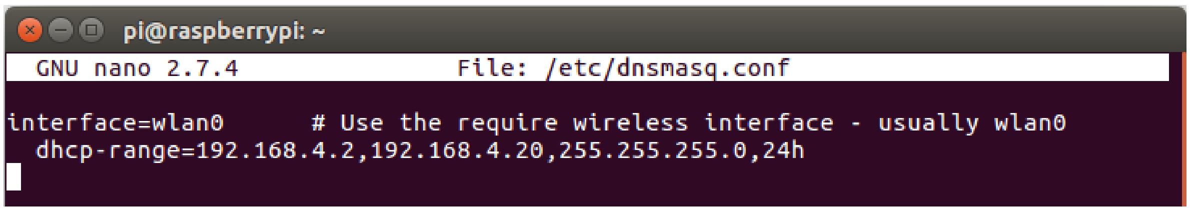
After starting dhcpcd ("sudo service dhcpcd start"), new wlan0 configurations will be seen when interfaces are listed with ifconfig. The wlan0 is now setup as access point with ssid "Multidetector", password "raspberrypi" and with IPv4 192.168.4.1. See wlan0 parameters on Figure 34.



```
wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.4.1  netmask 255.255.255.0  broadcast 192.168.4.255
        inet6 fe80::68dc:e72d:6d5c:1e6  prefixlen 64  scopeid 0x20<link>
        ether b8:27:eb:08:20:22  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 33  bytes 4892 (4.7 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure 34.: Wlan0 parameters

To complete network settings, there is need to edit dnsmasq.conf file. The file gives dnsmasq DHCP server information about interface on which should provide IPs, range of available IPs, mask and expiration time for IP. In this case will provide on wlan0 IP addresses between 192.168.4.2 and 192.168.4.20 with a lease time of 24 hours. The IP address 192.168.4.1 is not included because it is assign statically to Raspberry Pi. All content of dnsmasq.conf file was deleted and replaced with Figure 35.



```
  GNU nano 2.7.4              File: /etc/dnsmasq.conf

interface=wlan0      # Use the require wireless interface - usually wlan0
  dhcp-range=192.168.4.2,192.168.4.20,255.255.255.0,24h
```

Figure 35.: Dnsmasq.conf file

In this stage, Wi-Fi hotspot was ready and can be started with commands:

```
sudo systemctl start hostapd
sudo systemctl start dnsmasq
```

Now any device can connect to Wi-Fi hotspot with ssid=multidetector and pw=password, the IP of Multidetector is 192.168.4.1.

11 Designing Software for Multidetector

11.1 Choice of Programming Language

There are lot of programming languages in which the Raspberry Pi can be programmed on. Best suited and recommended for it is Python because there are a lot of libraries and it is most used by Raspberry users, so it is mostly documented and described.

“Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms and can be freely distributed.” [36].

There are two versions of Python which are still supported in the time of writing this work. Python 2 and Python 3. These two versions have a very similar syntax with small changes, but Python 3 is newer and will be supported in the future whereas the support for Python 2 will end in 2020 [37] and therefore Python 3 was chosen as the main programming language.

Python 3 is preinstalled in Raspbian. There was just a need to install pip3 which is program that ensures installation and management of Python libraries.

11.2 The Logic Behind Detectors Processes

To solve the problem with multiple detectors trying to run at the time and saving data into one common file, it was decided to run them concurrently as processes or threads and in main program loop just manage variables. So, each detector would run independently on each other and on main loop. Data would be gathered, processed and saved within the main loop.

This will have advantages not only in easier management of detectors and data saving, but also in performance of whole Multidetector. If all detectors would be treated as one code and run consecutively, there would be performance leaks each time the detector would be waiting for some physical process to be done. As an example, let’s consider ultrasonic sensor. When sensor sends sound wave, it waits until it hears echo and in this waiting time, whole system would be stopped. In other hand, if Ultrasonic detector would be run concurrently (a s separate thread or process), other tasks could be done in the time window when the sound is travelling back and forth to the obstacle. This would come handy also in cases when there are added delays between measurements of detectors (for example measure temperature and humidity once in 5 minutes).

The Python 3 has two standard ways to handle multiple tasks. The threading and the multiprocessing libraries and although their implementation is very similar, the final result in performance and logical structure program is not. [38]

11.2.1 Threads vs Processes

There are libraries “threading” [39] and “multiprocessing” [40], which handles threads within processes and processes This libraries documentation explains this topic pretty well.

When the threat is created, it is spawned within the existing process (in shared memory space), therefore starting a new thread is faster than to start a new process which is created with separate memory space. In the other hand, a process with its own memory space is independent on its parent process.

Threads are working in most ways as functions. The process in which they are running can pass them variable as an argument and the thread can access and change this variable because process shares the memory with it.

On the other hand, when new a process is created, it is done in a new memory space and therefore children process can’t access and change variables of parent process. When variable is passed to a new process in arguments, it is passed by copy, not by reference.

After the testing to get some experiences and better overview on this topic, it was decided to use processes for individual detectors. Processes are slower to start, but this will be done on initialization of Multidetector and user will barely notice it. Advantages are that processes will run concurrently independently on each other and on parent process. This will boost performance of multidetector and is one process will fall down, others will be still operating properly. After including library “multiprocessing” into the code, the new process can be created and started:

```
detectorProcess = multiprocessing.Process(target=detectorFunctioin, args=arguments)
detectorProcess.start()
```

11.2.1.1 Sharing Variables

To share variables between two processes, this variable must be a special type which ensure that variable has its own memory space and both processes has access to it. This can be done by object multiprocessing.Array() from multiprocessing or with third party library (queues or multiprocessing pools).

There will be need to create shared variable for each process. Children process will change value of this variable and parent process will read it. Until children process will not change a value again, main loop will keep reading same values and won't save them (if some of other values from different detector process won't change).

The shared variable will be type array of chars so there can be stored text or numbers in it. The maximal size of array was set 20 chars, but this parameter can be changed later if required.

```
sharedVariable = multiprocessing.Array('c', 20)
```

11.2.2 Race Condition

"A race condition is a behavior which occurs in software applications or electronic systems, such as logic systems, where the output is dependent on the timing or sequence of other uncontrollable events. Race conditions also occur in software which supports multithreading, use a distributed environment or are interdependent on shared resources. Race conditions often lead to bugs, as these events happen in a manner that the system or programmer never intended for. It can often result in a device crash, error notification or shutdown of the application" [41].

This problem is very serious when talking about concurrency or parallel programming. This can cause unexpecting changes of data. In most cases this could be solved by GIL which prevents more than one thread running at a time "In CPython, the global interpreter lock, or GIL, is a mutex that protects access to Python objects, preventing multiple threads from executing Python bytecodes at once." [42]. Processes doesn't share GIL between them, so they can run truly concurrently. With threads, they share GIL with parent process, so the fully controlled concurrency is not possible here. So, although GIL could prevent system from errors, it would slow it down and with processes we don't have to manage race condition or GIL.

Additionally, the main loop will be much faster than loops of processes (should be at last twice faster so it will catch every value) so there will be no problem with need to manage data streams in this variable from side of detector process.

11.3 Code Composition

There were algorithms designed earlier in this work in chapter [9.2, Figure 26.: Main program of Multidetector] and the coding will take these algorithms as a template. each part of the main algorithm must be programmed so the requirements on software are fulfilled.

The code will be divided into two files for better transparency. These will be "multidetector.py" [Attachment.: The Main Program "multidetector,py"] and "detectors.py" [Attachment.: Library "detectors]. Additionally, there will be a file for Multidetector configuration "multidetector.conf". There were created custom functions for working with files into which data will be fetched. These functions are available in "filesManipulation.py" [Attachment.: Configuration File "multidetector.conf"].

The main program will be written in "multidetectors.py" file and it is this file which will become executable and will be run to start process of measurement. This file will import the "detectors.py" file and load variables set by user from "multidetector.py"

11.3.1 Multidetector.py

At the beginning, the program check if was run with administration rights (as root user) by checking user id which is for root user always "0". This condition must be true to prevent later errors and allow program to change some of Raspberry Pi's settings.

```
userid = os.getuid()
  if username != 0:
    print("this program MUST be run from root user.")
    exit()
```

11.3.1.1 Read multidetector.conf

Next task main program have is to load data from „multidetector.conf" file. There is a library „configparser" which is meant for loading and editing configuration files. In code will be used function SafeConfigParser to load and edit data as following:

```
parserConf = SafeConfigParser()
parserConf.read(“multidetector.conf”)
parserConf.set(“sectionName”, “variableName”, “variableValue”)
parserConf.get(“sectionName”, “variableName”)
```

11.3.2 Initialization of Processes

The important part was to solve problem with initialization of processes so these could be started up with all arguments required. Each process can have different number of arguments and therefore from „multidetector.conf“ was loaded with „configparser“ variable with list of active detectors and then are active detectors found in list of detectors processes with listed arguments also in „multidetector.conf“ with help of variable of type dictionary which connects name of process (string) with process (function):

```
knownProcesses = {
 “ultrasonicProcess“: ultrasonicProcess,
 “switchProcess”: switchProcess,
 “soundProcess”: soundProcess,
 “bluetoothProcess”: bluetoothProcess
}
```

With this detector processes (functions) are saved into a list as „processVariable“ along with „name“, „processArguments“ and „sharedVariable“ which will be discussed later. Next, processes are started and will run until the main program stops. By this, the „initialization of measurement“ block from [Figure 26.: Main program of Multidetector] is done and program can continue into next part of the main algorithm.

11.3.2.1 Initialization of Data Files

The file in which data will be written in need to has a header. Multidetector.py will use the custom written library which will do following to initialize file:

- Get needed information about measurement from „multidetector.conf“
- Find the last file in the row of files starting with „filesGroup“
- Create a new file
- Create header from information about measurement
- Close file and return its path

11.3.2.2 The Main Loop

The main loop will run until the user indicates end of measurement. This loop responds to the Figure 29.: Collecting data from detectors block in main algorithm. Each iteration program will consecutively check the „sharedVariable" of each detector and if it changed, it will store all values from detectors into line in file.

```
activeDetectors[i]["sharedVariable"].value.decode("utf-8")
filesManipulation.writeValues(filePath, arrayInMemory)
```

To save data, program is using already mentioned custom library „filesManipulation.py" explicitly function „writeValues()" [Attachment.: Library "filesManipulatipon"].

11.3.2.3 Accessing GPIO Pins

To communicate with GPIO pins, there is Python library RPi.GPIO which helps with reading and controlling individual pins. At start, GPIO needs to be configured with BOARD (physical) or BCM (channel) numbering.

