AGH UNIVERSITY OF SCIENCE AND TECHNOLOGY IN KRAKOW

FACULTY OF ELECTRICAL ENGINEERING, AUTOMATICS, COMPUTER SCIENCE AND BIOMEDICAL ENGINEERING

DEPARTMENT OF AUTOMATION AND ROBOTICS

Master of Science Thesis

Vehicle detection using machine learning algorithms
Detekcja pojazdów z użyciem algorytmów uczenia maszynowego

Author: *Adrian Moskwik*
Degree programme: *Automation and Robotics*
Supervisor: *dr inż. Łukasz Więckowski*

Krakow, 2021

Serdecznie dziękuję mojemu promotorowi dr inż. Łukaszowi Więckowskiemu za poświęcony czas i pomoc w realizacji tego projektu. Dziękuje także mojej siostrze Żanecie i kuzynowi Dawidowi, za pomoc w zbieraniu danych treninigowych, oraz dziewczynie Ewelinie za pomoc w weryfikacji systemu i wsparcie mentalne. Chciałbym także podziękować rodzicom, Renacie i Pawłowi, za ciągłą pomoc na przestrzeci całego toku studiów, która pozwoliła mi je rozpocząć i ukończyć.

Contents

1. Introduction

The fourth industrial revolution has driven modern technology towards automation made possible by the constant exchange and processing of data. The key to such a solution is communication, which allows large amounts of information to be transmitted. This need has resulted in the Internet of Things, which allows data to be exchanged between all devices. Such a solution can be extremely useful in everyday life. Imagine a situation in which, we have the Internet of Things implemented in our home. A smart fridge that orders food by itself when it runs out, a smart thermostat that sets the temperature of a room depending on the weather, smart blinds that close when it's nighttime. Additionally, all these solutions can be easily controlled from a mobile device.

Of course, the Internet of Things is not limited to controlling the house. It can also be applied on a much wider scale. A smart city could be such a solution. Smart grids, smart street lights, or smart transport are just some of the solutions already in use today. Another very popular solution is smart parking, which allows us to locate a free parking space before we reach our destination. This has given rise to a need for reliable verification of the availability of such a spot. My work will focus on the problem of vehicle detection in a parking space.

1.1. The aim of the study

The main objective of this thesis is to develop efficient methods for the detection of vehicles parking and moving within isolated parking spaces. Correct detection and classification of parking events are to be based on magnetic field analysis, using one or a group of sensors prepared for this purpose.

To extract specific events, it is planned to use classification methods based on machine learning. It is required to create training and test datasets, guaranteeing proper identification of events by a trained neural network. Additional goals for the prepared system:

- *Detection accuracy independent of vehicle type.*
- *Elimination of the influence of interference (magnetic field variations) from vehicles parked in adjacent parking spaces.*
- *Automatic calibration of the sensors.*

– *A web or mobile application with a user GUI that allows for demonstration of operation and management of devices working in a group.*

The designed sensory system for detecting the presence of a vehicle will be subjected to tests to verify its effectiveness.

1.2. Layout of the thesis

The thesis consists of 6 chapters. The first chapter was devoted to an introduction to Smart Parking systems, and the need for correct vehicle detection in parking spaces. In the second chapter, there is a theoretical introduction of the Internet of Things system, as well as a review of solutions used so far to detect vehicles. The third chapter was presented the hardware part of created system, information about used sensors, test measurements and encountered problems. The fourth chapter was devoted to the review of machine learning algorithms and their verification using the collected data. Chapter five describes the communication of the system with the local server and the final tests of the created system. Chapter six contains a summary of the thesis workflow.

2. Theoretical overview

2.1. Internet of things

In the past, only computers could be linked together by a single network, but in modern times this is changing. More and more devices of all kinds are being connected to coexist. Phones, home devices, cars are exchanging increasing amounts of information with each other. This is referred to as the Internet of Things (Fig.2.1), a popular definition of which is: Internet of things is a network of physical objects [1].

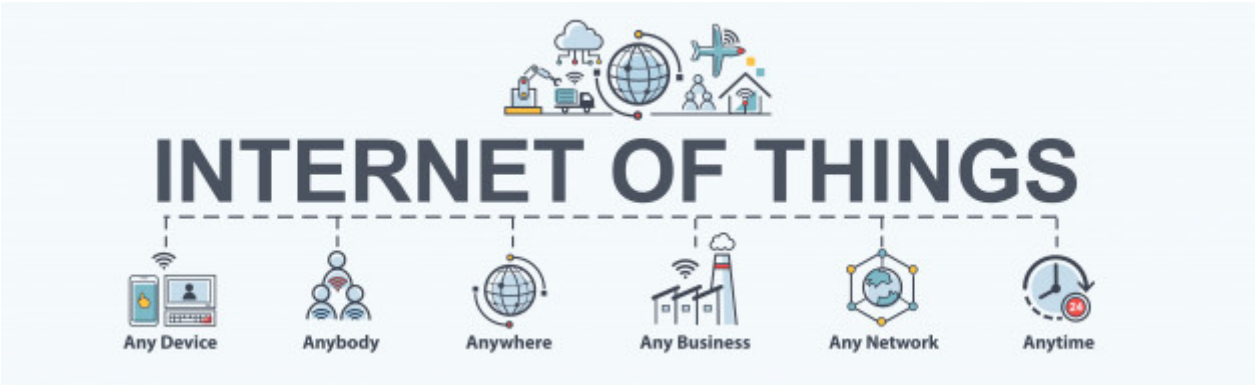

Fig. 2.1. Internet of things [2].

IoT is not a single technology, but more of a concept describing the interaction of multiple technologies that ultimately achieve communication between things. The technologies used can vary, but are most commonly divided into four layers that together make up the Internet of Things [3]:

- ***Physical/Device Layer*** *- it consists of end devices. Sensors to collect the necessary data and actuators to perform an action based on the collected data. For example, this layer may include a thermometer and a fan, that starts depending on the room temperature.*
- ***Network and Data Acquisition Layer*** *- is responsible for the network used to transmit data from the sensors and for combining data from several sources into a coherent whole.*
- ***Edge/Transport Layer*** *- is responsible for device and information management. It is also responsible for data filtering, prepossessing of data, access control, data analysis, etc.*

– ***Application Layer*** *- This is the layer closest to the end-user. It consists of user interfaces at the very top of the architecture. The variety of operation of these applications is very large, they can interfere in technology branches such as transportation, healthcare, smart home, industry, smart city and much more.*

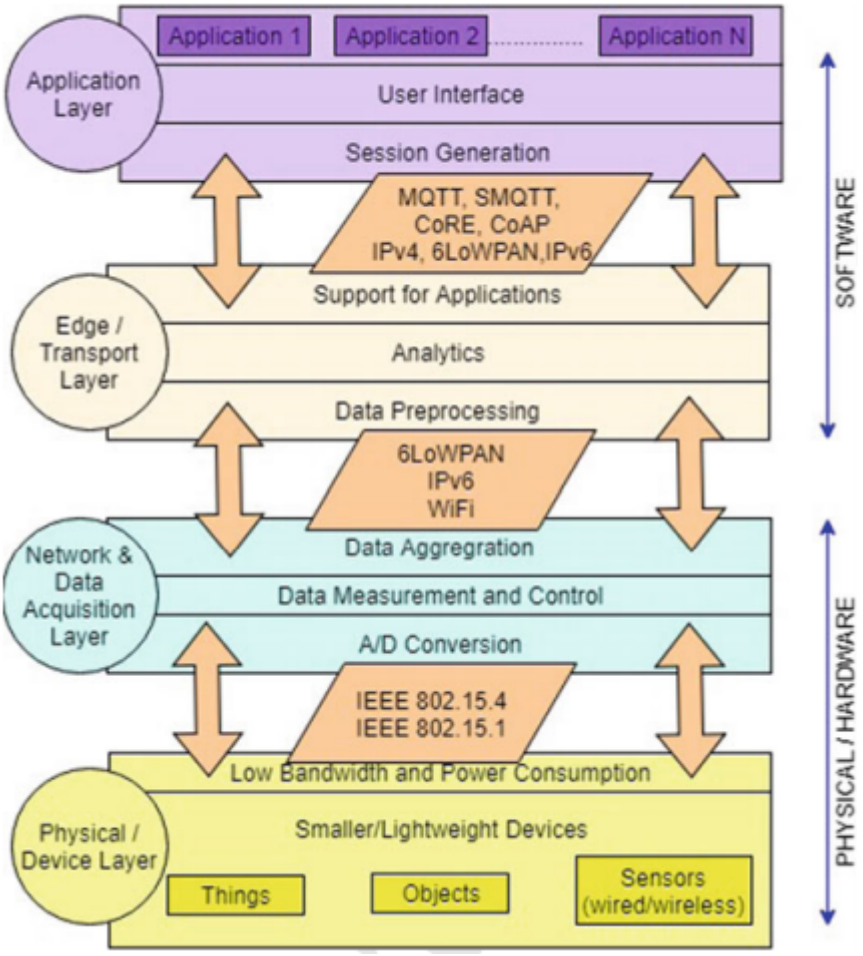

Fig. 2.2. The architecture of IoT [3].

Subsystems working together in this way create IoT solutions used to make life easier and better for the average person. They allow us to forget about the small things we have to do ourselves every day and aim to help us do as much as possible. These solutions interfere in many aspects of life, and one of the most flagship ones is the smart city, as the solutions included in it have a direct impact on a huge number of people.

2.2. Smart City

There is no single, universally accepted definition of a Smart City (Fig.2.3). It is a relatively young concept that is still being developed and researched. For some, Smart City is mainly defined by the flow

of information and communication of different technologies [4]. Others try to divide these technologies more, to get some definitions of the concept. *Centre of Regiona Science*, Vienna UT presented those indicators as: economy (competitiveness), governance (participation), environment (natural resources), people (social and human capital), mobility (transport and ICT) and living (quality of life).

Fig. 2.3. Smart City concept.

Currently, the term Seven Smart City Component System (7SCCS) is popular, it is dependent on the current development of mega cities, this causes it to change over time. According to 7SCCS, the basis of the smart city infrastructure is the built environment. This approach, unlike the others, marks the importance of the physical systems used in the smart city. It thus creates correlations between the determinants of the smart city and the built environments components used for it. 7SCCS' smart city indicators are [5]:

- *Smart Politics (e.g. transparent governance, public and social services)*
- *Smart Governance (e.g. high productivity, value manufacturing, flexibility of labor market)*
- *Smart People (e.g. active in public life, intelligent labor force, open-mindedness)*
- *Smart Science and Technology (e.g. sustainable and safe transport systems, advanced global ICT infrastructure, green energy resources)*
- *Smart Environment (e.g. environmental protection, sustainable nature management)*

– *Smart Living (e.g. personal safety, health conditions)*

– *Smart Bulding Environment (e.g. urban infrastructure, Hi-Tec construction technology)*

One of the systems to increase the comfort of transportation within the city is Smart Parking.

2.3. Smart parking

Delhi is the second largest city in the world, with a population of 28.5 million [6]. At the end of 2017, the number of registered cars was about 3 million. Between fiscal years 2007 and 2017, the compound annual growth rate of registered motor vehicles stood at 10.1 percent [7]. This shows that the number of cars in large cities is growing significantly, so it is becoming more and more important to have a hassle-free way to get to an available parking space to make everyday life easier. Smart Parking is the system to do just that.

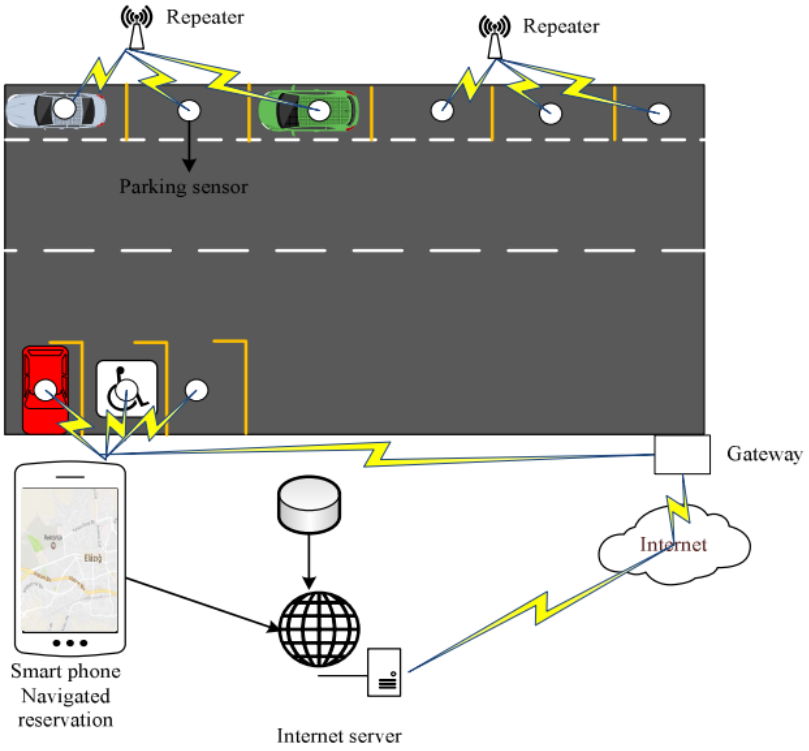

Fig. 2.4. The flow chart of the proposed Smart Parking system.