```
GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)
GPIO.setup(GPIO_BOARD_NUMBER, GPIO.IN)
```

11.3.2.4 Running Measurement

Measurement can be run by starting it with python3 command:

sudo python3 ./multidetector.py

But for easier access and controll, the file "multidetector.py" was edited to be executable by changing mode to -x:

```
!/usr/bin/python3
```

And by adding at the first line of code:

```
chmod +x myScript.py
```

With this, the file can be started just by typing its name within the same directory.

12 Detectors Implementation

It is finally time to starting with implementation of individual sensors and detectors into Multidetector system. All chosen sensors and detectors will be described from hardware and view and will be described how this will be used for integration.

12.1 Switch and Button

This first type of detectors is the simplest one. From technical part, it is only the switch connecting two wires which then change the voltage level on input digital pin.

When implementing switch on Raspberry Pi, there could be problems with reading direct voltage on input GPIO pin. For avoiding short circuit, there must be added a resistor.

Additionally, it has been tested, that when the pin is connected directly to the 3.3V, there are still some small changes of voltage in circuits of RPi and therefore the program is randomly switching from 0 and 1 even when switch haven't changed at all. To avoid this problem, loop is grounded and the default value on pin is 3.3V, high. When switch ground the circuit, the voltage on pin decrease and program recognize state as false. See at Figure 36.: Circuit of Buttom and Switch and Figure 37.: Implementation of Button

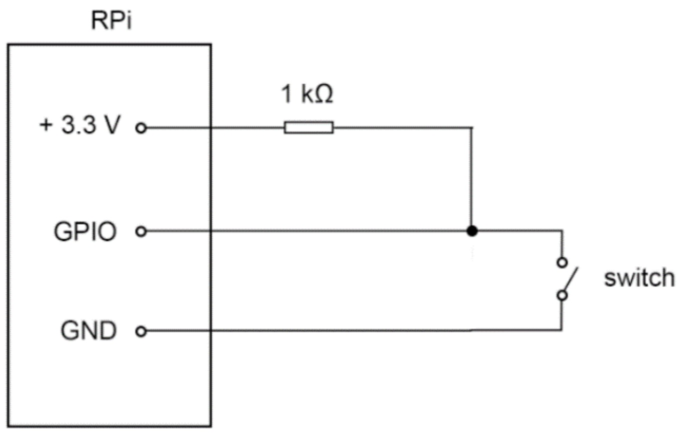


Figure 36.: Circuit of Buttom and Switch

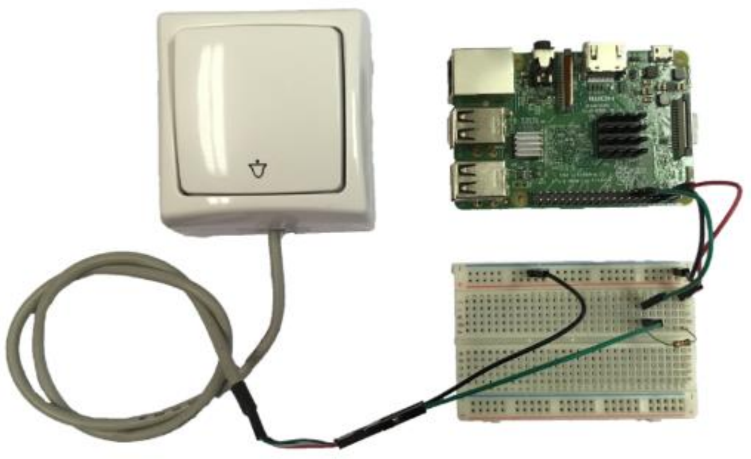


Figure 37.: Implementation of Button

From programming view, this problem is simple. First the program needs to read state in a pin. Due to the fact of the circuit mentioned above, there is part in program which swap 0 and 1 so data are easier to understand.

```
state = GPIO.input(GPIO_BOARD_SWITCH)
if state == 0:
  state = 1
else:
  state = 0
```

this whole process is periodically repeating in interval given as argument of a function. The state is in each iteration stored into “sharedVariable” so main program can read it and process it.

```
sharedVariable.value = str(state).encode(“utf-8”)
```

All of it is wrapped in function “switchProcess()” which is in file “detectors.py” [Attachment.: Library “detectors”]

12.2 IR Sensor FC-51

FC-51 sensor consists of IR Emitter LED, IR Receiver, and circuit controlling both. The part of the circuit is potentiometer which can adjust detection distance. There are three pins for connecting to a sensor, VCC, GND and OUT for reading state of sensor.

In this situation, the FC-51 is connected straight to the RPi. VCC on 3.3V pin, GND on GND and OUT to GPIO which will read state and process it. See Figure 38.: Circuit of IR Sensor FC-51 and Figure 39.: Implementation of IR Sensor FC-51.

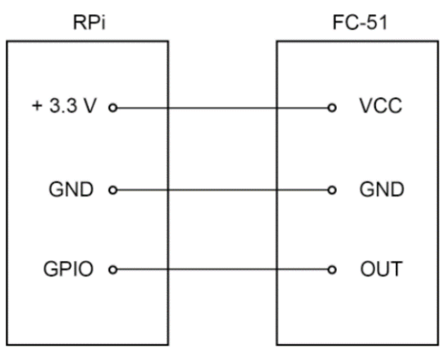


Figure 38.: Circuit of IR Sensor FC-51

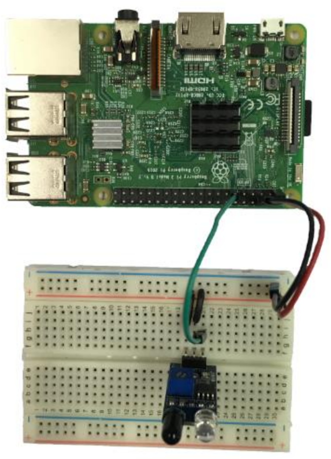


Figure 39.: Implementation of IR Sensor FC-51

There was no need to write a new function to implement this sensor. This sensor is simply switching from true to false, so the “switchProcess” function was used, and tests showed, that it is suitable for this hardware configuration.

12.3 Ultrasonic Sensor HC-SR04

Ultrasonic sensor HC-SR04 is a device able to send sound and listen to the echo and this will be used to measure distance to obstacle. The hardware consists of speaker, microphone and circuits to control them.

It is able to detect operate on distance between 2cm to 4m with measuring angle 15 degree [43]. The device was tested, and the accuracy showed to be about 0.5cm, which is for transportation applications where ultrasonic detection is used enough.

12.3.1 Hardware Part

There are four pins, VCC 5V, Trigger, Echo and GND. It transmits sound waves whenever the Trigger pin is "high" and sets Echo pin high whenever it is receiving sound. Both transmitter and receiver are set to same frequency 10μs (TTL pulse), so the device detects only sound echoed from obstacle [43].

Because the HC-SR04 is working on 5V voltage, there must be solved the interface with Raspberry Pi. There is 5V pin on pins header of RPi which serves only as power supply. The trigger pin of HC-SR04 can operate with voltage 3.3V which can every GPIO pin. The problem is in receiving information from Echo pin because pulses from this pin are 5V and this voltage could cause harm to chip of Raspberry Pi. Therefore, there must be added a voltage divider between Echo pin of HC-SR04 and input pin of Raspberry Pi. See Figure 40.: Circuit of Ultrasonic Sensor HC-SR04 and Figure 41.: Implementation of Ultrasonic Sensor HC-SR04.

The resistors of divider are set on 2000Ω and 1000Ω which gives voltage 3.333V and that is low enough for Raspberry Pi to handle. See formula below:

$$U_{GPIO} = U \,.\, \frac{R_1}{R_1 + R_2}$$

$$U_{GPIO} = 5 \,.\, \frac{2000}{2000 + 1000} = 3.33\bar{3}V$$

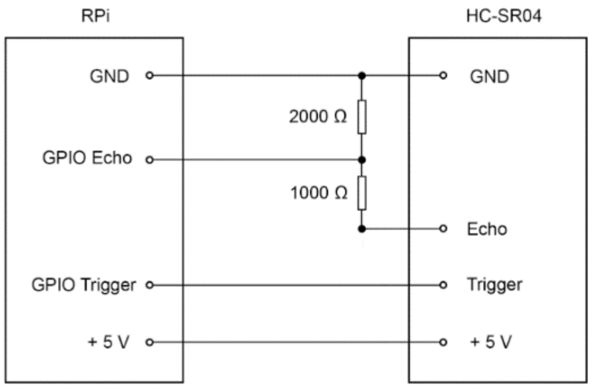


Figure 40.: Circuit of Ultrasonic Sensor HC-SR04

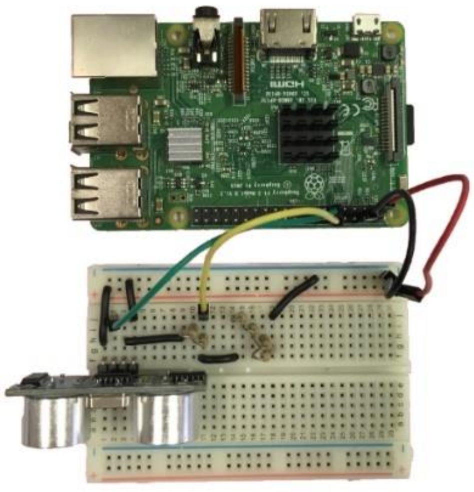


Figure 41.: Implementation of Ultrasonic Sensor HC-SR04

12.3.2 Software Part

First, it was need to write function for a one-time distance measure. On the trigger pin is sent a signal long 10µs.

```
GPIO.output(GPIO_BOARD_TRIGGER, True)
time.sleep(0.00001)
GPIO.output(GPIO_BOARD_TRIGGER, Fasle)
```

After the signal is sent, program waits till it detects response on Echo pin and listen how long it takes to a signal to return back.

```
while GPIO.input(GPIO.BOARD.ECHO) == 0:
  StartTime = time.time()
while GPIO.input(GPIO.BOARD.ECHO) == 1:
  StopTime = time.time()
```

From duration of sound wave travel and speed of sound wave in air (343 m/s), it can be counted the distance from obstacle. See formula below:

$$l = \frac{t * 34300}{2}$$

```
TimeElapsed = StopTime – StartTime
distance = (TimeElapsed * 34300) / 2
```

To repeat this process, there was created function “ultrasonicProcess” which will initialize properties of ultrasonic measurement and run infinite loop which periodically checks distance and each iteration sets “sharedVariable” to the new measured value.

Additionally, there was written a piece of code which allows to set a “null” value which indicates some constant distance with which other distances are compared (pavement and vehicles). This helps to separate data which are not relevant (so there are written only values where vehicle passed by).

```
if distance < null_distance:
  distance = “null”
```

12.4 Volcraft SL-451 Sound Level

Volcraft SL-451 Sound Level is the good opportunity to test and prove, that Multidetector will be able to process analogue data. This detector can provide a different voltage levels dependent on measured sound level. There is a 3.3mm jack output which has three contacts. Ground for reference voltage level and analogue alternating or direct current. [14]

To be able to access contacts on female jack, there was created cable with male jack on one side and male pins on other which fits into standard bread board Figure 42.: Jack Connection into Volcraft SL-451 Sound Level.

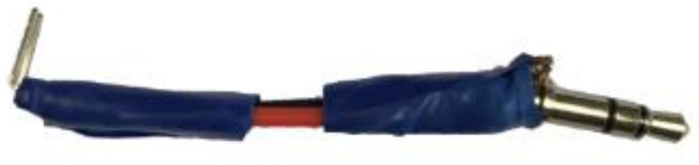


Figure 42.: Jack Connection into Volcraft SL-451 Sound Level

Another problem is, that Raspberry Pi has no analogue input, so there was need to add module which have ADC and thus can sample analogue signal into digital. Chosen module is ADC DAC Pi Zero which is designed for Raspberry Pi and because it fits RPi header, it can be easy plugged in and use.

ADC DAC Pi Zero has 2 channels 12 bites analogue to digital converter. Max sampling rate is 12, 000 samples/second which is more than enough for measuring sound levels of traffic. [44]

When the module is plugged in, it is still possible to use GPIO pins, but not all of them. To see which pins are allowed to use, there was need to check schematics of ADC DAC Pi Zero to see which pins are not connected to any parts of module, see Figure 43. Schematics Pins that can be still used are 3,5,7,8,10,11,12,13,15,16,18 of board numbering. The functionality will be tested with other detectors.

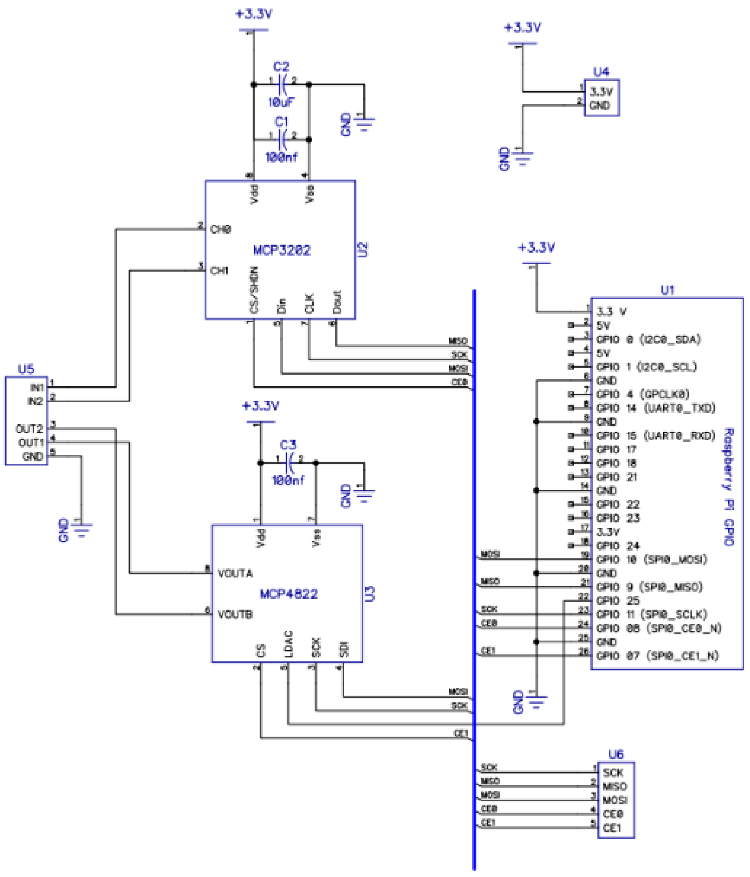


Figure 43.: Schematics of ADC DAC Pi Zero [45]

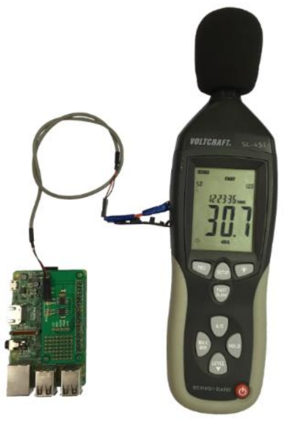


Figure 44.: Implementation of Volcraft SL-451 Sound Level

12.4.1 Software Part

“The ADCDAC Pi uses the SPI port” [46] and for better connection to board we need to use py-spidev python module and ADCDACPi library. So, there need to be done three things to prepare Raspberry Pi for analog digital converter.

The SPI is disabled in default configuration of Raspbian, to enable it there is need to run configuration with command “raspi-config” and in interface options enable SPI [47]

The package spidev for controlling SPI from Python code can be downloaded from repository `https://github.com/doceme/py-spidev.git` and installed by following installation instructions [47].

Next step was to install actual package written for this particular board [3]. These can be downloaded on repository and installed by following instructions on [46].

The library is imported into code and initialized by following commands:

```
from ADCDACPi import ADCDACPi
adc1 = ADCDACPi(1)
adc1.set_adc_refvoltage(3.3)
```

In this scenario, the board will activate line 1 and will set relative voltage 3.3V. This voltage can be changed if calibration would be needed.

The voltage levels are available by:

```
adc1.read_adc_voltage(1, 1)*100
```

Values are periodically loaded into shared variable, so the main loop can access them.

12.5 Pico Iteris Video Detector

Pico Iteris Video Detector has been mentioned multiple times in this work. The hardware is described in [3]. The only change is in removal of additional LEDs which worked only as visual control that system is working properly. There will be GUI in the Multidetector system which will fully replace control LEDs.

So, interface will be built by wires and one 1kΩ resistor. The circuit can be seen on Figure 45. This circuit will connect one Pico Iteris output (related to one virtual loop) with one GPIO pin. For N virtual loops, there is need to build such a circuit N times.

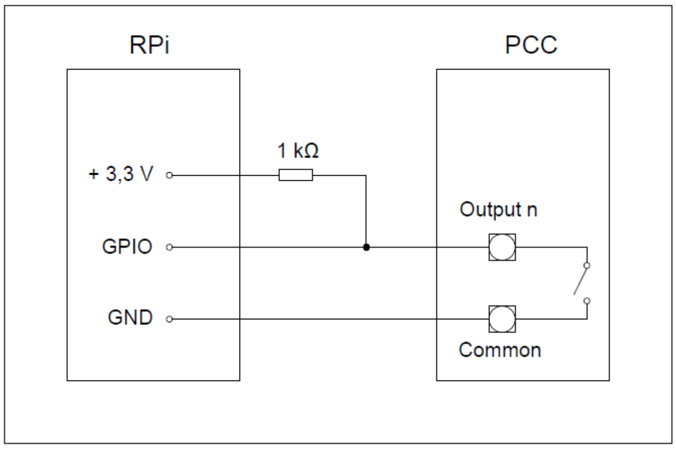


Figure 45.: Circuit of Connection RPi with Pico Iteris PCC [3]

The system from [3] has been improved with better battery (much lighter for measurement on terrain) and monitor which can be plugged in directly to PCC board of detector Figure 46.

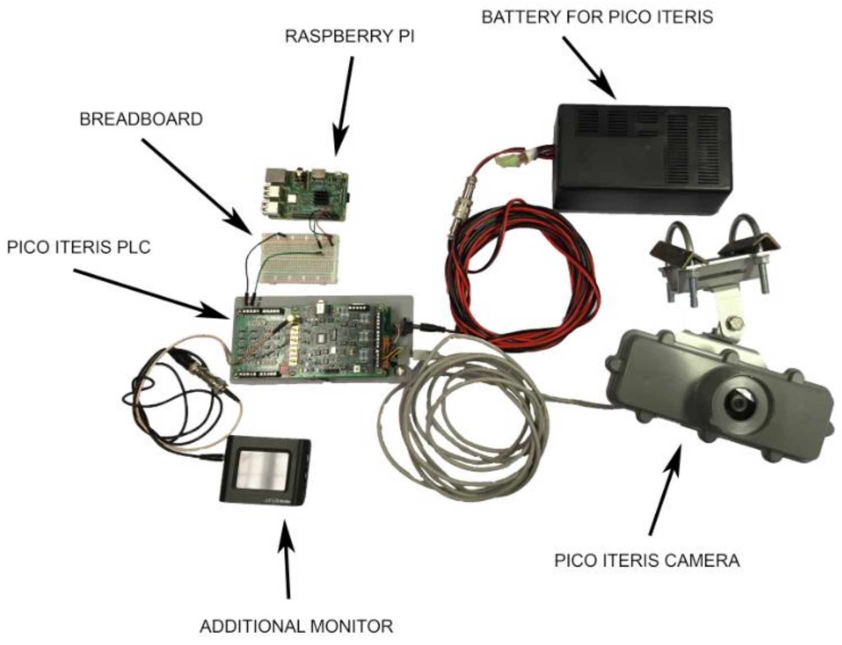


Figure 46.: Setup of Pico Iteris for Multidetector

From the software view, the program will threat the inputs from Pico Iteris same as the simple switch or button described above in the chapter [Switch and Button], so there is no need for further discussion on this topic.

12.6 Ubertooth One

12.6.1 Hardware Part

Ubertooth One was selected as a main device to catch Bluetooth communication. This device doesn’t need any further hardware adjustments, it will just plug into Raspberry Pi USB socket, see Figure 47. All properties of Hardware components of this device can be seen in [16].

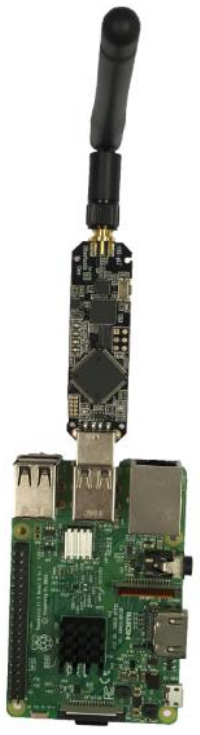


Figure 47.: Implementation of Ubertooth One

Before use the Ubertooth, the proper software needs to be installed on Raspberry Pi. There few libraries that needs to be installed: “lbbtbb” and “ubertooth”. the detailed installation process is described step by step on [16].

Programming of process for this device will be slightly different because in this case there is urging to record each one of the Bluetooth addresses. In case the Ubertooth will read more than one addresses, and loop wasn’t fast enough to process them some information would be last. Therefore, will be in process created Threat which will stack catch addresses into queue and process will consecutively send them to the main loop with given intervals.

For there was created function represented thread, this function initializes and run program on Ubertooth One which will pipe lines of catch communication.

```
process = subprocess.Popen(['ubertooth-rx', '-z'], stdout=subprocess.PIPE, bufsize=1)
```

The line is saved into variable and processed so there is pure LAP part of address with system time when addresses was catch (code below is not complete, whole code on [Attachment.: Library “detectors”]).

```
for line in iter(process.stdout.readline, b’’)
line = line.decode(encode=”utf-8”)
line = line.split(‘ ’)
re.match(r”^systime”, column)
re.match(r”^LAP”, column)
```

Additionally, there was written a function „isWaiting()“ [Attachment 3.: Library “detectors”] recycling a LAPs which were already caught in last ten seconds. This prevents data from row of same LAPs.

The main bluetoothProcess will get LAPs from queue and one by one sends them to main loop of detectors.