This system uses our location, usually taken from a mobile device, and then determines the vacant parking space closest to us (which can be reached the fastest). Smart Parking can be divided into two instances [8]. The first one is the hardware. It consists of sensors that send data taken from the parking space to the host system. This system analyzes the readings and decides whether a space is available or not. This information is then sent via the Internet to a database. The second part consists of a program responsible for selecting a parking space based on the analysis of available spaces in the database. One of

the algorithms for these purposes is Pricing Algorithm. It makes a decision based on whether you want to get to the nearest destination, or the nearest available parking space from your current location [9]. System proposed by team from Firat University is shown in Fig.2.4.

Smart Parking is not just a concept for research. The system is already in commercial use. Australian company *Smart Parking Limited*, installs its systems e.g. in shopping centers, supermarkets, airports, commercial parking sites, universities and large scale municipal street environments, in 17 countries around the world.

2.4. Solutions used for vehicle detection

As mentioned in the previous subsection, an important part of Smart Parking is the hardware part, i.e. the modules that deal with analyzing if a parking space is available. In order to do this, two things are needed, a unit that retrieves data from sensors and a host system that runs algorithms to analyze the data.

There are several different sensors suitable for this task, such as a laser sensor, a magnetic sensor or a camera. The data read from these sensors is then analyzed. In this aspect, there is also a large spectrum of methods used, from thresholding to neural networks.

2.4.1. Laser Radar System

The laser radar system is a solution proposed by Tatsuro Yano, Takeshi Tsujimura, Masahiko Mikawa, Hiroyuki Suds in *Vehicle Distinction Using Laser Radar System* [10] in 2001. In those days, the big limitations were the computational time needed to process the measurement points in the case of data from a laser range finder, because the entire environment was scanned, resulting in a large number of measurement points. The use of a CCD camera, on the other hand, was impractical due to the high interference from solar flares.

Japanese scientists' solution consists of a laser distance sensor and pan-tilt arm. The measurement procedure consists of 4 phases in which measurements are taken. Then the collected points are converted into a common coordinate system. The pan-tilt unit allows the laser sensor to be easily re-positioned so that the process can be easily repeated. It continues until the edge point is measured.

Innovative at the time was the use of real-time feedback, which allowed the laser position to be adjusted depending on previous measurements. This significantly reduced the number of measurement points obtained.

The results from the tests obtained were satisfactory. By using feedback, the accuracy of vehicle detection was increased. In addition, it became possible to determine the occupancy of a parking space based on only 10 measurement data.

2.4.2. Road Edge Detection Algorithm on Multi-layer Laser Radar

Over the years, hardware limitations have ceased to be a big problem. Computing power has increased dramatically, making solutions that were once impossible in a parking spot, now available in a moving car. An example of this is using multi-layer laser radar to detect cars in a real-time environment. This is done, for example, by using the Road Edge Detection Algorithm [11].

The primary goal of the algorithm is to divide the road into driveable and non-driveable areas. This allows for a significant reduction in the number of measurement points. The split data can then be classified into clusters using algorithms, based on Euclidean distance. Such analysis allows detection and classification of the surroundings, including cars in the vicinity. The test confirmed that the processing algorithm designed in this way allowed the detection of a moving car as well as moving pedestrians.

Although the use of multi-layered radar in autonomous cars, is becoming more common and eagerly used, it is not the best solution for detecting cars in parking spaces. This is due to several factors. The cost of the laser sensor is quite high and while this is applicable for mapping the environment while driving, using this in a static environment would not be cost-effective. Another downside is that the laser scans the environment at a single elevation level, so it is not possible for one sensor to detect cars in multiple spaces. Another issue is the computing power used, which is quite high when mapping the entire environment, which is not necessary when determining parking space availability.

2.4.3. Car detection based on image processing

One of the most widely used methods for vehicle detection is image processing [12]. It is used both for dynamic detection (moving cars) and static detection (cars standing in a parking space). In dynamic detection, image processing is often only part of the detection system and works in conjunction with other sensors such as lidar, radar or ultra-sensors. This is due to the speed of changes occurring on the road. However, for a parking space, such a solution is often sufficient.

This method uses a camera as a data source, such a solution allows real-time detection. The data read from the camera is first converted from the RGB colour scale to grayscale (Fig. 2.5). This reduces the size of the data and speeds up the algorithm because when using this solution the image is represented as a single matrix with the brightness values of the pixel.

The grayscale image is then the input of the algorithm. For vehicle detection, many algorithms can be used to meet the system requirements and be able to correctly detect a vehicle that is in a parking space. Examples of algorithms that are used for such purposes are the adaptive background model, You Only Look Once (YOLOv3) or even Neutral network.

2.4.3.1. Adaptive background model

The use of an Adaptive background model for vehicle detection in parking space was presented by Kairoek Choeychuen in "Available car parking space detection from webcam by using adaptive mixing features." in 2012 [14]. This model consists of two steps. The first is to determine the offline background

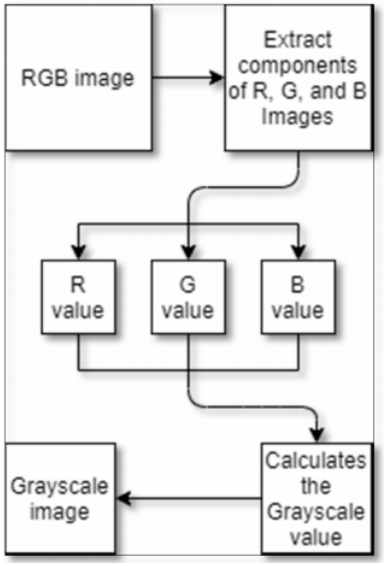

Fig. 2.5. Convert the RGB image to grayscale [13].

model and the second is to determine the online background model. The offline model is created prior to the start of detection, and the online model is a change over time of the offline model. The models determined in this way are then subtracted from each other, and based on this difference detection of an object is made (Fig. 2.6).

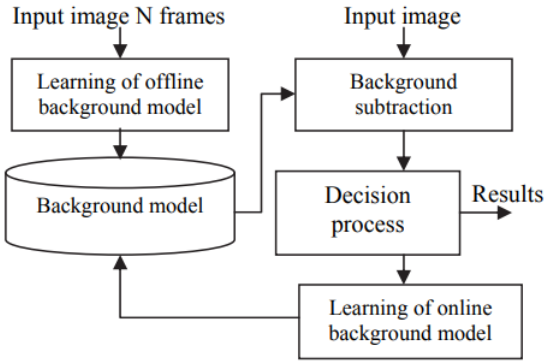

Fig. 2.6. Overview of object detection module [14].

The whole algorithm is shown in Fig. 2.7. Once an object is detected, a decision is made as to whether it is a vehicle. This is done based on a comparison of key appearance features. The next step is to eliminate the noise caused by changes in lighting. For this purpose, a combination of multiple data that were taken under different illumination was used. The number of combined samples is changed dynamically. Then occupancy/vacancy decision is made.

Such an algorithm allows for the efficient detection of parking space occupancy. Unfortunately, the author of this paper does not give the coefficient of correct detection, however he notes that the system works correctly and will be developed in the direction of creating a car parking map automatically.

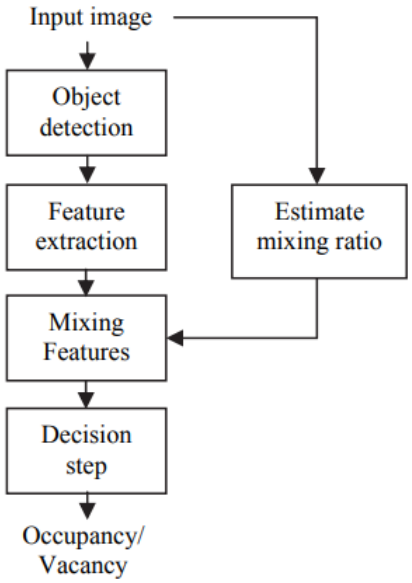

Fig. 2.7. Block diagram of appearance-based available car parking space detection [14].

2.4.3.2. YOLOv3

The use of The You Only Look Once (YOLOv3) algorithm for vehicle detection in parking space was presented in "Car Detection in Roadside Parking for Smart Parking System Based on Image Processing." by D. KManase, Z. Zainuddin, S. Syarif, A. K. Jaya in 2020 [13].

The algorithm was created to detect an object in real-time. It is done by using a repurposed classifier or localizer to make a detection. YOLO uses an artificial neural network (ANN) to detect objects in an image. This network divides the data into multiple areas and then predicts each bounding box and probability for each region. These bounding boxes are then compared to each predicted probability. System flow is shown in Fig. 2.8.

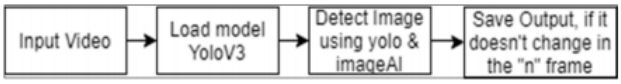

Fig. 2.8. Flow detection of cars using YOLOv3 [13].

The data used in paper were taken using a CCTV camera with a height of 2.5 meters (Fig. 2.9). The test was carried out using different scenarios. Detection results of parked cars using YOLOv3 with ImageAI produce an average accuracy of 96.88%.

2.4.3.3. Convolutional neural network

In "On-Street Parking Spot Detection for Smart Cities" written by S. Goren, D. F. Oncevarlk, K. D. Yldz, T. Z. Hakyemez in 2019, Convolutional Neural Network (CNN) was used. Initially, data were

Fig. 2.9. One example of the detection results using YOLOv3 [13].

collected and labeled. This allowed the classifier to be trained. The resulting model was tested by introducing new data for classification. Several different neutral networks were tested. The final one was implemented in a real application to determine the path to a free parking space. This shows that deep learning can be effectively used to create a free parking space detection system.

2.4.4. Vehicle detection via analyzing magnetic field changes

The Earth is a system of two dissimilar magnetic poles that produces a characteristic field called the dipole field. This field is subject to various types of anomalies such as deposits of iron ore, solidified volcanic lava. Such anomalies are areas where the direction of the magnetic field deviates from the average value for a given latitude. Earth's natural electric field is 100 V/m and the average magnetic induction is 40μT.

Electromagnetic fields are also generated by everyday objects. Every wire that carries a current generates such a field. Therefore, the presence of many devices such as a telephone, a television or a motor vehicle has an effect on the magnetic field value, which can be measured with a magnetic sensor. Analysis of magnetic field changes is widely used to classification of the devices causing the change, or to detect vehicle or even human movement [15].

2.4.4.1. Vehicle classification

Many vehicle classification research uses a video camera. This solution allows effective classification, but it is quite expensive. For this reason, a good alternative is to use magnetic sensors. They allow easier installation, have a low price, are not dependent on visibility and are small in size.

In "Vehicle Classification Method Based on Single-Point Magnetic Sensor" by Yao Hea, Yuchuan Dua, Lijun Suna in 2012 [16], a solution using measurements from singe-point magnetic sensor for vehicle classification was presented. In order to obtain the input data of the algorithm, the waveform signal was converted into a numerical format, data fusion technology for feature extraction was used. This process was followed by sample selection to decrease the dimension of the feature subset. In the presented paper this was done by using Filter-Filter-Wrapper model. The C-SVM algorithm was used for the classification purpose. The kernel function used was Gaussian radial basis function (RBF). Penalty

Table 2.1. Experimental results of vehicle identification based on C-SVM [16].

PSO-C-SVM Classifier	Reality				Identification Result	Accurancy
	Bus	S&M	Large Truck	Wrong Signal		
Bus	22	0	0	1	23	95.65%
Small and Medium	0	89	0	8	97	91.75%
Large Truck	0	0	5	0	5	100%
Sum	22	89	5	9	125	92.80%
Recall	100%	100%	100%	-	-	

factor, which can have a decisive effect on the performance of the classifier, and kernel parameter were determined by Particle swarm optimization (PSO). The developed algorithm was tested by deploying new 116 vehicles. The classification results are shown in Table 2.1. The results obtained in this research confirm the successful application of the magnetic sensor for vehicle classification.

Vehicle classification is problematic due to the large unbalanced datasets, e.g., the number of cars on the road is much larger than the number of buses. In the paper "Vehicle Classification Using an Imbalanced Dataset Based on a Single Magnetic Sensor" by Chang Xu, Yingguan Wang, Xinghe Bao and Fengrong Li from 2018 [17], the use of oversampling method SMOTE is presented. This algorithm randomly generates samples that are added to the learning sets for balancing. The classification of the vehicles was done using the most popular algorithms: the k-nearest neighbor, the support vector machine and the back-propagation neutral network. The study shows that the SMOTE algorithm increased the classification accuracy of the SVM algorithm from 77.27 to 79.55%, while the kNN algorithm increased it from 70.46 to 95.46%.

2.4.4.2. Dynamic vehicle detection

Magnetic sensors are used not only for vehicles classification, but also for detection. The presence of a vehicle, which mainly consists of metal parts, disturbs the magnetic field of the earth, which causes the magnetic sensor to change its measurements. Based on this, it can be derived whether there is a vehicle above the sensor or not.

In "Dynamic Vehicle Detection via the Use of Magnetic Field Sensors" by Vytautas Markevicius, Dangirutis Navikas, Mindaugas Zilys, Darius Andriukaitis, Algimantas Valinevicius and Mindaugas Cepenas in 2016 [18], an experiment was presented in which ARM sensors were used for dynamic detection and speed measurement. For detection, different vehicles were moved over two sensors placed 30cm apart. Disturbances in the earth's magnetic field allowed their detection. In order to determine the speed of the vehicle, the distance between the sensors was increased to 2m. Twelve methods were tested,

the best results were achieved when using only the Z component of the sensor measurements, this allowed average relative error of speed determination of 1.5%. The paper notes the great potential of using AMR sensors to study the change in the Z component of the magnetic field.

2.4.4.3. Vehicle detection in parking spot

Detecting a change in magnetic field, can also be used for static vehicle detection. A parking space is a good example of such an implementation. By placing a magnetic sensor in a parking space, it is possible to determine if the space is free or not. This may be a good solution in the context of Smart Parking in a Smart City, given the estimated low cost of a sensor per parking space.

As previously mentioned, such detection takes advantage of the change in the earth's magnetic field due to the presence of the vehicle. This is shown in Fig 2.10. This allows you to observe how the presence of the vehicle affects the change in the magnetic field of the earth. This suggests that the best solution would be to place the sensor underneath the vehicle, as it could read the largest field changes caused by both the front and rear of the vehicle. However, it would also be acceptable to place the sensor in front of the car, since in most current vehicles, the engine is placed in the front. This is the part that causes the most magnetic field changes.

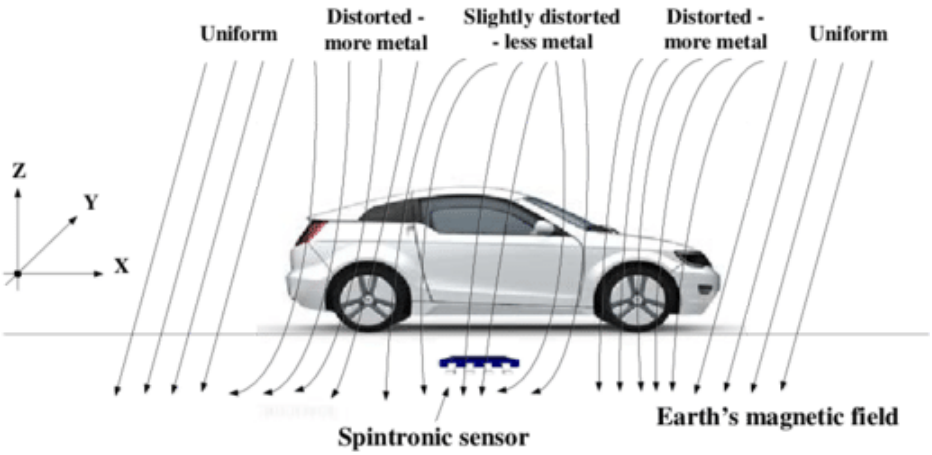

Fig. 2.10. The disturbance of the Earth's magnetic field by a passing vehicle. [19].

In "A remote test method for parking detection system based on magnetic wireless sensor network." by S. Qiu, H. Zhuand and F. Yuw in 2017 [20], a system using magnetic wireless sensor network (WSN) (Fig. 2.11) was presented. This solution shows a practical implementation of magnetic sensors for vehicle detection in a parking space, applied to a real Smart Parking system.

In "Implementation and Characterization of a Smart Parking System based on 3-axis Magnetic Sensors" by C. Trigona, B. Andò, V. Sinatra, C. Vacirca, E. Rossino, L. Palermo, S. Kurukunda and S. Baglio in 2016 [21], the use of 3-axis magnetic sensors for car detection in a parking space is presented. Tests were conducted using 5 sensors on consecutive parking spaces. The system created correctly verified the presence of a vehicle in a parking space, but only for one car. When a second vehicle occupied

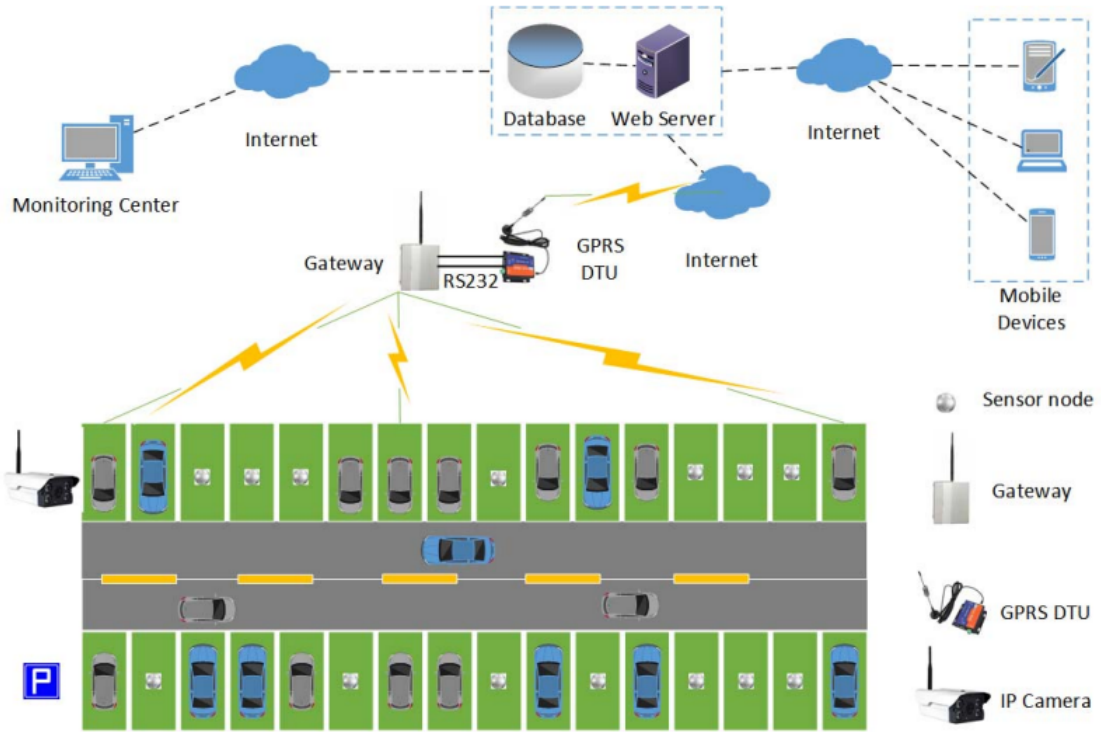

Fig. 2.11. The parking detection system [20].

the adjacent space, interference and erroneous measurements occurred. The paper suggests removing background noise from the readings to avoid this problem.

In "Wireless magnetic sensor node for vehicle detection using finite element simulation." by Q. Zhang, G. Wang, K. Gong, B. Li, M. Meng and S. Song in 2013 [22], a simulation of using wireless magnetic field sensors for vehicle detection was presented. In this simulation, the magnetic field of the earth was created and then its change from the moment of oncoming vehicles was tested. The disruption of the earth's magnetic field was mainly caused by the engine and wheels, which have a lot of ferromagnetic components in them. The field change was measured using a two axis magnetic sensor.

In the simulation, 3 adjacent parking spaces were presented (Fig. 2.12). Parking spaces 1 and 3 were assumed to be vacant and the vehicle parked only in space number 2. The readings from the parking spaces were then collected. The presence of a vehicle in parking space number 2, caused a change in the readings of sensors 1 and 3. It follows that it is very likely that the sensors will not show the correct readings, but will be disturbed by the presence of a vehicle in an adjacent parking space. The paper suggests that in order to avoid such interference, a photosensitive resistance sensor could be used in addition to the sensors to improve detection performance.

An interesting solution that can eliminate the problem of interference caused by the presence of vehicles in adjacent parking spaces is to use machine learning algorithms to analyze sensor readings. In such a situation, not only the readings from a given parking space but also from two neighboring parking spaces are analyzed to determine the presence of a vehicle. In this approach, the correctness of

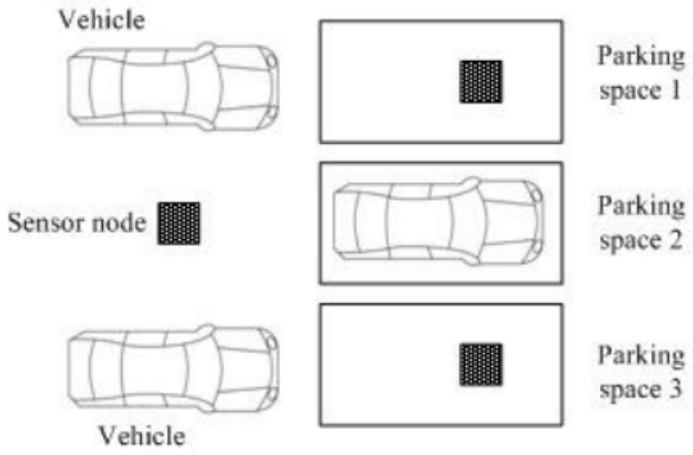

Fig. 2.12. The schematic of three different vehicles parking over the sensors. [22].

detection can be independent of interference from neighboring spots. An analysis of such a solution will be realized in this thesis.

2.5. Machine learning

Machine learning is a subset of artificial intelligence (Fig. 2.13). It is based on a computer program performing certain tasks. The machine learns from its previous experience. If the task can be evaluated, it gains more and more experience with the number of tasks performed. The machine makes decisions and predictions based on the data [23]. In machine learning, algorithms are trained to find patterns and correlations in large data sets. Systems using such a solution are becoming more and more effective and therefore are used more and more often. Machine learning is used in many fields such as medicine, multimedia, image processing and object detection.

Machine learning includes various models and techniques. There are three main machine learning models: supervised learning, unsupervised learning and reinforcement learning. The model is chosen based on the problem it is to be used for. Each of these models has many different algorithms used. They mainly focus on data classification, pattern search and decision making.

2.5.1. Supervised learning

In supervised learning, learning is based on examples. The training data is classified before learning begins. This allows the algorithm to know what the expected value is. For example, we want to use the algorithm to distinguish a banana from an apple. So the input images will be labelled as apple or banana. Based on this, the algorithm will be able to classify subsequent images because it "knows" what an apple and a banana look like from the training data.