```
sharedVariable.value = arrayOfLaps.pop(0)
```

13 Testing of Multidetector

To prove that designed Multidetector system is working, it needs to be tested. These tests will show, what Multidetector can do and where the future attention should be directed to make it better.

In the laboratory conditions, each detector should be tested separately to find out if all detector processes work properly. After proving that collecting data works, there should be run tests with multiple detectors connected at once to test initialization of Multidetector and if it is able to collect data from more than one detector process, process these data and finally save them.

Testing in real conditions will show if Multidetector is capable of working with real vehicles. These tests won’t be done to test working individual detectors technologies or process data to get information about traffic (intensity, density, etc.) so the tests won’t necessary be long term.

13.1 Laboratory Testing

In the laboratory, there were ran tests of each individual sensors and detectors multiple times, see tests on Figure 48. Due to fact that all sensors and detectors were regularly tested during design and implementation, there were no problems and whole system was working correctly.

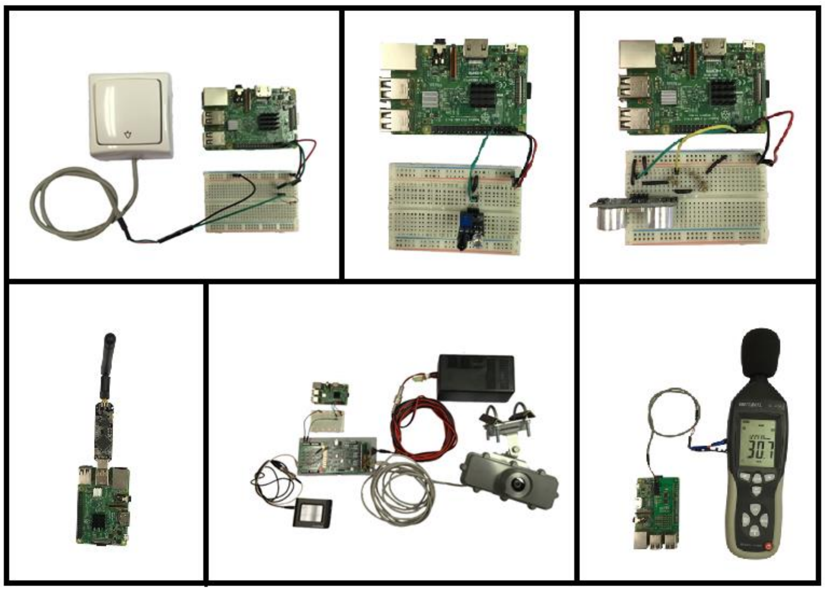


Figure 48.: Testing of individual devices

To test if the Multidetector works properly while multiple sensors and detectors are connected, there were chosen few members capable of measuring meaningful data even in laboratory conditions. These members were:

- Button
- Active IR
- Ultrasonic

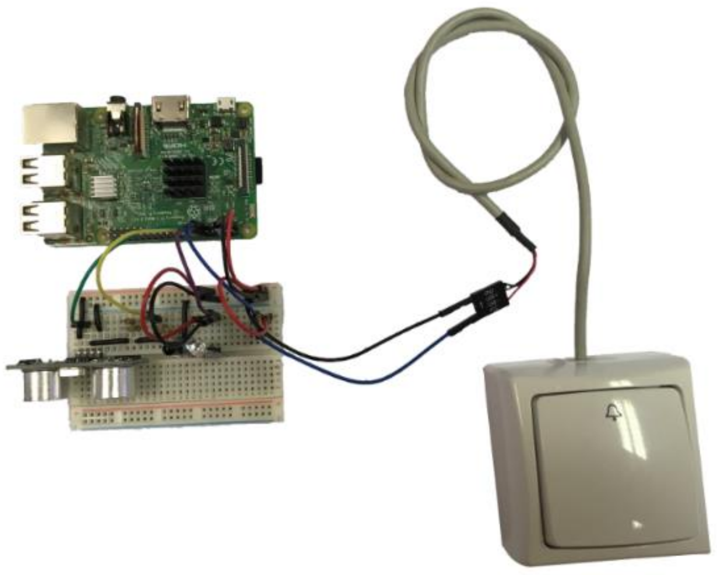
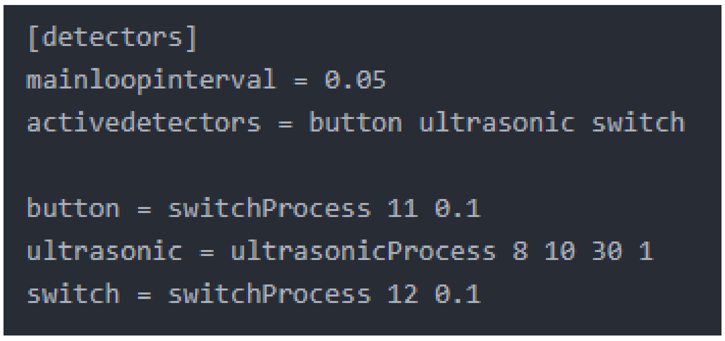


Figure 49.: Multidetector system setup for laboratory testing

First thing to be done was to properly edit “multidetector.conf” file. The main loop period was set to 0.05 s, which is fast enough so no data were lost. There was set three detectors named “button”, “ultrasonic” and “switch” for IR sensor. The button was set to 11th pin, ultrasonic trigger on 8th, echo on 10th and IR sensor state on 12th pin. For ultrasonic sensor was set longer period to simulate situation when data from detector processes are not coming at the same time. See multidetector.conf setup for laboratory tests on Figure 50.

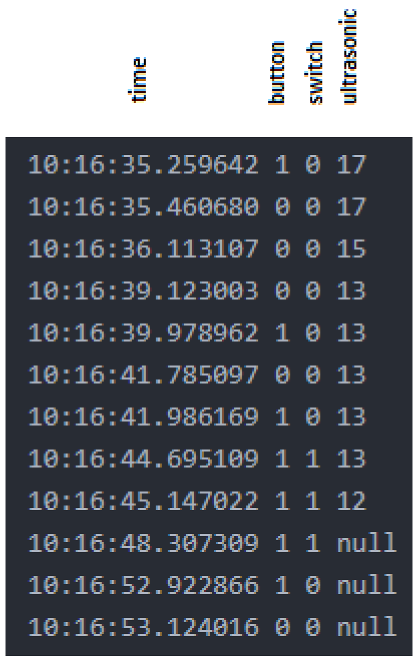


```
[detectors]
mainloopinterval = 0.05
activedetectors = button ultrasonic switch

button = switchProcess 11 0.1
ultrasonic = ultrasonicProcess 8 10 30 1
switch = switchProcess 12 0.1
```

Figure 50.: Multidetector.conf for laboratory setup

The test was run multiple times and there was created an artificial load simulating real measurement. Button represented pedestrians request button, ultrasonic sensor represented situation on parking slot and IR barrier some sensor or detector detecting occurrence of vehicle.

There was an effort to have switched on all detector with each other, so the real data integration is putted into test. Most important was to watch data in real time and check that all changes were written into file (by change it is meant switch from zero to one or change of value coming from detector process). The data of one of the measurements are in [Attachment.: Laboratory Measurement File “testMeasurement1.txt”]. The example from this file see at Figure 51.



```
10:16:35.259642 1 0 17
10:16:35.460680 0 0 17
10:16:36.113107 0 0 15
10:16:39.123003 0 0 13
10:16:39.978962 1 0 13
10:16:41.785097 0 0 13
10:16:41.986169 1 0 13
10:16:44.695109 1 1 13
10:16:45.147022 1 1 12
10:16:48.307309 1 1 null
10:16:52.922866 1 0 null
10:16:53.124016 0 0 null
```

Figure 51.: Sample results from laboratory testing

On a Figure 51, there is a null value in data on fourth column. This doesn’t indicate that the ultrasonicProcess has an error, but there was set a null value which should represent a pavement distance. When there is no car detecting, ultrasonicProcess keeps sending null value which prevents data from meaningless rows.

All output data were checked and there was no error encountered during the measurement.

13.2 Measuring in Terrain

The second measurement was testing Multidetector on real conditions. Chosen place was the same as in [BC work], so there is no need for further description. Measurement was done 8th of October 2018 as 9:00 h.

The testing configuration consists of Ubertooth One, video detector Pico Iteris (two virtual loop each on one line) and button. Ubertooth One was catching Bluetooth addresses and was simulating vehicle identification. Pico Iteris was counting vehicles at each line separately and button could represent pedestrian’s signalization, but in this scenario, it served as control counter operated by human.

There were chosen four detectors for measurement in the terrain:

- Ubertooth One
- Volcraft SL-451 Sound Level
- Button
- Video detector Pico Iteris

13.2.1 Hardware Preparation

The whole setup was prepared beforehand at home and each detectors connection into system tested individually. The day before measurement all batteries were fully charged up and tested their functionality.

After coming to the measurement spot, whole system was set up and turned on. The camera was mounted into spot so the whole communication was in field of view. Virtual loops were set with program “PICO Setup Tool” and checked their functionality. See virtual loops placement on Figure 53.

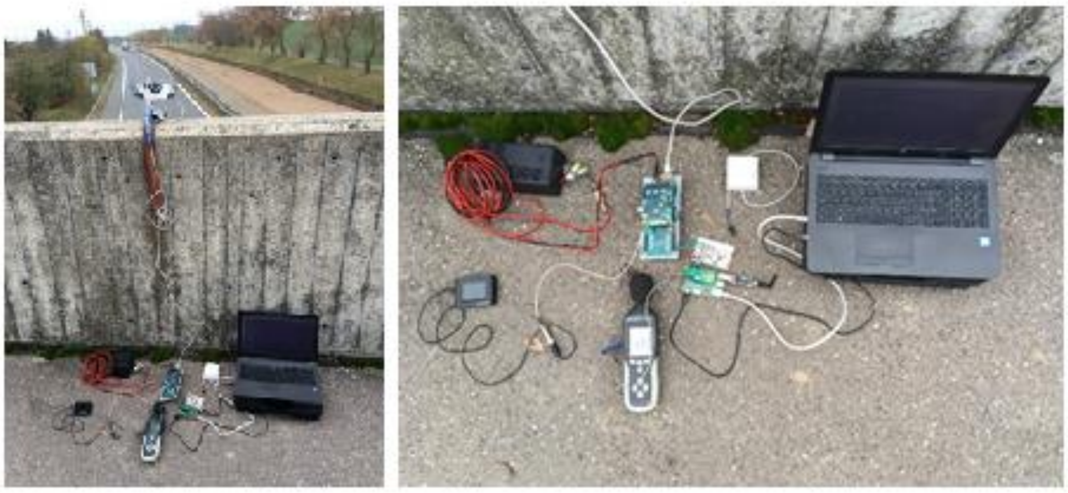


Figure 52.: Measurement setup

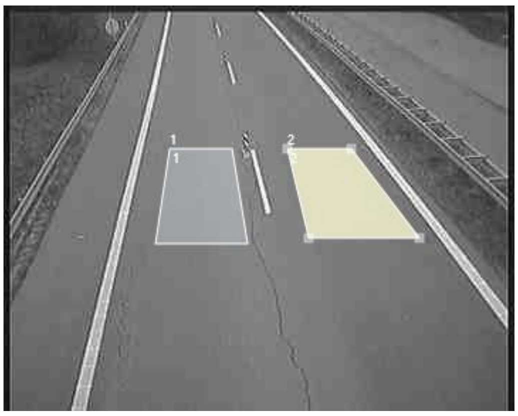
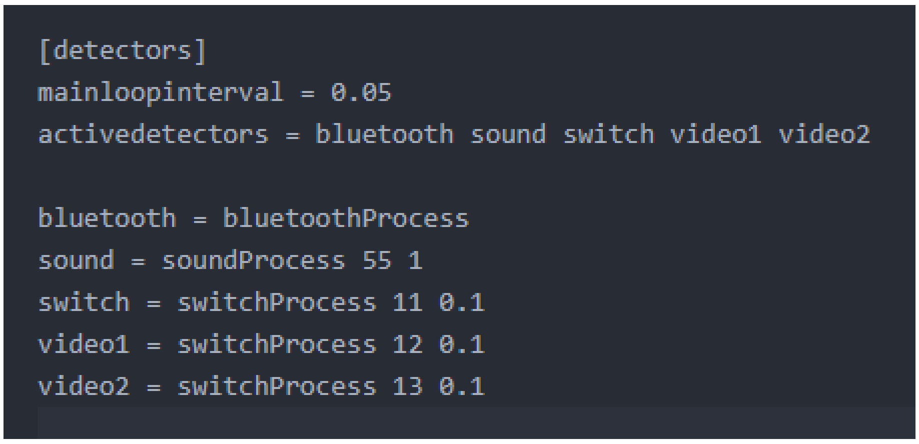


Figure 53.: Virtual loops of Pico Iteris placement

13.2.2 Software Preparation

As with laboratory measurement, even here is need to edit “multidetector.conf” file to setup measurement, see at Figure 54.