There are many machine learning algorithms that use supervised learning, such as:

– *Nearest Neighbour*

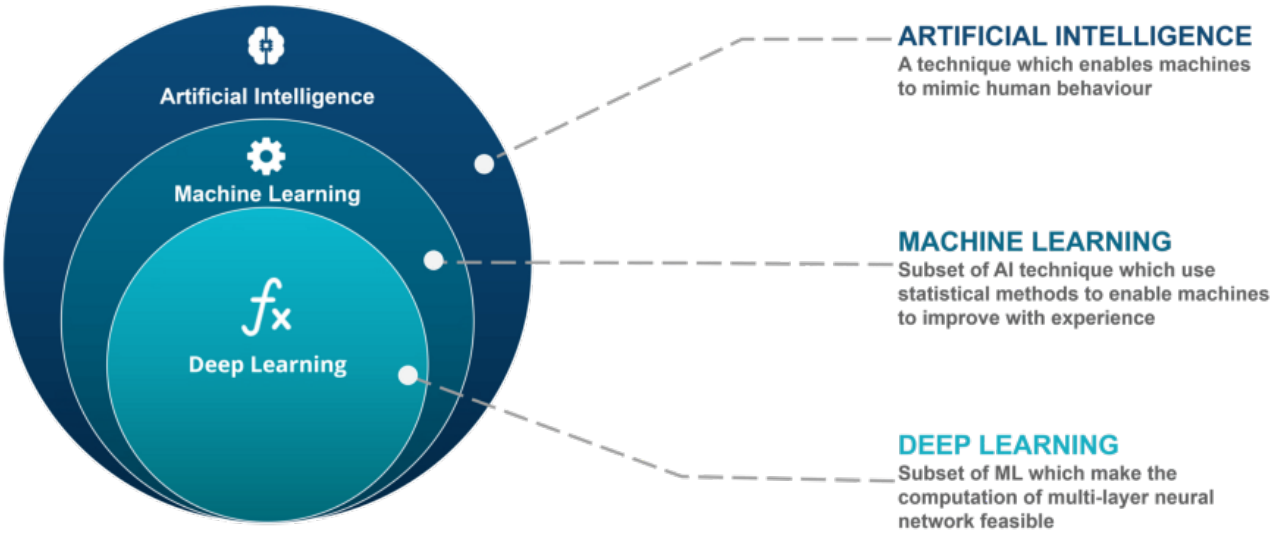

Fig. 2.13. Artificial intelligence subsets [24].

– *Naive Byes*

– *Decision Trees*

– *Linear Regression*

– *Support Vector Machines (SVM)*

– *Neural Networks*

Some of them will be described in more detail in chapter 4.

2.5.2. Unsupervised learning

In unsupervised learning, the input data is not classified/labelled in any way. The detection of patterns and correlations is done only by analyzing the input data, without the expected output value of the learning data. Examples of machine learning algorithms using unsupervised learning are:

– *K-means Clustering*

– *Hidden Markov Model*

– *DBSCAN Clusterings*

– *Principal component analysis (PCA)*

2.5.3. Reinforcement learning

Supervised learning uses labelled training data, unsupervised learning uses unlabeled training data. A reinforcement model, on the other hand, does not use labeled input data but a set of allowed actions,

rules and potential final states. When the goal of the algorithm is changeable, a system of rewards and experiences is used. Applications of reinforcement learning include mechanisms for automated price bidding for buyers of online advertising space, developing computer games, and playing the stock market at high stakes.

3. Test stand

The proposed system is based on the analysis of the change of the Earth's magnetic field, in order to detect the vehicle. However, in the literature presented in Section 2, it is noted that the use of thresholding alone is not effective enough. The possibility that vehicles are present in adjacent parking spots can significantly affect the sensor readings, and therefore can result in detection errors. In this thesis I decided to examine the use of machine learning algorithms to eliminate this problem and to perform an efficient vehicle detection.

In order to perform research a real parking system was created. It consists of 3 parking spaces located adjacent to each other. The change in the earth's magnetic field is measured at each spot. The collected data is then sent to computer and analyzed by machine learning algorithm. The decision made about the presence of a vehicle is later published on the website.

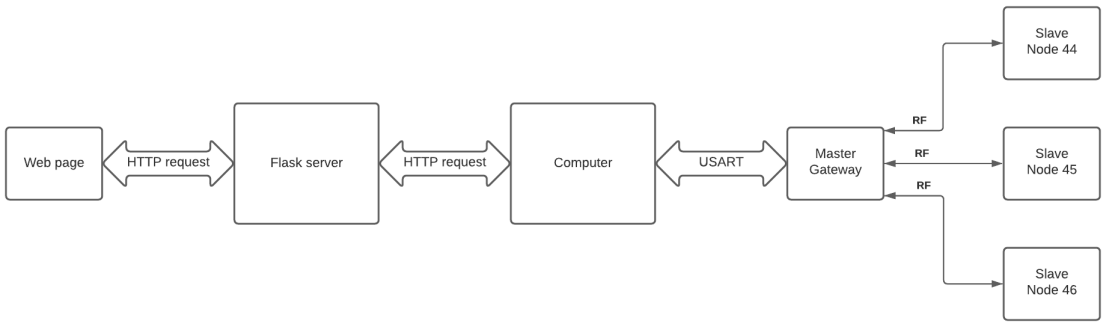

Fig. 3.1. System block diagram.

3.1. System schema

The block schema of the system is shown in Fig. 3.1. The parking space occupancy sensor was developed by Dr. Więckowski (Slave Node 44-46). They form a master-slave topology in which magnetometer readings are sent to the gateway. This communication is done via radio, this allows them to be used wirelessly. The master sends incoming data frames via the USART to a host computer, which is running a script with a machine learning algorithm, that makes parking space occupancy decisions. The data received from the master and the parking space occupancy prediction are sent via HTTP requests

to a local server, running the flask framework. In this way, HTTP requests sent from the browser to the localhost allow it to send a response with the current status of the parking space and its occupancy.

3.2. Parking space occupancy sensor module

To measure the change of the earth's magnetic field, a detector for the presence of a vehicle in a parking space was used. A few changes were made to the sensor software and transmission method, but mainly it was used as shown in the technical documentation [25].

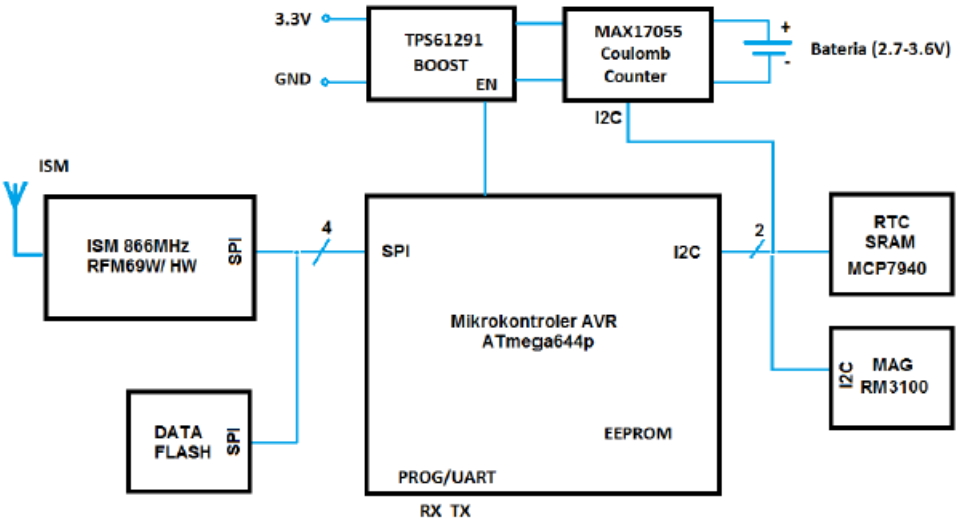

Fig. 3.2. Sensor block diagram [25].

3.2.1. Sensor's hardware

The block diagram of the prototype detector electronic circuit is shown in Fig. 3.2. It consists of the following main components [25]:

– PCB with ATmega644p processor from Atmel/Microchip. The microcontroller is responsible for reading and processing data from the magnetic sensor, the ISM (or LORA) radio module and the RTC real time clock circuit.

– The ISM 868 radio module of type RFM69W or RFM69HW. It is the basic component of the detector communication with the parking meter controller. Communication with RFM69W is performed using SPI interface.

– The RM3100 type precision magnetic field sensor, made with MEMS (magneto-inductive) technology. This chip contains a 3-axis magnetometer that can be successfully used to detect the presence of a vehicle in a parking space. The configuration and data reading from the sensor is done using an I2C interface.

– A real time RTC clock module of type MCP7940. The chip has currently been used to store calibration parameters and vehicle presence status. It is an internal 64-byte SRAM with battery backup. Other uses for the RTC chip include accurate timing of events, advanced wake-up functions (configurable day/night mode).

– Data Flash memories (4-Mbit). Configuration and data reading from Flash chip is performed using SPI interface. Flash memory can be used in the process of wireless Over-the-Air programming (OTA) firmware update or to store diagnostic information. The software update functionality is the subject of a separate study planned for the future.

– Power Module with TPS61291 chip. The battery voltage converter operates in two modes: battery voltage (by-pass) and battery voltage boost to 3.3 VDC (Boost Coverter). During the commissioning work it was possible to eliminate the need to operate the converter system for the battery voltage range of 2.9 - 3.6V. In this range, the detector system behaves stably. However, it is recommended to turn on the booster when the battery voltage drops below 2.9V. The minimum battery voltage guaranteeing stable operation (TPS61291 inverter circuit switched on) was determined experimentally and is 2.25V.

– Battery controller (coulomb counter), based on the MAX17055 chip. An independent battery controller, it provides precise current, voltage and temperature measurements, with very low current consumption (7uA).

The detector's components i.e. microcontroller, power supply system, magnetic sensor and RTC clock module, are placed on the fabricated PCB (electronic circuit). The ISM RFM69W/HW radio module has been placed on the detector's bottom side. The general appearance of the detector is shown in Fig. 3.3.

The following are the main assumptions from the technical documentation that the hardware part of the detector fulfills [25]:

– Non-contact detection method using a magnetic field sensor.

– Measurement of the temperature inside the detector housing and the voltage value of the battery powering the detector.

– The detector has an independent power supply, using a lithium battery with a nominal voltage of 3.6V and a basic capacity of 8.5 AH (size C /R14) [26] and with an increased capacity of 19 AH (size D/R20) [27].

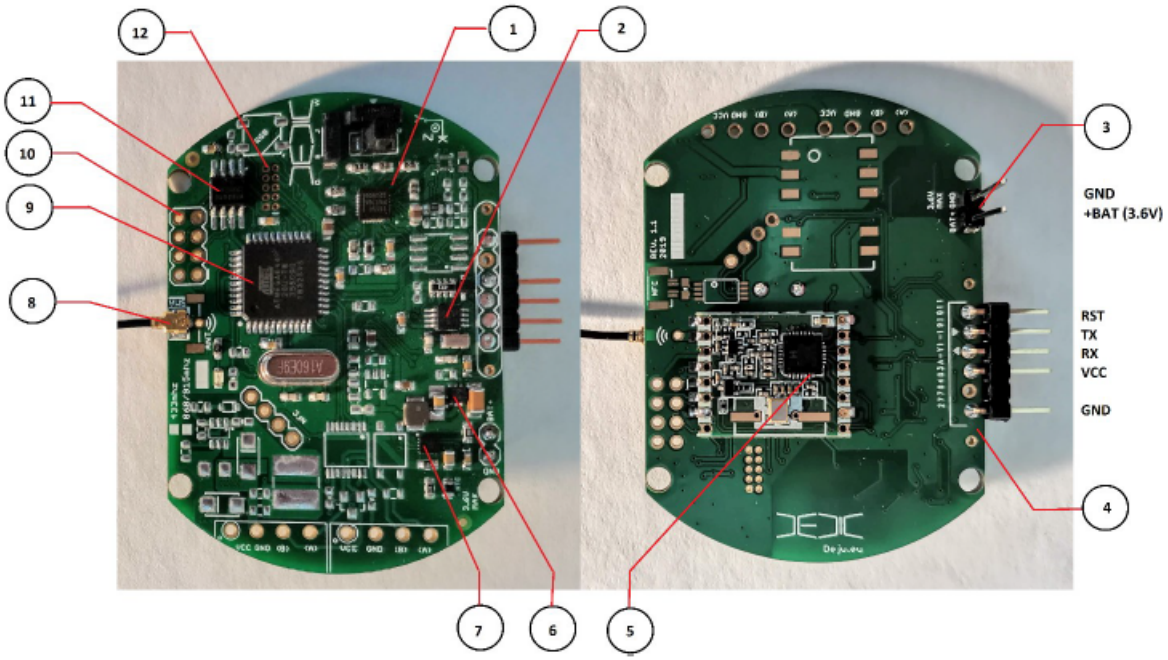

Fig. 3.3. General view of the detector: 1- RM3100 magnetometer, 2- MCP7940 RTC clock, 3- battery power connector, 4- UART/ programming connector, 5- RFM69W/HW, 6- Boost converter (TPS61291), 7- battery supervisor (MAX17055), 8- u.fl. antenna connector, 9- microcontroller (ATmega644p), 10- AVR ISP connector, 11- Data Flash memory (4Mbit e.g. W25X40), 12- AVR JTAG connector [25].

– It has key parameters responsible for correct communication of the detectors with the base station (unique ID number, unique group number - networkID, encryption key, etc.). The parameters are stored in the internal non-volatile EEPROM memory. They can be changed from only the detector via the serial interface.

– The detector requires periodic calibration. Calibration can be performed remotely - initiated by the host system via a radio link.

– The detector communicates with the base station using the RFM69W/HW radio transceiver operating in the ISM band. The maximum range to ensure proper communication in the parking zone was determined to be approximately 200 m in a straight line from the detector to the hub (star topology).

– It is possible to restore the device to the initial state (CURRENT RESET), by activating a magnetic switch.

– The operating frequency of the radio transmitter/receiver for communication with the concentrator is: EU 868 MHz ISM Band (Europe) Smart Parking EU.

– The operating temperature range of the detector electronics is -20 to +65°C.

The printed circuit board (PCB) of the prototype detector, was designed to be placed together with the lithium battery, in a cylindrical housing with dimensions: 36 x 40 mm (R x H) shown in Fig. 3.4. Commercially available enclosures, made of PET / PETE (polyethylene terephthalate), can be used in further research and development. The housing provides an ingress protection class > IP68. The detector sensor itself (PCB board), fits within the outline of a circle of 62 mm diameter.

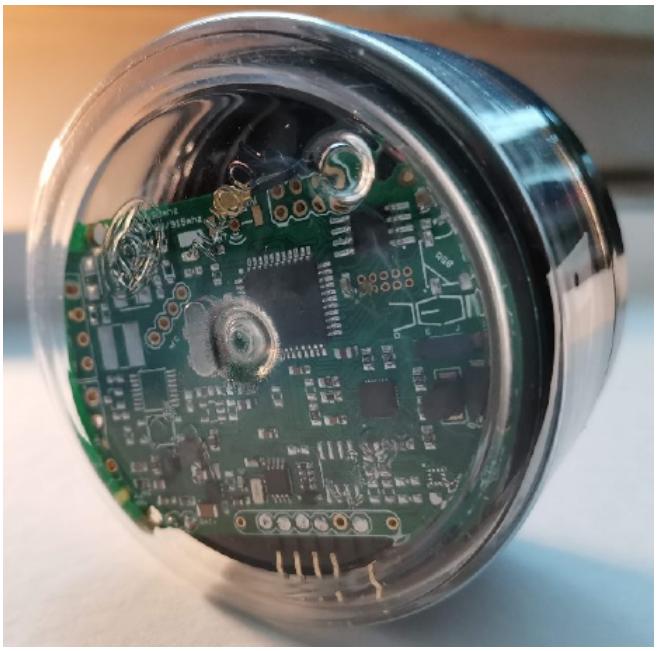

Fig. 3.4. Sensor view in PET / PETE housing [25].

3.2.2. Communication

Communication between the sensor and the master takes place via RF. Initially, there were two types of frames used for communication: a configuration frame and an information frame. The first one has not been modified and information about it can be found in the technical documentation. The information frame has been slightly modified by adding timestamp, which further allowed for better data analysis and visualization. The content of the configuration frame is shown in Table 3.1.

To optimize communication and data flow, several solutions were tested. The first idea was to send a large amount of collected data on request to the master. When a request was sent, the slave was supposed to send back a larger packet of the latest stored data. Unfortunately, the problem in such a solution turned out to be the maximum message length, through the library used for radio communication [28]. It allows to send maximum 61 bytes, while one data frame has 44 bytes. This means that it is possible to send only one frame at a time.

The second transmission method considered was to store the data in the sensor's memory and send it to the master when a certain threshold value of the sensor readings was exceeded. This solution was abandoned for several reasons. The first was once again a problem with the maximum message length.

Table 3.1. Information frame elements sent to the serial port.

Nr.	Field name	Type	Description
1.	timeStamp	unsigned long	Measurement time
2.	nodeID	unsigned char	Sensor's identification number
3.	carFlag	unsigned char	Occupancy status based on thresholding
4.	magValueX	signed int	The magnetic field value for the X axis of the sensor.
5.	magValueY	signed int	The magnetic field value for the Y axis of the sensor.
6.	magValueZ	signed int	The magnetic field value for the Z axis of the sensor.
7.	batteryVoltage	unsigned int	Battery voltage value, expressed in [mV] *100
8.	batteryCurrent	signed int	Instantaneous value of the current drawn from the battery, expressed in [mA] *100
9.	boardVoltage	unsigned int	Value of the detector power supply bus voltage, expressed in [mV] *100
10.	boardTemp	signed int	Temperature in the detector housing, expressed in []°C]*100
11.	radioRSSI	unsigned char	RSSI measurement relative quality of the radio signal that was received by the detector
12.	radioTransmitLevel	unsigned char	saved powerLevel
13.	boostEnable	unsigned char	Status of the boost converter {0:Disable/1:Wneble}
14.	rstSrc	unsigned char	Restet
15.	debug	char*	Information if status changed ("change..") or not ("alive...")

The second was a problem with the memory in the device, it would be impossible to store large amounts of data without installing additional memory. Another was the lack of continuous transmission. When using the data to train a machine learning algorithm, data from the non-presence of the vehicle is also important. Additionally, continuous transmission would allow the sensor status information to be made available more quickly to a verification system (e.g. a website).

Based on previous attempts, it was finally decided to have continuous instantaneous data transmission. This means that a single data frame is sent immediately after reading the data from the sensors, without analyzing its value. This solution is also not ideal because it causes high load on the radio channel. In case of a greater number of sensors, the number of channels should be increased, however, in case of data transmission from 3 sensors, one channel is sufficient. Data from each sensor is read and sent to the master every 500ms.

In addition there are 2 frames sent from the master to each sensor. The first one is the calibration frame which resets the offset on the sensors, this sets the sensors to position 0. The sensors are programmed in such a way that they do not send data before receiving at least one calibration frame. The second additional frame is the synchronization frame. It sends the master program time. The sensors then

synchronize their time with it, which is stored in each measurement. In this way, the measurements sent from each sensor are synchronized, allowing them to be put together. This frame is sent to each sensor every 12s to maintain synchronization.

3.3. System startup

Once the communication method was developed, initial attempts were made to commission and test the sensors system. However, one significant problem was encountered. Communication was only possible with one particular sensor (node 45). This behavior was unexpected, as the same program was uploaded to all three sensors. The first conclusion was that it was possible to communicate with only one sensor at a time, however even when only one sensor was active the communication still could not be initiated. As a second step the hardware part of the sensors was checked, but after the tests it was concluded that all modules, including the radio module, were working correctly. After some time an error was found in the program code. It was caused by the CmdMessenger.h [29] library used. It was used to configure the sensor through a GUI using the serial port. It was not determined why it was causing the error (probably the communication with the serial port could not be established, thus the program was falling into an infinite loop) and why one of the sensors was working correctly despite having the same piece of code. After solving this problem the transmission with all 3 sensors was successfully established. An example of the sensor readings returned to the serial port is shown in Fig. 3.5.

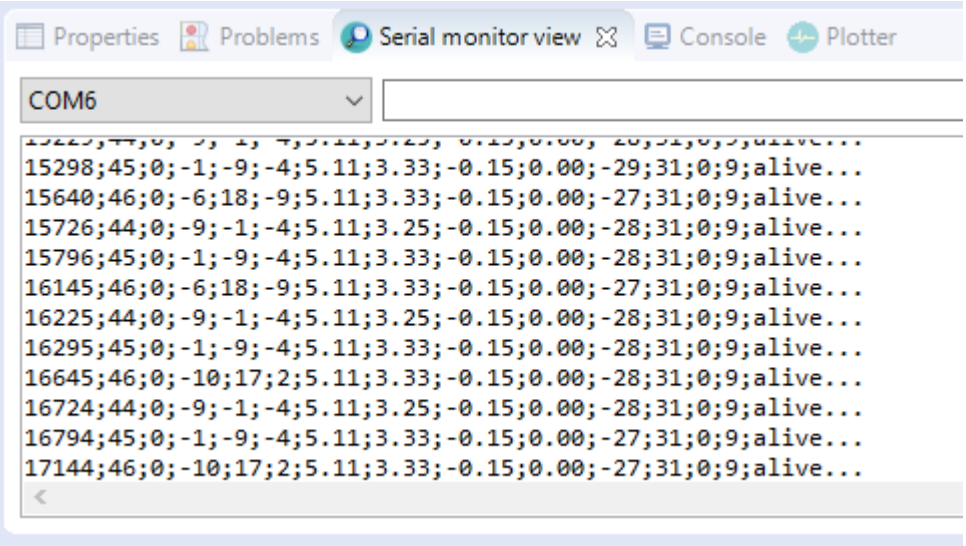

Fig. 3.5. Vehicle parking in space with node 45.

After successfully testing the sensors under laboratory conditions, it was decided to test their performance in outdoor environments. A series of preliminary experiments were conducted in an actual parking lot. The placement of the sensors in the parking spaces and the position of the master are shown in Fig 3.6. After activating the sensors and calibrating them, the vehicle started to enter the parking space. At the start of the measurement, the vehicle began the parking manoeuvre. When the vehicle was parked

correctly, the measurement was stopped. A sample measurement from this experiment is shown in Fig. 3.7.

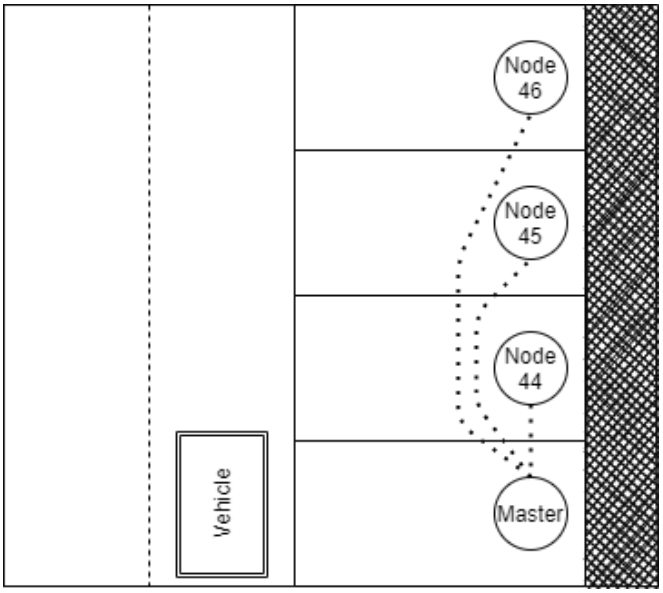

Fig. 3.6. Sensor arrangement in parking area used for experiments.

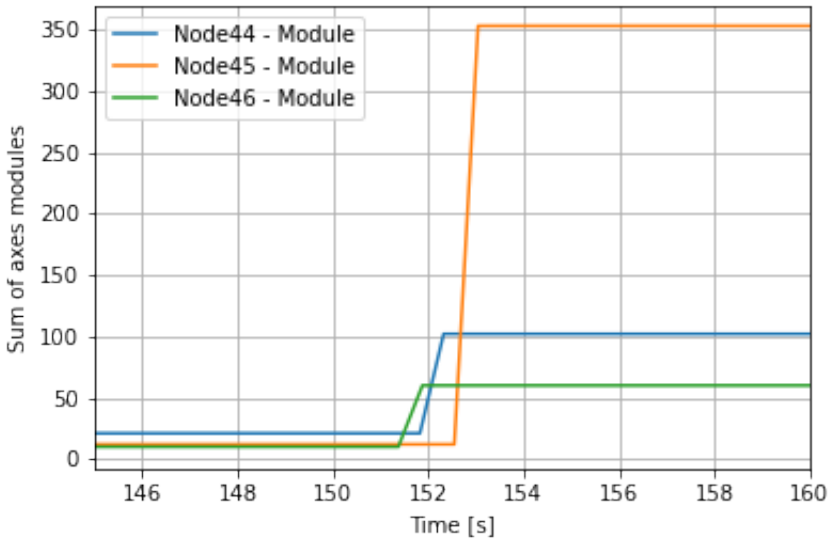

Fig. 3.7. Vehicle parking in space with node 45.

It was also decided to check the sensors' behaviour when a vehicle drives along with the parking spaces, i.e. at a distance of about 6m from the sensor (Fig. 3.8). The change of the earth's magnetic field was negligible in this case, so it would not be possible to detect such a crossing. Additionally, the distances from which the sensor detects the vehicle were also examined. These distances are accordingly

2 lengths of the parking space, one length of the parking space and the vehicle parked in the parking space. Results are shown in Fig. 3.9.