```
[detectors]
mainloopinterval = 0.05
activedetectors = bluetooth sound switch video1 video2

bluetooth = bluetoothProcess
sound = soundProcess 55 1
switch = switchProcess 11 0.1
video1 = switchProcess 12 0.1
video2 = switchProcess 13 0.1
```
Figure 54.: Multidetector.conf for laboratory setup

13.2.3 Process of Measurement

The measurement needed to be run multiple times due to some problems with connection of Pico Iteris. Final part last 15 minutes and the only encountered issue was with Voltcraft sound level meter where it went into sleep mode after 10 minutes and had to be woke up.

The whole system was placed the way, so it was easy to see system and traffic at once see at Figure 55. The system was switched on and off few times to see its behavior at the booting process. After that, system was left few times on for longer period and whole time was observed its functionality.

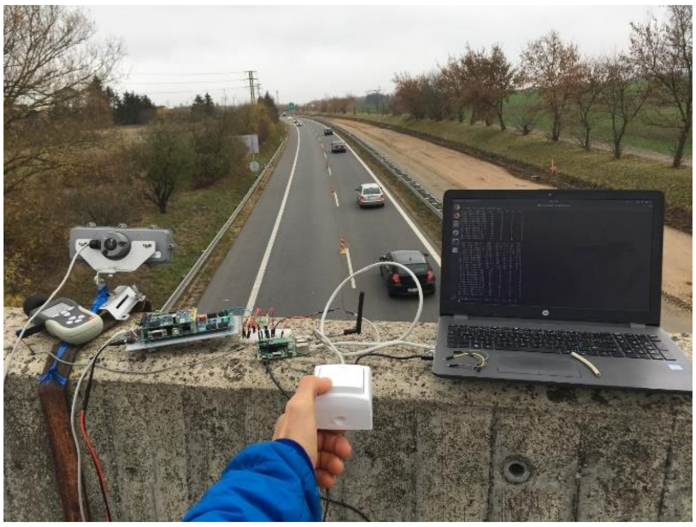
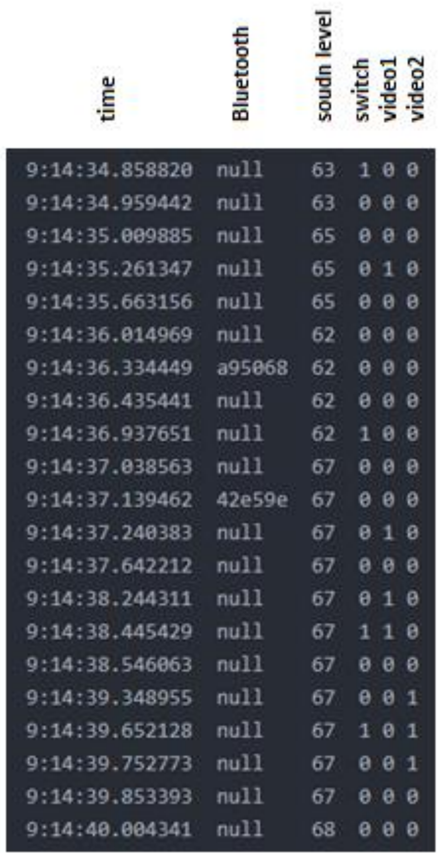


Figure 55.: Process of measurement

13.2.4 Results of Measurement

One of the measurement files can be seen at [Attachment.: On Field Measurement File "testMeasurement2.txt"]. In this file it can be seen that whole system was working correctly. On sample data Figure 56, we can see that on change of each detector, the whole row is saved even though anyone else stayed same. It can be also seen that the period of sound detector is greatest of all, so it is staying at one value for a longest time.



```
9:14:34.858820  null    63  1 0 0
9:14:34.959442  null    63  0 0 0
9:14:35.009885  null    65  0 0 0
9:14:35.261347  null    65  0 1 0
9:14:35.663156  null    65  0 0 0
9:14:36.014969  null    62  0 0 0
9:14:36.334449  a95068  62  0 0 0
9:14:36.435441  null    62  0 0 0
9:14:36.937651  null    62  1 0 0
9:14:37.038563  null    67  0 0 0
9:14:37.139462  42e59e  67  0 0 0
9:14:37.240383  null    67  0 1 0
9:14:37.642212  null    67  0 0 0
9:14:38.244311  null    67  0 1 0
9:14:38.445429  null    67  1 1 0
9:14:38.546063  null    67  0 0 0
9:14:39.348955  null    67  0 0 1
9:14:39.652128  null    67  1 0 1
9:14:39.752773  null    67  0 0 1
9:14:39.853393  null    67  0 0 0
9:14:40.004341  null    68  0 0 0
```

Figure 56.: Sample results from real conditions testing

It was also important to control state of detectors. in one time of measurement, the Voltcraft sound level detector switched into a sleep mode and data stayed for few minutes at the same level. This was caused by detector individual and not by error of Multidetector and therefore this will not be taken as a problem which need to be discussed further.

On bluetoothProcess, there was set a null value which prevents a multidetector loop from writing one value all over again until the Ubertooth won't catch another Bluetooth address. So, from Figure 56 it is clearly visible, that at that time interval, there were two captured addresses “a95068” and “42e59e”.

13.2.5 Process Measured Data

To prove, that Multidetector is really capable to collect meaningful data which can tell us useful information about traffic, data from “testMeasurement2.txt” will be processed.

Data were transferred into Excel and was counted number of caught Bluetooth addresses, presses of switch, signals from video detector and average sound level.

Table 6.: Results from measuring

caught Bluetooth addresses	63
vehicles on switch	251
vehicles on video1	133
vehicles on video2	129
sum on video	262
sound level average [dB]	64

There were 63 of Bluetooth addresses caught which is 24% of total count (taken from video detector). To research if this number is high enough is not a purpose of this work, but in this configuration, Multidetector would be able to identify 24% of vehicles.

There is difference between switch and video count. This was probably caused by multitasking on field. For the whole time of measurement, there was need to instantly check system functionality so often some car pass by without being caught by operator. Again, switch was added into measurement just to add another element into data so there was more complexity in data set.

To better describe data, there was added minute intensity Table 7.: Minute intensity and showed in Figure 57.: Minute intensity switch and video. These values could be influenced by time difference in vehicle capture where one second difference between switch and video can cause a big difference in one interval.

Additionally, it was proved at [3], that with two virtual loops is possible to get a speed of individual vehicles. This tested system could be expanded with sensors and detectors added in this work. These would act as virtual loops and with proper configuration the Multidetector would be able to detect speed of passing vehicles.

Table 7.: Minute intensity

MINUTE	SWITCH	VIDEO1	VIDEO2	VIDEO SUM
1	20	11	10	21
2	15	4	9	13
3	17	10	7	17
4	19	14	7	21
5	18	10	6	16
6	18	12	6	18
7	13	8	5	13
8	11	7	9	16
9	11	2	8	10
10	20	11	7	18
11	17	10	8	18
12	13	5	16	21
13	24	10	5	15
14	15	11	10	21
15	12	4	0	4

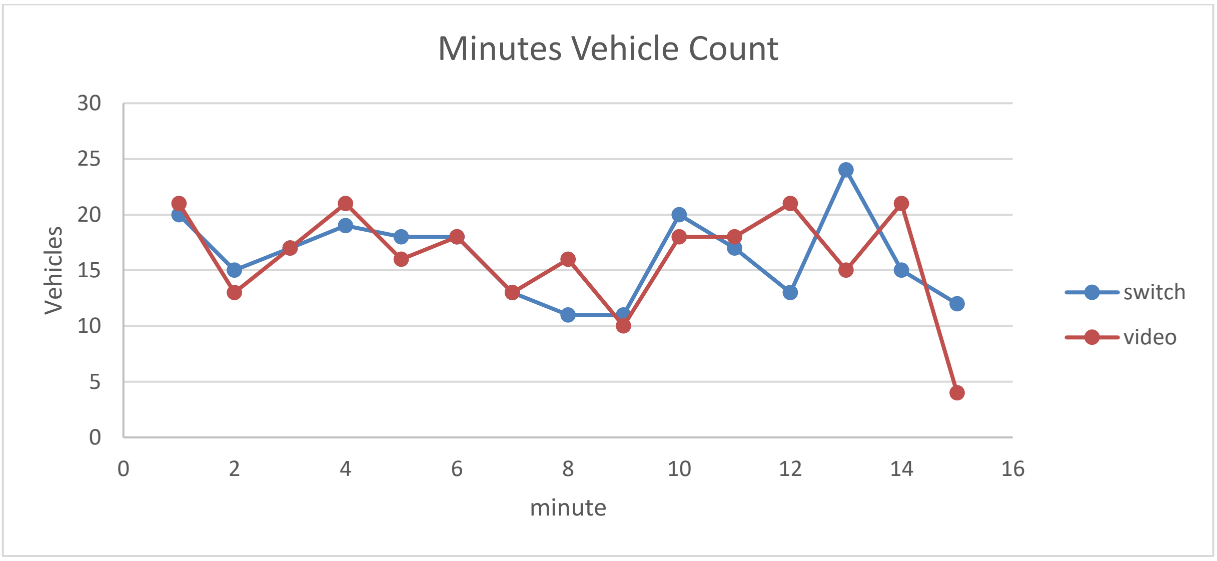


Figure 57.: Minute intensity switch and video

The last measured parameter was sound level. This parameter would give more interesting information in long term research. There was created two charts. Figure 58.: Sound levels shows a whole measurement, where is seen an interval (between 800 and 900 second) where detector switched into sleep mode and data stayed same. On Figure 59.: Sample sound levels is seen shorter interval where individual data are more recognizable.

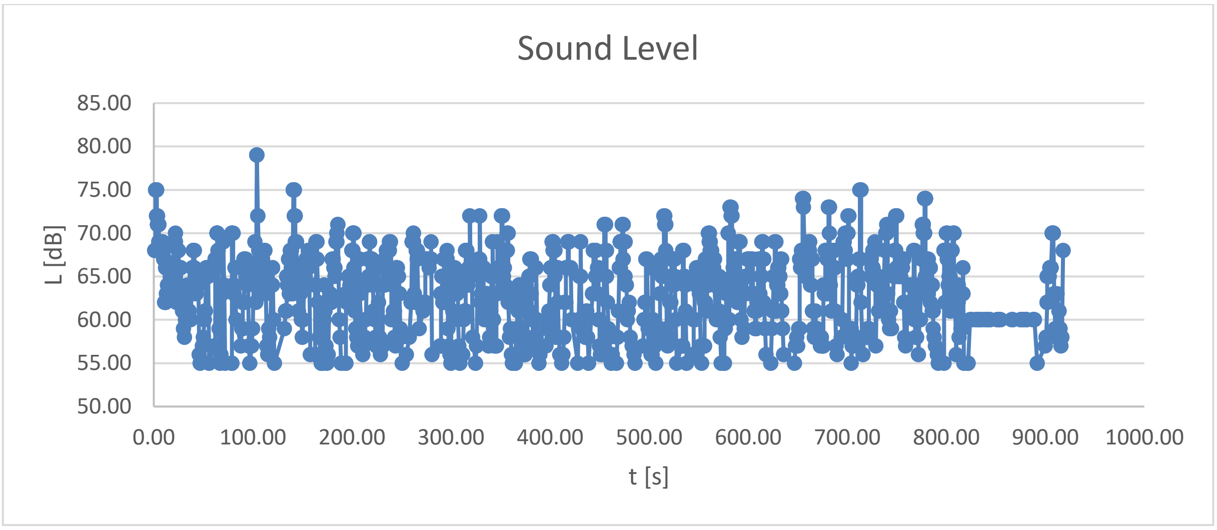


Figure 58.: Sound levels

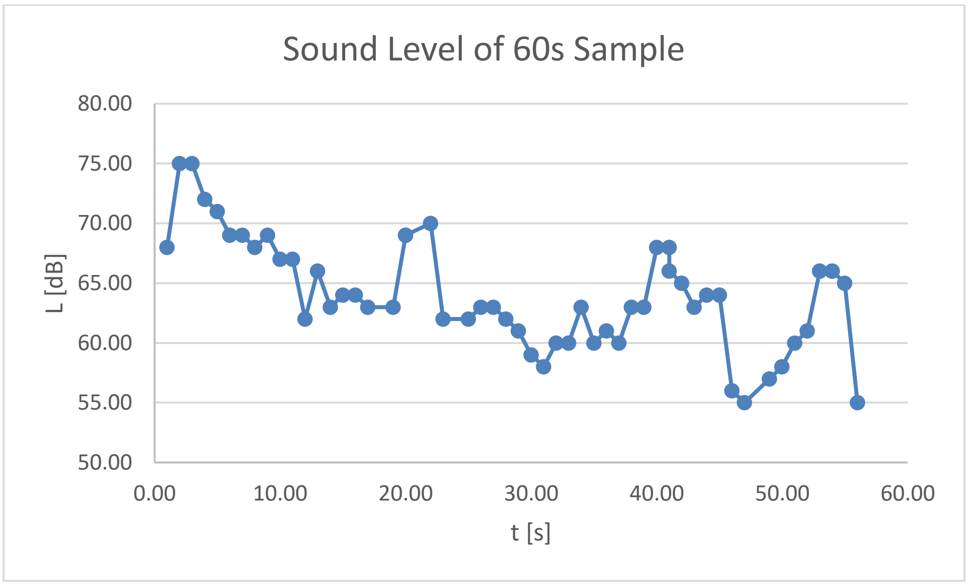


Figure 59.: Sample sound levels

This chapter proves that Multidetector, as was designed, built and tested is fully operative and can be used for a traffic measurement and researches. By this, the whole process of creating a Multidetector at its first version is closed.

14 Conclusion

14.1 The Process of Multidetector Design

The main aim of this work was to find out what requirements should be set on system integrating traffic sensors and detectors, design, build and test such a system. The device which purpose is to do this physical and logical integration was called Multidetector and the whole system Multidetector System. The requirements from sensors and detectors side were strictly set at Physical Integration and Data Integration

There was need to set requirements on hardware on which will be Multidetector running and after defining these, as the best solution for this task was selected Raspberry Pi. The rest of this work was aiming on designing software for data integration as well as system functionality support. The last chapter describes how Multidetector was tested and if it can be used on real conditions to provide real data of vehicles, traffic and associated parameters.

It is important to mention again, that design of system is general, independent on platform (Raspberry Pi) on which it was build and tested in this work. It can be built on any other discussed platforms (self-designed computer, Arduino or Tecomat Foxtrot), or other chosen piece of technology which meet requirements listed in chapter Definition of Multidetector.

Even chosen programming language is not compulsory. The important are algorithms described in chapter Algorithms of Data Fusion. Code written for this particular version of Multidetector, which use Python programming language, is mirroring described algorithms and can be used as model to inspire code written in other programming language or modified in next versions of Multidetector.

There was selected a set of sensors and detectors to represent each interface. For these, there were programmed processes to control them and collect data. These processes can again serve as examples of how to design a new process if there is need to add a new device into system.

There was set a software environment of Multidetector. With this, it is now possible to run operating system and other processes and services which ensure running and managing Multidetector (SSH, Wi-Fi, Python 3, etc.). Additionally, the whole operating system with all settings and software required for proper operating is included in electronic attachments if this work on CD. Therefore, it is extremely easy to copy it to SD card, mount it into Raspberry computer and boot and use Multidetector.

14.2 Testing of the Multidetector and What It Can and Can’t Do

Individual processes were regularly tested during development. At the end, the whole system was tested in laboratory conditions and then in real situation on traffic. From the laboratory testing came out, that all processes were working correctly, communication channels were passing information without time delay or modifying data and the main program was collecting data from individual detector processes and saving them in correct format with correct time stamp.

Real condition tests were very similar to the laboratory testing as far as Multidetector settings. There was used different set of sensors with corresponding detector processes. The Multidetector ran measurement for a longer period and collected more data. This testing proved, that Multidetector is capable of collecting data, and these data can be processed into meaningful traffic parameters.

The Multidetector was primary meant to help with traffic researches as a mobile measurement unit. This was also tested and proved to be working. Other branches could be parking slots detector (ultrasonic), input nodes of permanent telematic systems, long term researches systems, tunnel systems, etc. Above mentioned are using similar sensors and detectors but use of Multidetector for similar systems should be tested before implementation.

In current version of Multidetector there is missing a proper graphical user interface. The idea was to create web server which would serve a page to any device connected in Wi-Fi already build in Multidetector. The process of developing web server and all pages with content, style and the JavaScript code already started, but it showed up to be a much bigger topic then thought at the beginning of this work. For better use of Multidetector, GUI will be done after this work in separate paper as an extension of current Multidetector version.

14.3 Plans for the Future

As tests showed, it is complicated to process data manually, so there is a big urge to make a special software for data evaluation. This software doesn’t necessarily need to be a part of Multidetector, it can be a separate program for PC which will be designed especially for data format which Multidetector save into files.

As a possible use of Multidetector, there was mentioned telematic systems. These systems usually consist of detectors, actors and some processing or decision-making unit (in its basic configuration). Multidetector could act as an input node into such a system (as a detector). But most of these systems are processing data instantly and the latency of delivering data is crucial. Therefore, next logical step would be to make a version of Multidetector where data are not saved into files, but when the main process collect data from detector processes, it would send these data straight to communication channel connecting Multidetector with telematic system.

There is also possibility to create telematic system based on Multidetectors. It was showed, that Multidetector can identify vehicles with the right set of detectors. Therefore, it would be possible to set up directional survey. This system would require a net of Multidetectors communicating with each other and an additional evaluation software. To achieve it, there would be need to solve above mentioned topic of online output data streams and communication between individual Multidetectors.

Another space for Multidetector to get better in future is in casing. Most of the traffic detectors accessible on market has a good protection against weather conditions which could harm detectors. Good casing and mounting system could as well prevent Multidetector against vandalism or stealing.

The whole system will be further tested and used for measurements to find out bugs and week parts which can be improved to make Multidetector more robust and easy to use.

A List of Sources

[1] Li, K., Li, W., Chen, Z., Liu, Y. *Computational Intelligence and Intelligent Systems: 9th International Symposium*. Guangzhou, China : Springer Singapore, 2017. ISBN 978-981-13-1651-7

[2] Langr, M.: *Fúze heterogenních dopravních dat pro odhadování směrových vztahů*. Dissertation. Praha : ČVUT, 2015

[3] Posekaný, L.: *Processing and Evaluation of Traffic Data from Pico Iteris Videodetection System*. Bachelor thesis. Praha : ČVUT, 2016

[4] Dictionary.com. *Sensor, Meanings and Definitions of Words at Dictionary.com.* [online]. Copyright © Random House, Inc. 2018 [cit. 23.11.2018]. Retrieved from: https://www.dictionary.com/browse/sensor

[5] Cambridge Dictionary | English Dictionary, Translations & Thesaurus. *SENSOR, meaning in the Cambridge English Dictionary* [online]. Copyright © Cambridge University Press [cit. 23.11.2018]. Retrieved from: https://dictionary.cambridge.org/dictionary/english/sensor

[6] The National Academies Press. *Introduction to Sensors* [online]. Retrieved from: https://www.nap.edu/read/4782/chapter/4#10

[7] The Free Dictionary. *Detector, Definition of Detector by The Free Dictionary* [online]. Copyright © 2003 [cit. 23.11.2018]. Retrieved from: https://www.thefreedictionary.com/detector

[8] LAWRENCE A. KLEIN. *Traffic detector handbook. 3rd ed*. United States: Federal Highway Administration, Turner-Fairbank Highway Research Center, 2006, 2006.

[9] Vaisala [datasheet]. *Vaisala Nu-Metrics Portable Traffic Analyzer NC200*. B211017EN-A ©Vaisala 2010 [cit. 23.11.2018]. Retrieved from: http://www.coralsales.com/attachments/85/NC200_Vaisala_Portable_Traffic_Analyzer_Datasheet.pdf

[10] IEEE. *IEEE Std 802.3™-2012*. New York: NY 10016-5997 USA, 2012.

[11] IEEE. *IEEE Std 802.11n™-2009*. New York: NY 10016-5997 USA, 2009.

[12] Bluetooth. *Bluetoooth SIG* [online]. Copyright © 2018 Bluetooth SIG, Inc. All rights reserved. [cit. 23.11.2018]. Retrieved from: https://www.bluetooth.com/

[13] Techopedia Inc. *Definition from Techopedia - Universal Serial Bus (USB)* [online]. Copyright © 2018 Techopedia Inc. [cit. 23.11.2018]. Retrieved from: https://www.techopedia.com/definition/2320/universal-serial-bus-usb

[14] Voltcraft [datasheet]. *Digital Noise Meter with Data Log SL-451*. © Copyright Conrad Electronic Česká Republika, s. r. o, 2011

[15] The Tech Terms. *Data Definition* [online]. Copyright © 2018 Sharpened Productions [cit. 23.11.2018]. Retrieved from: https://techterms.com/definition/data

[16] Posekaný, L. Růžička, M.: *Ubertooth as a Traffic Detector*. Semestral work [CD], Praha : ČVUT, 2017. Retrieved from: elektronické přílohy soubor Ubertooth_as_a_Traffic_Detector.pdf

[17] TechnologyUK - Home Page. *Bluetooth* [online]. Copyright © 2001 [cit. 24.11.2018]. Retrieved from: http://www.technologyuk.net/telecommunications/communication-technologies/bluetooth.shtml

[18] Project Ubertooth - Home. *Project Ubertooth* [online]. [cit. 23.11.2018]. Retrieved from: http://ubertooth.sourceforge.net/

[19] GitHub. *Great Scott Gadgets: Ubertooth One*. [online]. Copyright © 2018 [cit. 23.11.2018]. Retrieved from: https://github.com/greatscottgadgets/ubertooth/wiki/Ubertooth-One

[20] QQ Online Trading. *IR Infrared 2 - 30cm Obstacle Detaction Sensor Module FC-51* [online]. Retrieved from: http://qqtrading.com.my/ir-infrared-obstacle-detaction-sensor-module-fc-5

[21] India Exporter Manufacturer. *Ultrasonic Sensor at Rs 700 /piece* [online]. Copyright © 1996 [cit. 23.11.2018]. Retrieved from: https://www.indiamart.com/proddetail/ultrasonic-sensor-18436665012.html

[22] SearchEnterpriseWAN. *What is a sensor network?. Wide Area Network (WAN) information, news and tips* [online]. [cit. 23.11.2018]. Retrieved from: https://searchenterprisewan.techtarget.com/What-is-a-sensor-network

[23] Veracity – Innovative Solutions for IP Video Surveillance. *Power over Ethernet (POE) Explained*. [online]. [cit. 23.11.2018]. Retrieved from: http://www.veracityglobal.com/resources/articles-and-white-papers/poe-explained-part-2.aspx

[24] Techopedia - Where Information Technology and Business Meet. *What is a Central Processing Unit (CPU)? - Definition from Techopedia*. [online]. Copyright © 2018 Techopedia Inc. [cit. 25.11.2018]. Retrieved from: https://www.techopedia.com/definition/2851/central-processing-unit-cpu

[25] Arduino-shop.cz. *Obchod s Arduinem* [online]. Copyright © ECLIPSERA s.r.o. [cit. 23.11.2018]. Retrieved from: https://arduino-shop.cz/

[26] TECO - Automation – Home. *PLC Tecomat Foxtrot* [online]. Copyright © 2017 Teco a. s. [cit. 24.11.2018]. Retrieved from: https://www.tecomat.com/products/cat/cz/plc-tecomat-foxtrot-3/

[27] TECO - Automatizace – Home. *CP-1000* [online]. [cit. 24.11.2018]. Retrieved from: https://www.tecomat.cz/uploads/images/produkty/foxtrot/Foxtrot0.png

[28] PLC programming - TECO - Automation. *MOSAIC*. [online]. Copyright © 2017 Teco a. s. [cit. 24.11.2018]. Retrieved from: https://www.tecomat.com/support/plc-programming/

[29] Tecomat [product catalogue]. *Tecomat Foxtrot/CFox/RFox Product Catalog*. Retrieved from: https://www.tecomat.cz/modules/DownloadManager/download.php?alias=catalogue_foxtrot_2014

[30] Raspberry Pi [datasheet]. *Raspberry Pi 3 Model B+*, [cit. 23.11.2018]. Retrieved from: https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-Model-Bplus-Product-Brief.pdf

[31] Raspberry Pi — Teach, Learn, and Make with Raspberry Pi. *Raspberry Pi 3 Model B+ - Raspberry Pi* [online]. Retrieved from: https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/

[32] Raspberry Pi - Teach, Learn, and Make with Raspberry Pi. *Schematic1 : Page2 – Power, XOS* [online]. Copyright © [cit. 23.11.2018]. Retrieved from: https://www.raspberrypi.org/documentation/hardware/raspberrypi/schematics/rpi_SCH_3bplus_1p0_reduced.pdf

[33] Raspberry Pi — Teach, Learn, and Make with Raspberry Pi. *Raspberry Pi - Teach, Learn, and Make with Raspberry Pi* [online]. Retrieved from: https://www.raspberrypi.org

[34] SSH (Secure Shell) Home Page. *SSH Communications Security* [online]. Copyright ©2018 SSH Communications Security, Inc. All Rights Reserved. [cit. 23.11.2018]. Retrieved from: https://www.ssh.com/ssh/

[35] Raspberry Pi NAT [datasheet]. *Setting up a Raspberry Pi as an access point in a standalone network (NAT)*. [cit. 23.11.2018]. Retrieved from: https://www.raspberrypi.org/documentation/configuration/wireless/access-point.md

[36] Python.org. *What is Python? Executive Summary* [online]. Copyright ©2001 [cit. 23.11.2018]. Retrieved from: https://www.python.org/doc/essays/blurb/

[37] Python Programming Language – Legacy Website. *Python 2.7 Release Schedule*. [online]. Copyright © 1990 [cit. 25.11.2018]. Retrieved from: https://legacy.python.org/dev/peps/pep-0373/

[38] Python 3.7.1 documentation [datasheet]. *Concurrent Execution*. [cit. 25.11.2018]. Retrieved from: https://docs.python.org/3/library/concurrency.html

[39] Python 3.7.1 documentation [datasheet]. *Threading — Thread-based parallelism*. [cit. 25.11.2018]. Retrieved from: https://docs.python.org/3/library/threading.html

[40] Python 3.7.1 documentation [datasheet]. *Multiprocessing — Process-based parallelism*. [cit. 25.11.2018]. Retrieved from: https://docs.python.org/3/library/multiprocessing.html

[41] Techopedia - Where Information Technology and Business Meet. *What is a Race Condition? - Definition from Techopedia*. [online]. Copyright © 2018 Techopedia Inc. [cit. 25.11.2018]. Retrieved from: https://www.techopedia.com/definition/10313/race-condition

[42] Python Software Foundation Wiki Server. *GlobalInterpreterLock*. [online] [cit. 23.11.2018]. Retrieved from: https://wiki.python.org/moin/GlobalInterpreterLock

[43] ELEC Freaks [datasheet]. *Ultrasonic Ranging Module HC - SR04*. Copyright © 2018 by AspenCore, Inc. All Rights Reserved. [cit. 23.11.2018]. Retrieved from: https://www.electroschematics.com/wp-content/uploads/2013/07/HCSR04-datasheet-version-1.pdf

[44] ADCDAC Pi Zero [datasheet]. *2 Channel ADC and 2 Channel DAC for the Raspberry Pi*. Copyright © AB Electronics UK 2012 – 2018. [cit. 23.11.2018]. Retrieved from: https://www.abelectronics.co.uk/kb/article/1053/adc-dac-pi

[45] ADC DAC Pi Zero [Schematics]. *Schematics of ADCDAC Pi Zero*. [cit. 23.11.2018]. Retrieved from: [https://www.abelectronics.co.uk/docs/pdf/schematic-adc-dac-pi.pdf]

[46] AB Electronics UK | Raspberry Pi expansion boards and accessories. *ADCDAC Pi with Python on a Raspberry Pi* [online]. [cit. 25.11.2018]. Retrieved from: https://www.abelectronics.co.uk/kb/article/15/adcdac-pi-with-python

[47] AB Electronics UK | Raspberry Pi expansion boards and accessories. *SPI and Raspbian wheezy*. [online]. [cit. 25.11.2018]. Retrieved from: https://www.abelectronics.co.uk/kb/article/2/spi-and-raspbian-linux-on-a-raspberry-pi

[48] AB Electronics UK | Raspberry Pi expansion boards and accessories. *Python Library and Demos for the Raspberry Pi*. [online]. [cit. 25.11.2018]. Retrieved from: https://www.abelectronics.co.uk/kb/article/23/python-library-and-demos

A List of Figures

A List of Tables

A List of Attachments

Attachment 1.: The Main Program “multidetector.py”