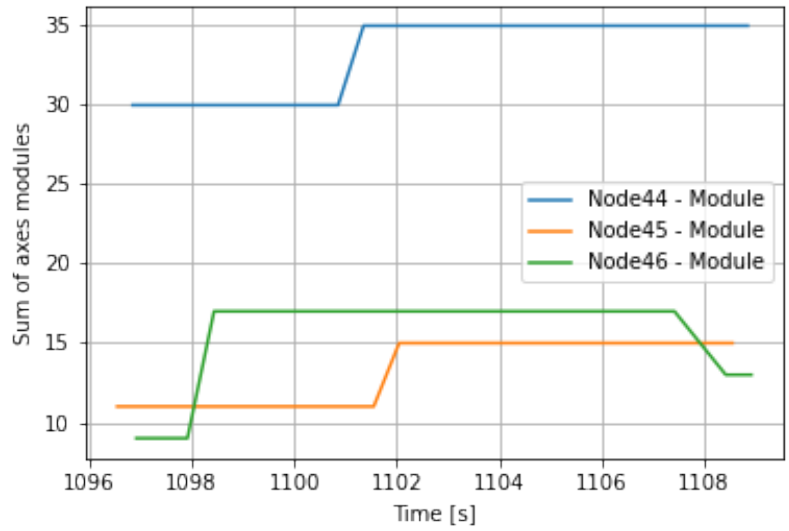

Fig. 3.8. Vehicle moving near the parking spaces.

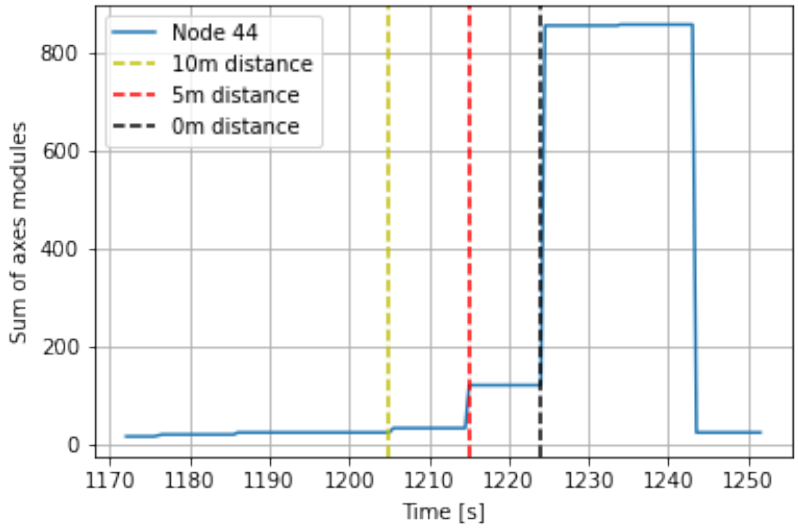

Fig. 3.9. Change of magnetic field value based on vehicle distance.

An experiment was also conducted in which vehicles were parked at all three spots. In this case, the connection to sensor 46 was lost, probably due to poor placement of the master, so that the signal may have been blocked by parked vehicles.

The main conclusions that were drawn after the experiments and analysis of the collected data:

– The sensors are placed in a plastic casing together with a battery, which means that they are not always aligned perpendicularly to the ground. As a result, changes in the magnetic field caused by an oncoming vehicle are visible not only on the Z axis but also on the X and Y axes. Therefore, when analyzing the data, attention is paid to the sum of the data moduli from all axes and to the sum of squares.

– Sensors should be placed towards the master in such a way that oncoming traffic does not block communication between them.

– Vehicles passing along the parking area (not on the side of the sensors) have no significant impact on the magnetometer readings.

– Vehicles affect the earth's magnetic field differently, depending on the size of the vehicle, the size of the engine, and the number of electronics in the vehicle. Therefore, the sensor values will be different for different cars.

4. Machine learning algorithms

The created system was able to correctly send the data read from the sensors to the master unit (in this case a computer). This allowed the data to be analyzed from there. As previously mentioned, the data representing the change in magnetic field values will be analyzed by a machine learning algorithm, which will determine a prediction about whether a seat is occupied or vacant. For this purpose, a training dataset was collected and labeled. It was then used to verify several supervised machine learning algorithms. The algorithm with the best results was then implemented on the real system.

4.1. Collection of training data

In order to collect training data, an experiment was performed on a real parking area. During the design, the conclusions of the first real measurement attempt were taken into consideration. This time, also data readings from three consecutive parking spaces were measured, but the master was placed in front of the sensors instead of next to them as before. The sensor arrangement in the parking area is shown in Fig. 4.1.

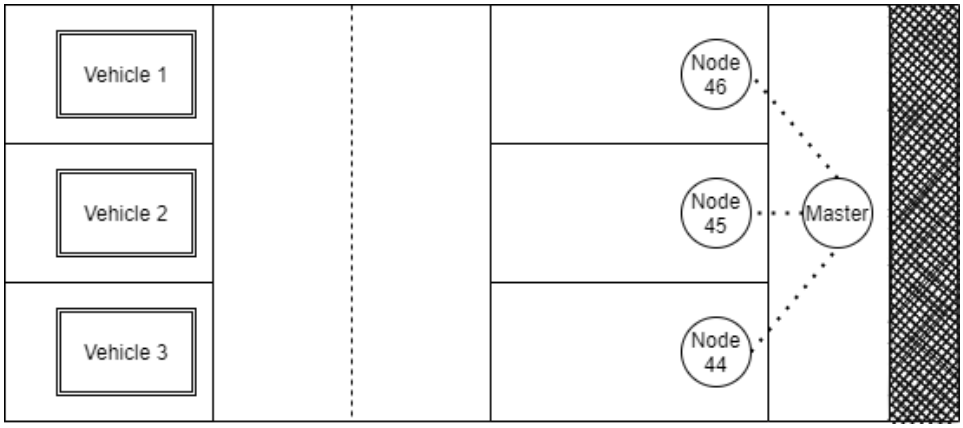

Fig. 4.1. Sensor arrangement in parking area used for training data collection.

The experiment started with sensor calibration, then the data from the serial port was saved to a .txt file, this was done using CoolTerm. With such a prepared system the actual part of the experiment began.

Vehicles started approaching the parking spaces in a different order, in such a way as to consider all possible cases, e.g., a car approaching the middle parking space, then a car approaching the bottom parking space, and so on. All possible scenarios are described in Table 4.1.

Table 4.1. Parking scenarios.

Nr.	Node 44	Node 45	Node 46	Label
1.	0	0	0	0_0_0
2.	1	0	0	1_0_0
3.	0	1	0	0_1_0
4.	0	0	1	0_0_1
5.	1	0	1	1_0_1
6.	1	1	0	1_1_0
7.	0	1	1	0_1_1
8.	1	1	1	1_1_1

Three different cars were used for the experiment, their main parameters are shown in Table 4.2. The experiment was repeated three times so that each vehicle was in the middle spot at least once. The measurement data obtained are shown in Fig. 4.2-4.4.

Table 4.2. Vehicles specifications.

Nr.	Model	Curb weight [kg]	Engine Capacity [cm^3]	Year of production
1.	Opel Corsa -E	1162	1398	2017
2.	Toyota Corolla	1290	1598	2013
3.	Opel Astra 1.6	1065	1598	2000

From the graphs it can be seen that the greatest changes in magnetic field values are caused by the Toyota Corolla, this may be due to the most built-in electronic systems, as this car is the highest class of the three test vehicles. The graphs also show significant differences in the magnetic field produced by the same vehicle. This is caused by the distance the car reaches, as it can be directly over the sensor, but it can also be several centimeters in front of it and still fit in the parking space.

After collection, the measurement data were processed. Due to the different positions of the sensors (not perpendicular to the ground), the analysis of the measurements is based on the sum of the modules and the sum of the squares of all axes. This will allow an algorithm to be created that is independent of sensor orientation. The timestamp and readings from the three axes were extracted from the configuration frame. The timestamp was used to group the measurements from all 3 locations as one data. From the measurements from all axes, the sum of the moduli and the sum of the squares of the measurements were determined. The data processed in this way were combined together, then based on the knowledge of the parking order and data analysis, the measurements were manually labelled. An example of the training

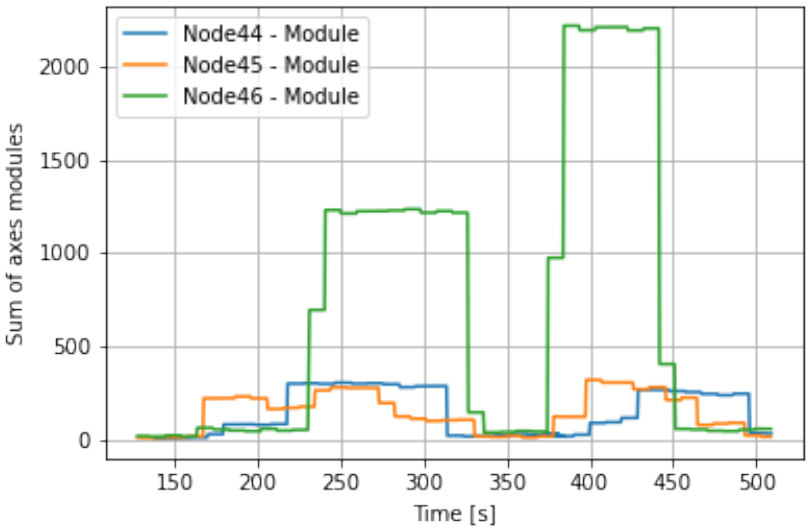

Fig. 4.2. Node44: Opel Astra 1.6, Node45: Opel Corsa -E, Node46: Toyota Corolla.

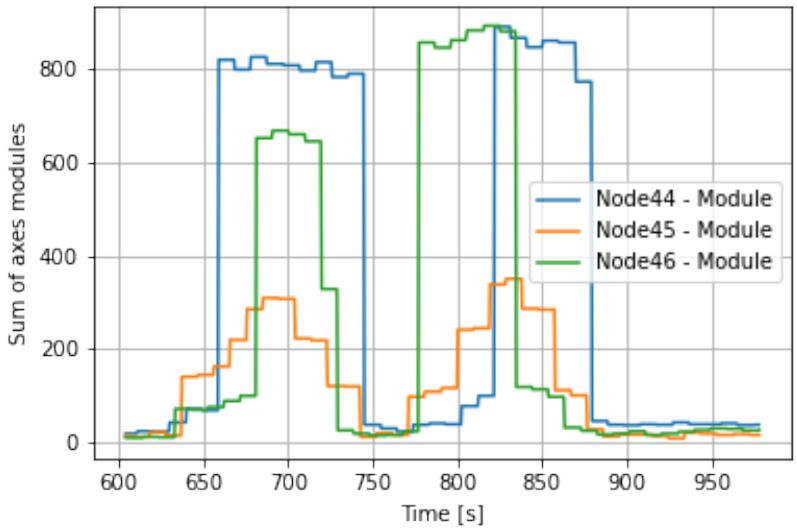

Fig. 4.3. Node44: Opel Corsa -E, Node45: Opel Astra 1.6, Node46: Toyota Corolla.

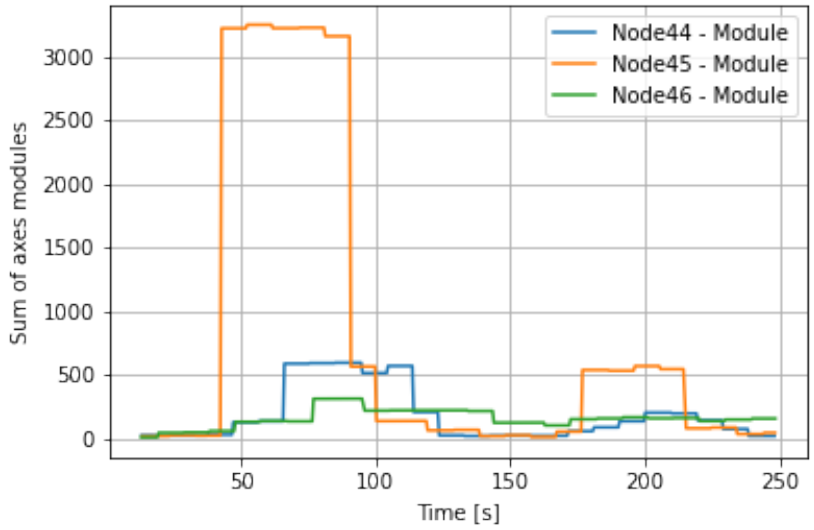

Fig. 4.4. Node44: Opel Corsa -E, Node45: Toyota Corolla, Node46: Opel Astra 1.6.

	Module-44	Power-44	Module-45	Power-45	Module-46	Power-46	Value
0	7	29	11	49	19	149	0_0_0
1	7	29	11	49	19	149	0_0_0
2	7	29	11	49	19	149	0_0_0
3	7	29	9	41	19	149	0_0_0
4	8	26	9	41	19	149	0_0_0

Fig. 4.5. Training data with labeled value.

set is shown in Fig. 4.5. The performed measurements resulted in approximately 2500 measurement points.