```
!/usr/bin/python3

import multiprocessing
import RPi.GPIO as GPIO
import time
import datetime
from configparser import SafeConfigParser

# detectors functions and processes
import detectors
import filesManipulation

# check if program is runned by sudo user
userid = os.getuid()
if username != 0:
    print("this program MUST be run from root user.")
    exit()

# initialize start time and config file
initTime = datetime.datetime.now().strftime("%H:%M:%S.%f")
print("init time: ", initTime)
parserConf = SafeConfigParser()
parserConf.read('multidetector.conf')
parserConf.set('device', 'startOfMeasurement', str(initTime))
parserConf.set('device', 'fileNumber', str(1))
with open('multidetector.conf', 'w') as configfile:
    parserConf.write(configfile)

# init GPIO ports
GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)
print("initialization of RPi GPIO ports, numbering BOARD")

# load configuration file and READ detectors
mainLoopInterval = float(parserConf.get('detectors', 'mainLoopInterval'))
detConf = parserConf.get('detectors', 'activeDetectors')
detConf = detConf.split(" ")

# loop active detectors listed in multidetector.conf file,
# load names, arguments and create processes
# save all into array of dictionaries:
#   name: "custom name of detector"
#   processName: "name of function which will be runned as process"
#   processVariable: "created process stored into variable so it can be used"
activeDetectors = []
for detector in detConf:
    values = parserConf.get('detectors', detector)
    values = values.split(" ")
    # sharedVariable is referrence to a shared variable of type ctypes.c_char
    sharedVariable = multiprocessing.Array('c', 20)
    arguments = values[1:]
    arguments.append(sharedVariable)
    det = {
        "name": detector,
        "processName": values[0],
        "processArguments": arguments,
        "processVariable":
  multiprocessing.Process(target=detectors.knownProcesses[values[0]], args=arguments),
        "sharedVariable": sharedVariable
    }
    activeDetectors.append(det)

filePath = filesManipulation.newFile()
tableHeader = "time\t"
recordsInMemory = 0
arrayInMemory = []

# print active detectors as loaded and saved into dictionary
print("Detectors loaded and prepared for measurement: ")
for det in activeDetectors:
    print("\tname: ", det["name"], "\t runned as: ", det["processName"], "\twith
  arguments: ", det["processArguments"][:len(det["processArguments"])-1])
    tableHeader += str(det["name"]) + "\t"
    det["processVariable"].start()
tableHeader += "\n"
print(tableHeader)

with open(filePath, 'a') as f:
    f.write(tableHeader)

time.sleep(1)

# initialize list of strings as they will be saved into file
# in each iteration I'll check if values changed, if at last one did, I'll write it
  into file
writtenValues = [""]*len(activeDetectors)

try:
    # main loop checking running detector processes and getting data from them
    while True:
        someValueChanged = False
        for i in range(0,len(activeDetectors)):
            if writtenValues[i] !=
  activeDetectors[i]["sharedVariable"].value.decode("utf-8"):
                writtenValues[i] =
  activeDetectors[i]["sharedVariable"].value.decode("utf-8")
                someValueChanged = True
        if someValueChanged:
            intoMemory = [datetime.datetime.now().strftime("%H:%M:%S.%f")] +
  writtenValues
            arrayInMemory.append(intoMemory)
            print(intoMemory)
            recordsInMemory += 1
            if recordsInMemory > 10:
                filesManipulation.writeValues(filePath, arrayInMemory)
                arrayInMemory = []
                recordsInMemory = 0
        time.sleep(mainLoopInterval)

except KeyboardInterrupt:
    print("\nstopped by user")
    for detector in activeDetectors:
        detector["processVariable"].terminate()
        print(detector["name"], "is now down")
```

Attachment 2.: Configuration File “multidetector.conf”

```
[device]
ID = 1
measurementFilesPath = ./testResults/
startofmeasurement = 20:09:43.747148
nameofoperator = Lukas
filenumber = 2
filesgroup = testMeasurement

[RPi]
# allowedgpio = 3,5,7,8,10,11,12,13,15,16,18,19,21,22,23,24,26
# allowedgpio with adcdacpi = 3,5,7,8,10,11,12,13,15,16,18

[detectors]
# interval in which the Multidetector will read detectors values
mainloopinterval = 0.05

# detectors user want to activate (named by user)
activedetectors = bluetooth sound switch video1 video2

# ultrasonicProcess(trigger, echo, null threshold, interval)
ultrasonic = ultrasonicProcess 8 10 30 1

# switchProcess(pin, interval)
switch = switchProcess 11 0.1
video1 = switchProcess 12 0.1
video2 = switchProcess 13 0.1

# soundProcess(null threshold, interval)
sound = soundProcess 55 1

# bluetoothProcess()
bluetooth = bluetoothProcess
```

Attachment 3.: Library “detectors”

```
import RPi.GPIO as GPIO
from ADCDACPi import ADCDACPi
import time
import subprocess
import re
import threading

GPIO_Allowed = [3,5,7,8,10,11,12,13,15,16,18,19,21,22,23,24,26]

# (int distance) UltrasonicDistance(GPIO_BOARD_TRIGGER, GPIO_BOARD_ECHO)
# is function which sends pulse on trigger of HC-SR04 and listen for echo
# it calculate how long it take to sound to travel to obstacel and back and from this
  time calculate distance from obstacle
# returns calculated distance
def ultrasonicDistance(GPIO_BOARD_TRIGGER, GPIO_BOARD_ECHO):
    # send a signal 0.00001s long
    GPIO.output(GPIO_BOARD_TRIGGER, True)
    time.sleep(0.00001)
    GPIO.output(GPIO_BOARD_TRIGGER, False)

    StartTime = time.time()
    StopTime = time.time()

    # wait untill echo will return
    while GPIO.input(GPIO_BOARD_ECHO) == 0:
        StartTime = time.time()
    while GPIO.input(GPIO_BOARD_ECHO) == 1:
        StopTime = time.time()

    # calculate distance
    TimeElapsed = StopTime - StartTime
    distance = (TimeElapsed * 34300) / 2
    return distance

# ultrasonicProcess(GPIO_BOARD_TRIGGER, GPIO_BOARD_ECHO, null_distance, interval,
  sharedVariable)
# is a function which initialize ultrasonic detector
# in infinite loop checks in given interval a distance with help of
  UltrasonicDistance() function
# in each iteration set shared variable (distance) to a new value
# if distance is greater than given limit, it returns 0
def ultrasonicProcess(GPIO_BOARD_TRIGGER, GPIO_BOARD_ECHO, null_distance, interval,
  sharedVariable):

    # convert pins to int and check if they are allowed to use
    GPIO_BOARD_TRIGGER = int(GPIO_BOARD_TRIGGER)
    if GPIO_BOARD_TRIGGER not in GPIO_Allowed:
        print(GPIO_BOARD_TRIGGER, "is not in allowed list of pins")
    GPIO_BOARD_ECHO = int(GPIO_BOARD_ECHO)
    if GPIO_BOARD_ECHO not in GPIO_Allowed:
        print(GPIO_BOARD_ECHO, "is not in allowed list of pins")

    null_distance = int(null_distance)
    interval = int(interval)

    GPIO.setup(GPIO_BOARD_TRIGGER, GPIO.OUT)
    GPIO.setup(GPIO_BOARD_ECHO, GPIO.IN)
    while True:
        # get distance
        distance = int(ultrasonicDistance(GPIO_BOARD_TRIGGER, GPIO_BOARD_ECHO))
        # check for null value
        if distance < null_distance:
            distance = "null"
        # set shared variable to a new value
        sharedVariable.value = str(distance).encode("utf-8")
        time.sleep(interval)

# switchProcess(GPIO_BOARD_SWITCH, interval, sharedVariable)
# is a function for a checking of a state on given pin number
# periodically (in given interval in seconds) checks the state on pin
# every iteration sets a shared variable by state on pin (0/1)
# can be used for button, switch, videodetection, ...
def switchProcess(GPIO_BOARD_SWITCH, interval, sharedVariable):
    # convert pins to int and check if they are allowed to use
    GPIO_BOARD_SWITCH = int(GPIO_BOARD_SWITCH)
    if GPIO_BOARD_SWITCH not in GPIO_Allowed:
        print(GPIO_BOARD_SWITCH, "is not in allowed list of pins")

    interval = float(interval)

    # SWITCH TEST
    GPIO.setup(GPIO_BOARD_SWITCH, GPIO.IN)
    while True:
        state = GPIO.input(GPIO_BOARD_SWITCH)
        if state == 0:
            state = 1
        else:
            state = 0
        sharedVariable.value = str(state).encode("utf-8")
        time.sleep(interval)

# soundProcess(null_dB, interval, sharedVariable)
# is a function which runs an infinite loop (repeating in given interval)
# and in each iteration read a voltage level on analogue input pin of ADC-DAC Pi Zero
# calculate value of sound in dB and set sharedVariable to this new value
# if sound level is below given null_dB value, it set sharedVariable to null
def soundProcess(null_dB, interval, sharedVariable):
    adc1 = ADCDACPi(1)
    adc1.set_adc_refvoltage(3.3)

    while True:
        v1 = adc1.read_adc_voltage(1, 0)*100
        if v1>int(null_dB):
            sharedVariable.value = str(int(v1)).encode("utf-8")
        time.sleep(int(interval))

# bluetoothProcess(sharedVariable)
# is a function which initialize Ubertooth One
# it start a new threat ubertoothThread(arrayOfLAPs) which is running scans and
 reading LAP parts of addresses from Ubertooth
# LAPs are added into array (FIFO - First In First Out) = queue
# bluetoothProcess() is then getting LAPs one by one from queue and servs them
 through sharedVariable
def bluetoothProcess(sharedVariable):
    # empty "queue"
    arrayOfLAPs = []
    # thread ubertoothThread(arrayOfLAPs)
    ubertooth = threading.Thread(target=ubertoothThread, args=(arrayOfLAPs,))
    ubertooth.start()
    # loop which reads LAPs from thread and putting them into sharedVariable in given
  interval
    while True:
        if arrayOfLAPs:
            sharedVariable.value = arrayOfLAPs.pop(0)   # already encoded
        else:
            sharedVariable.value = "null".encode("utf-8")
        time.sleep(0.1)
    ubertooth.join()

# ubertoothThread(arrayOfLAPs) is function serving as thread
def ubertoothThread(arrayOfLAPs):
    # create process of Ubertooth
    process = subprocess.Popen(['ubertooth-rx', '-z'], stdout=subprocess.PIPE,
  bufsize=1)
    alreadyLoadedLAPs = []
    # loop lines with catched devices
    for line in iter(process.stdout.readline, b''):
        if len(line) < 50:
            continue
        device = bluetoothProcessLine(line)

        if isWaiting(device, alreadyLoadedLAPs):
            continue

        if device:
            arrayOfLAPs.append(str(device[1]).encode("utf-8"))

        else:
            print("empty line")

# isWaiting will check if LAP was seen in last "waitTime" seconds
# if seen, it return True if was seen
def isWaiting(device, alreadyLoadedLAPs):
    isWaiting = False
    waitTime = 10

    if alreadyLoadedLAPs:  # if array is not empty
        # remove from array all expired elements
        for el in alreadyLoadedLAPs:
            if el[0] + waitTime <= device[0]:
                alreadyLoadedLAPs.remove(el)
                continue

        # check if is contained in alreaadyLoadedLAPs
        for el in alreadyLoadedLAPs:
            if device[1] == el[1]:
                isWaiting = True

    # if not in list, write it down
    if not isWaiting:
        alreadyLoadedLAPs.append([device[0], device[1]])

    return isWaiting

# will take a line and process it
# extract systime and LAP part of address
def bluetoothProcessLine(line):
    returnArray = []
    line = line.decode(encoding="utf-8")
    line = line.split(' ')
    for col in line:
        if re.match(r"^systime", col):
            systime = col
            systime = int(systime.split('=')[1])
        elif re.match(r"^LAP", col):
            LAP = col
            LAP = LAP.split('=')[1]
            # returnArray.append(LAP)
    if LAP and systime:
        returnArray = [systime, LAP]
    return returnArray

# define known processes (by its process names)
# this dictionary is here so program can call functions by string (its name)
knownProcesses = {
    'ultrasonicProcess': ultrasonicProcess,
    'switchProcess': switchProcess,
    'soundProcess': soundProcess,
    'bluetoothProcess': bluetoothProcess
}
```

Attachment 4.: Library “filesManipulation”

```
import time
import datetime
import os
from configparser import SafeConfigParser

i = 1

def formHeader():
    header = ""
    parserConf = SafeConfigParser()
    parserConf.read('multidetector.conf')
    header += "Multidetector ID: " + parserConf.get('device', 'ID') + "\n"
    header += "Start of measurement: " + parserConf.get('device',
  'startOfMeasurement') + "\n"
    header += "Time of creating file: " +
  datetime.datetime.now().strftime("%H:%M:%S.%f") + "\n"
    header += "File number: " + parserConf.get('device', 'fileNumber') + "\n"
    header += "Name of operator: " + parserConf.get('device', 'nameOfOperator') +
  "\n\n"

    # change number of file for next one
    parserConf.set('device', 'fileNumber', str(int(parserConf.get('device',
  'fileNumber'))+1))
    with open('multidetector.conf', 'w') as configfile:
        parserConf.write(configfile)

    return header

def formFooter():

    return footer

# find free smallest fileName and return this name
def newFile():
    parserConf = SafeConfigParser()
    parserConf.read('multidetector.conf')
    dirPath = parserConf.get('device', 'measurementFilesPath')
    filesGroup = parserConf.get('device', 'filesGroup')
    biggestInDir = 0
    smallestFreeInDir = 0
    NewFilePath = ''

    # check if dir path ends with "/"
    if not dirPath.endswith('/'):
        dirPath = dirPath + "/"

    # if path doesn't exists, create it
    if not os.path.exists(dirPath):
        os.makedirs(dirPath)

    # loop through files stareting with same filesGroup
    for file in os.listdir(dirPath):
        if file.endswith('.txt') and file.startswith(filesGroup):
            fileNumb = file[len(filesGroup):-4]
            try:    # check if string represent int number
                int(fileNumb)
                if biggestInDir < int(fileNumb):
                    biggestInDir = int(fileNumb)
            except ValueError:
```

```
            print("Problem with filename")

    smallestFreeInDir = biggestInDir + 1
    NewFilePath = dirPath + filesGroup + str(smallestFreeInDir) + '.txt'

    with open(NewFilePath, 'w') as f:
        f.write(formHeader())

    return NewFilePath

def writeValues(filePath, arrayToSave):
    data = ""
    for line in arrayToSave:
        for item in line:
            data += str(item) + "\t"
        data += "\n"

    with open(filePath, 'a') as f:
        f.write(data)

    print(data)



def closeFile(filePath):
    with open(filePath, 'a') as f:
        f.write(data)
    return True
```

Attachment 5.: Laboratory Measurement File “testMeasurement1.txt”

```
Multidetector ID: 1
Start of measurement: 10:15:34.442516
Time of creating file: 10:15:34.540764
File number: 1
Name of operator: Lukas

time            switch IR     ultrasonic
10:15:35.554065 0      0      null
10:15:37.661432 0      1      null
10:15:37.762226 0      0      null
10:15:39.367740 0      1      null
10:15:39.568818 0      0      null

                .
                .
                .

[whole data available on CD attachment, file “testMesurement1.txt”]

                .
                .
                .

10:16:58.797947 0      0      null
10:17:00.503718 0      0      22
10:17:03.514168 0      0      23
10:17:06.724644 0      0      null
10:17:09.734729 0      0      26

Number of records:  55
Time of file end: 10:17:09
```

Attachment 6.: On Field Measurement File “testMeasurement2.txt”

```
Multidetector ID: 1
Start of measurement: 9:09:43.747148
Time of creating file: 9:09:43.845814
File number: 1
Name of operator: Lukas

time            bluetooth sound switch video1 video2
9:09:44.865667  null   68     0      1      0
9:09:45.869345  null   75     0      1      0
9:09:46.321232  null   75     0      1      1
9:09:46.672615  null   75     1      1      1
9:09:46.823590  null   75     1      1      0
9:09:46.874075  null   72     0      1      0
9:09:47.175362  null   72     0      0      0

                       .
                       .
                       .

[whole data available on CD attachment, file “testMesurement2.txt”]

                       .
                       .
                       .

9:24:57.502060  null   61     0      0      0
9:24:57.753541  null   61     0      1      0
9:24:58.506993  null   59     0      1      0
9:24:58.557863  null   59     0      0      0
9:24:59.512319  null   57     0      0      0
9:25:00.516727  null   58     0      0      0
9:25:01.520359  null   68     0      0      0

Number of records:  1683
Time of file end: 9:25:01
```