4.2. K-nearest neighbors algorithm

One of the simplest machine learning algorithms is the k-nearest neighbor classifier. This method classifies an identified object based on the class to which most of its K nearest neighbors belong. To determine the class of neighbors, the algorithm uses a training dataset. For a new observation y_0, k points x_r, r =1,...k, closest to point y_0 are found. The classification is then performed based on majority voting for the k neighbors [30].

In the considered system is a 6-dimensional problem, because the analysis by the algorithm will be given the sum of modules and the sum of squares of all axes, of each of 3 parking spaces. Euclidean distance was used as the distance metric. The distance between two points in Euclidean space is the

length of a line segment between two points. It can be calculated from the Cartesian coordinates of the points using Pythagoras' theorem, which is why it is sometimes called the Pythagorean distance. Equation 4.1 presents a mathematical formula for determining this distance.

$$d_e(x,y) = \sqrt[2]{(x_1 - y_1)^2 \ + \ (x_2 - y_2)^2 \ + \ ... \ + \ (x_N - y_N)^2} \tag{4.1}$$

As mentioned before, the parameter k will determine the number of nearest neighbors that are considered in classifying a new observation. A small value of k may cause instability in the algorithm and high susceptibility to disturbances. On the other hand, too large value of k parameter may lead to a selection of neighbors located in other classes. There is no single selection of parameter k, so it was determined experimentally. The effectiveness of the algorithm was tested for different values of the parameter. The results of this experiment are shown in Fig. 4.6. The value that is chosen is the largest possible value, but with high classification efficiency. In the plot we can read that the most optimal value is $k = 5$, because from this value the effectiveness starts to decrease drastically.

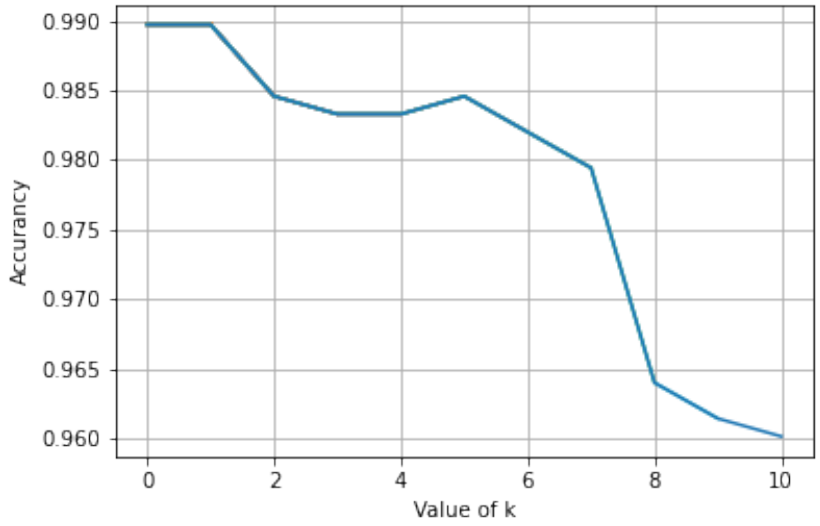

Fig. 4.6. Accuracy of the algorithm depending on k.

The algorithm was created in python language using KNeighborsClassifier [31] function. It allows the creation a classifier based on training data and k parameter. The collected data were normalized and then subdivided 70:30 into training and test sets. The parking events are classified into 8 different classes (Table 4.1). The created classifier allowed a 98.33% success rate. This result is very good, it was able to correctly classify 765 out of 778 samples. Misclassifications are shown in Table 4.3, these are the classes to which misclassified samples should be assigned. It can be seen that most occur when there is no vehicle in any of the three spots. To eliminate this problem, more training data should be collected from this class so that a small change in the field value does not cause misclassifications.

Table 4.3. kNN algorithm's misclassifications.

Class	0_0_0	1_0_0	0_1_0	0_0_1	1_0_1	1_1_0	0_1_1	1_1_1	Total
Misclassifications	8	0	1	1	3	0	0	0	13

4.3. Logistic regression

One of the most popular varieties of regression analysis is logistic regression. The most important feature of logistic regression is that the dependent variable (explained, predicted) is a dichotomous variable, that is, it takes two values, most often 0 and 1. Unfortunately, in the case of this problem, it will not work well, because there are 8 possible classifications. However, we have tested the use of logistic regression to determine the occupancy status of only the middle place, thus reducing the problem to a binary one. The values of the dichotomous variable can be transformed into the form of the probability of occurrence of a given event, which takes values between 0 and 1. When the logit transformation (4.2) is applied, it is possible to linearize the logistic regression model and present it as linear regression.

$$logit(p) = ln\frac{p}{1-p} \tag{4.2}$$

Then the logistic regression model defines equation 4.3. The left-hand side of the equation is the conditional probability that the variable will take a value equal to 1 for independent values $x_1, x_2, ..., x_k$.

$$P(Y = 1|x_1, x_2, ...x_k) = P(X) = \frac{e^{a_0+\sum_{i=1}^{k} a_i x_i}}{1 + e^{a_0+\sum_{i=1}^{k} a_i x_i}} \tag{4.3}$$

where:

$a_i, i = 0, ..., k$ are the regression coefficients,

$x_1, x_2, ..., x_k$ are independent variables.

As with the kNN algorithm, a ready-made function from the python libraries LogisticRegression [32], was used for the logistic regression algorithm. It created a classifier based on training data and test data. The expected value was reduced to a binary problem of middle space occupancy detection (Node 45). The algorithm thus trained was put to test. It achieved an efficiency of 97.12%. This is again a very good result, slightly worse than that of the kNN algorithm. However, it should be noted that the classification was done only for the middle place, excluding the other 2 places. The classification result is good enough that it could also be implemented in a single parking spot.

4.4. Decision tree

A decision tree is defined as a tree representing the division of a set of objects into homogeneous classes [30]. In such a tree:

– the internal nodes describe how the division into classes is made, based on the values of the objects' characteristics,

– leaves correspond to classes of objects,

– edges represent values of features, based on which the division was made.

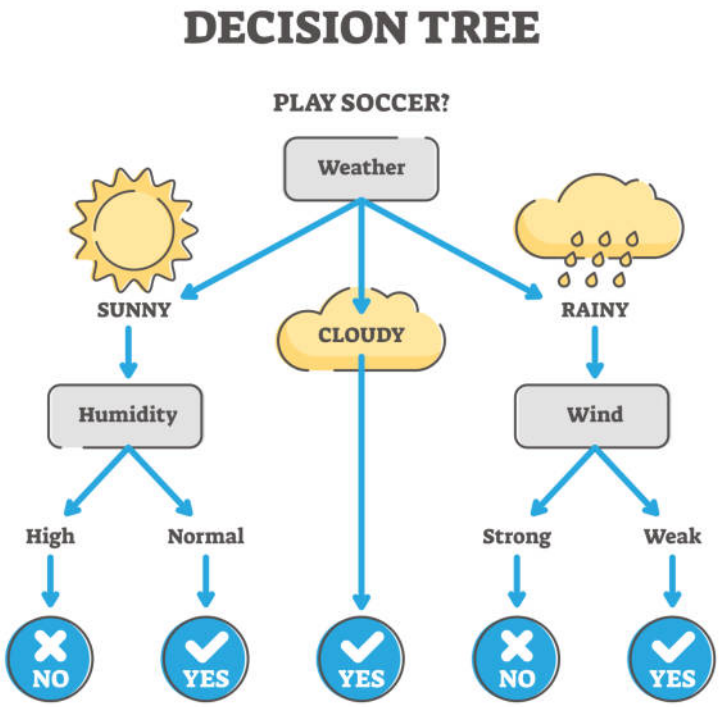

Fig. 4.7. Example of a decision tree [33].

An example of a decision tree is shown in Fig. 4.7. It is used to decide whether to go play soccer depending on the weather. Inner nodes represent weather parameters (weather, humidity, wind), while edges represent values of these parameters (sunny, cloudy, rainy), based on which the decision is made. Leaves represent the class to which the observation is assigned, in this case it will be the decision whether to go play soccer.

The algorithm was implemented in python language. DecisionTreeClassifier [34] function was used to create a classifier. The algorithm thus created was trained and then tested. It allowed an efficiency of 88.95%, this result is also very good, slightly worse than the kNN algorithm. Misclassifications are shown in Table 4.4, it can be observed that for this algorithm, the most errors occurred when a vehicle was present only at the extreme top position (node 46), this may be due to over-fitting the tree to the training data, as in 2 out of 3 experiments performed, the Toyota Corolla (Table. 4.2) was present at this position, which causes the largest field changes. This may have caused the presence at this location to be dependent on the value of the change caused by it.

Depth (maximum distance between the root and any leaf) of the tree is equal 7 and has 23 leaves (Fig. 4.8). This means that the constructed tree is relatively small. However, further learning of the tree may cause significant increase in depth and number of leaves of the tree, which may result in instability of

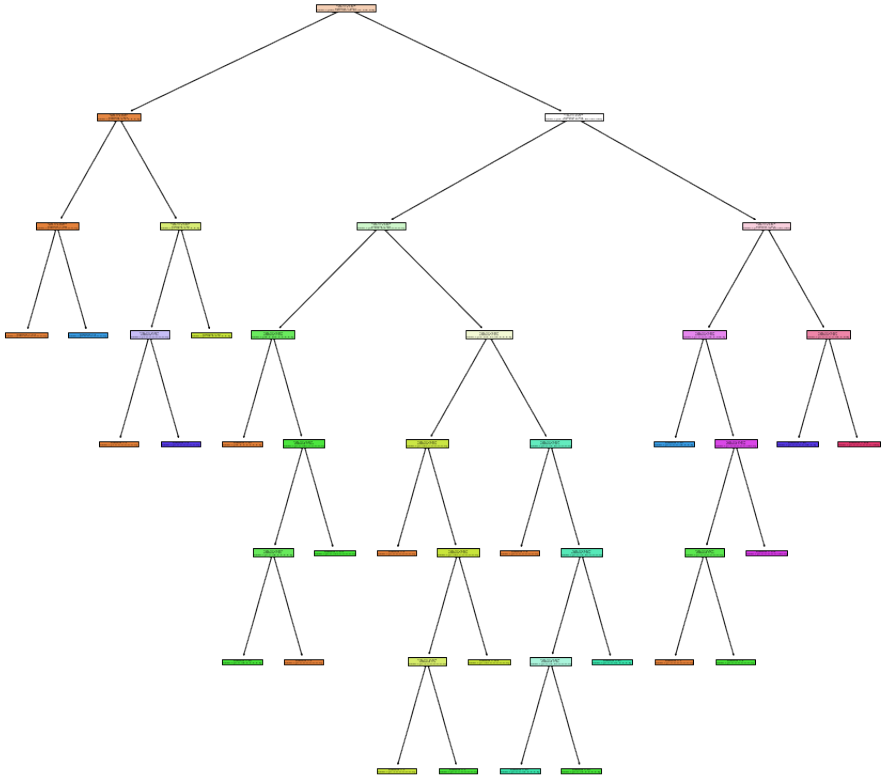

Fig. 4.8. Created decision tree.

classification. Decision trees also have a very high tendency to over-fit the training data, and thus would not perform well for the problem under consideration.

4.5. Random forest

An extension of the decision tree algorithm is the random forest. It involves constructing multiple decision trees at learning time and generating a class. Random forests are a solution to the decision tree problem with over-fitting the training data. An important advantage of random forests is that during model construction, the classification error estimator is automatically obtained as a byproduct of the learning sample selection algorithm for each tree.

Table 4.4. Decision tree algorithm's misclassifications.

Class	0_0_0	1_0_0	0_1_0	0_0_1	1_0_1	1_1_0	0_1_1	1_1_1	Total
Misclassifications	18	6	0	31	8	0	12	11	86

Random forests also provide more "smoothed" results than those obtained using a single classification tree. The difference is particularly noticeable on large datasets. This is because with random forests, the final decision is an averaging of multiple partial decisions from the different classification trees that comprise the forest.

A disadvantage of classification using random forests is its relative opacity. To some extent, a random forest resembles a black box: the user has no insight into the decision-making process, and the structure of the model is difficult to interpret because of its size and complexity. Since the final decision is the average of many independent partial decisions, there is usually no way to explain why the model's decision is the way it is.

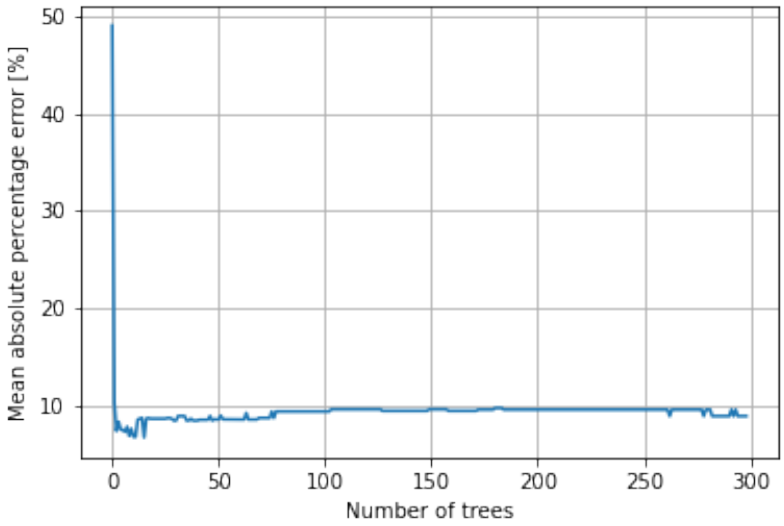

Fig. 4.9. Accurancy depending on the number of trees in the forest.

The algorithm was created using the RandomForestRegressor [35] classifier. The number of decision trees included in the random forest was chosen experimentally. This classifier returns a decision in numeric form, so in order to perform the classification, it was necessary to change the classes from strong to int type. On this basis, the absolute classification error was later calculated. Fig. 4.9 shows the mean absolute percentage error as a function of the number of created trees. It can be observed that the error value stabilizes at the level of about 10% when the number of trees is greater than 100. For this reason, 200 decision trees were used in the created algorithm. The trained algorithm correctly classified 90.44% of the test samples, which gives approximately 706 correct classifications. This result is similar

to previous algorithms, however, the runtime of the algorithm is significantly longer than that of previous methods.

4.6. SVM

Support Vector Machine (SVM) is an algorithm used for the classification and prediction of data. It is mainly based on selecting a hyperplane that best separates data from different classes. Thus, it is important to maximize the margin of separation between classes while maintaining the smallest classification error [36]. The distances between a hyperplane and its closest training data are the support vectors. Fig. 4.10 shows an example application of the algorithm, to split two sets. SVM allows the use of nonlinear hyperplanes, which allows the use of this method for problems with more than 2 classes, so it can be used to determine the occupancy of parking spaces.

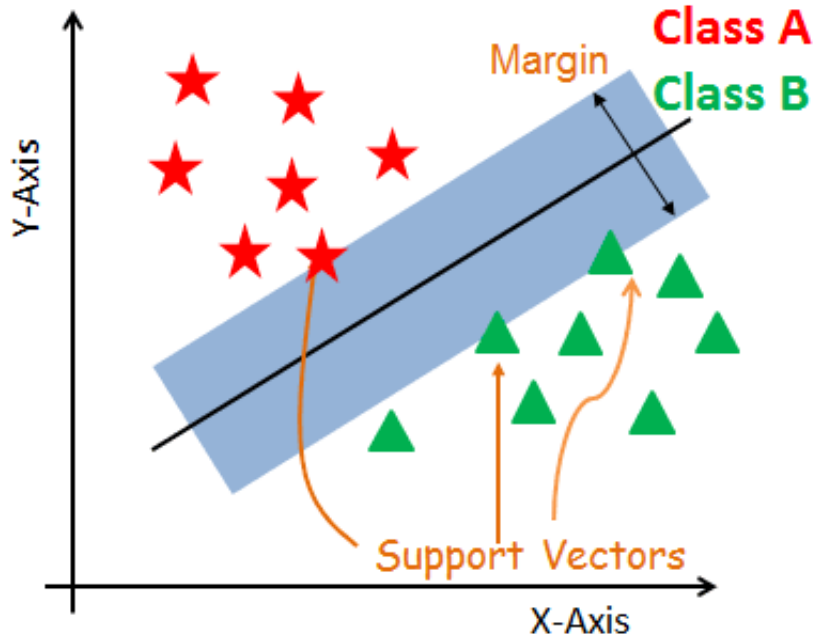

Fig. 4.10. Example of use SVM algorithm [37].

The classifier was created using the svm [38] function from the sklearn library. It allows choosing the kernel with which the algorithm should be initialized. Due to the dimensionality of the analyzed problem, radial basis function kernel (RBF), also called Gaussian kernel, was used. The classifier, created in this way, achieved 85.6% efficiency, which means correctly classifying 665 out of 778 observations. The misclassifications are presented in Table 4.5. They show that the algorithm performed worst when the vehicle was only in the middle (node 45) or only in the upper spot (node 46). Previous algorithms obtained better results. The problem of the solution may be an improperly chosen kernel, or difficulty in choosing the appropriate hyperplane for the analyzed problem.

Table 4.5. SVM algorithm's misclassifications.

Class	0_0_0	1_0_0	0_1_0	0_0_1	1_0_1	1_1_0	0_1_1	1_1_1	Total
Misclassifications	6	5	22	28	4	3	7	10	85

4.7. Multilayer Perceptron

A neural network is a term for a family of models, characterized by a wide parameter space and flexible structure. These models are developed based on how the human brain works. A neural network is similar to the brain in two respects [39]:

– the network obtains knowledge by learning,

– it uses the strength of connections between neurons, called synaptic weights, to store knowledge.

Neural networks are used in predictive applications, such as Multi-layer Perceptrons (MLP) and radial basis function(RBF), which are supervised in the sense that the results predicted by the model can be compared with known values of the predicted variables.

Perceptron is the simplest artificial neutron network. Its task is to classify an input vector x into one of two classes C_1 or C_2. If the input signal takes the value 1 then the vector x is classified into class C_1, if it takes the value 0 then into class C_2. The perceptron divides the N-dimensional space of input vectors into two half-spaces separated by a $N-1$ dimensional hyperplane. This hyperplane is called the decision boundary. If the feature space is two-dimensional then the boundary separating them is a straight line [40].

Multi-layer Perceptron is a network consisting of multiple layers of individual neurons (perceptrons), such that the outputs of the neurons of the previous layer form a vector fed to the input of each neuron of the next layer. The neurons of each layer always have the same number of inputs equal to the number of neurons of the previous layer +1. A schematic of the construction of a multilayer perceptron is shown in Fig. 4.11.

Multi-layer Perceptron is an algorithm that learns a classifier function from the training set received. From a set of observations $X = x_1, x_2, ..., x_n$ and an expectation value y, it learns a nonlinear approximating function for classification or regression.

A ready-made MLPClassifier [42] class that implements a multi-layer perceptron algorithm that trains using Backpropagation was used to create the algorithm. The neural network created was able to classify 93.06% of the observations correctly. Table 4.6 shows the misclassifications. As with the decision tree, the most errors were made in classifying the situation when the vehicle was only at the top spot (node 46). The repetition of this result in the second algorithm may suggest that the conclusion made earlier, is correct and the use of the vehicle that causes the greatest change in the earth's magnetic field, twice at the same location, may lead to misclassification.

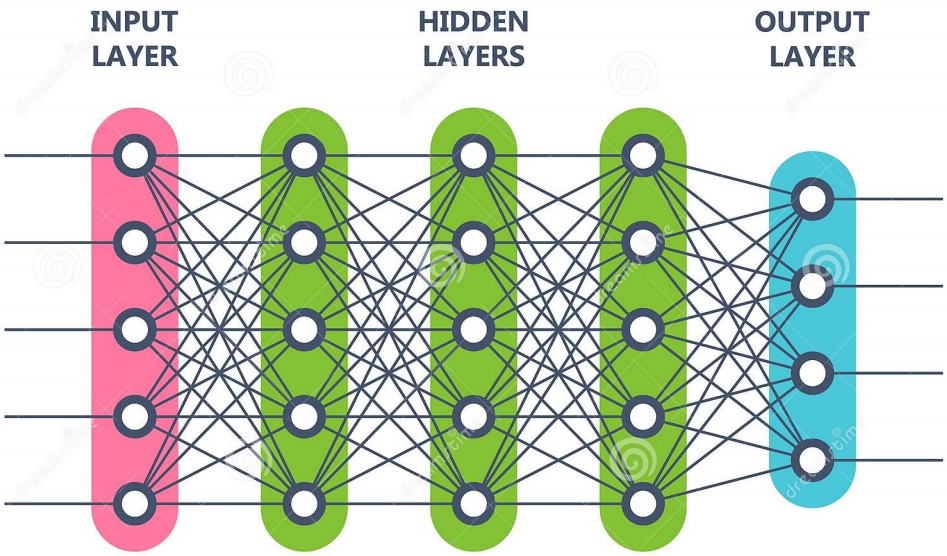

Fig. 4.11. Example of Multi-layer Perceptron with 3 hidden layers [41].

Table 4.6. MLP algorithm's misclassifications.

Class	0_0_0	1_0_0	0_1_0	0_0_1	1_0_1	1_1_0	0_1_1	1_1_1	Total
Misclassifications	4	9	5	36	0	0	0	0	54

The overall result is similar to the others, but the execution time of the algorithm is much longer than the previous ones (except for the random forest). Neural networks have a wide range of applications; however, it may be too advanced for the problem of parking space occupancy detection.

4.8. Algorithms comparison

Six algorithms were tested to select the most optimal one for the considered problem of classifying (deciding) whether parking spaces are occupied or not. All algorithms were trained and tested with the same datasets. The final two parameters of each algorithm, accuracy and execution time, were measured.

Accuracy was determined by comparing the prediction to the expected value of the test samples. For the Random Forest algorithm, the value was calculated based on the average absolute percentage error. The difference from the other algorithms is due to the fact that it was necessary to change the expectation value from string to a numerical value. In the other algorithms, accuracy was measured using the accuracy_score [43] function.

The execution time of the algorithm was determined by the average. Each algorithm was executed 1000 times, then the average value from one iteration was determined. In each iteration the algorithm operated on the same training and test data.

A summary of the average time of the algorithms and their accuracy is shown in Table 4.7. It can be observed that the performance of all the algorithms is satisfactory, none of them obtained a score lower

Table 4.7. Algorithms comparison.

Algorithm	Accuracy [%]	Average execution time [s]
kNN	98.33	0.101
Logistic regression	97.17	0.049
Decision tree	88.95	0.011
Random forest	90.44	1.319
SVM	89.08	0.317
MLP	93.06	2.433

than 85% efficiency. However, based on the results obtained, the most optimal one was selected for the problem under consideration.

The logistic regression algorithm was ruled out at the beginning due to its applicability only to dual problems. It is presented in the table for information purposes only. The decision tree algorithm achieved good results, however, it was rejected due to its susceptibility to over-fitting the training data. MLP orad Random forest algorithms also achieved decent scores, however, their complexity results in long execution times, which could cause long delays in system performance. Additionally, the problem under consideration is not complex enough for such advanced algorithms to be used.

Of the other two algorithms, kNN performed better than the SVM algorithm in both accuracy and execution time. For this reason, it was decided to implement the kNN algorithm on a real system. This algorithm is simple in its operation, but allows to achieve well classification performance. Using the SVM algorithm for the problem under consideration would not be a bad solution either, however it achieved slightly worse results during testing.

5. System verification

After selecting the most optimal machine learning algorithm for the problem under consideration, work began on its implementation, in order to operate in real-time. A python script was created to receive incoming data on the serial port. The received data frame was saved and analyzed. The algorithm at any given time stores the latest frame from each sensor. While receiving the data, it is checked if the value of module sum has changed, for any of the sensors. If so, the created kNN classifier makes a prediction about parking space occupancy, based on the most recent sample. This solution allows for a prediction close to real-time, with a delay due to sensor data being sent every 500ms, data being sent through the serial port, and calculations being performed by the algorithm.

5.1. Web server

In order to better visualize the processed data, Flask was used. It is a framework written in Python programming language, used to build web applications. The created server uses localhost, so currently, it is not possible to connect to it via the Internet, for this purpose a public IP would have to be assigned to the server.

5.1.1. Local server on flask

In the considered system only three parking spaces were used, therefore it was decided to store the data inside the program instead of creating a dedicated database. In the case of expanding the platform by additional space, the best solution would be to integrate the server with the database.

The created server uses localhost, its IP is 127.0.0.1 and port is 5000. The server supports several APIs, after sending a request to a predefined endpoint, it returns data assigned to this response. The created endpoints are shown in Table 5.1.

Two Flask objects are used for communication, with the data format as JSON. One is responsible for parking space occupancy data (Fig. 5.2) and the other is responsible for providing information about data from individual sensors (Fig. 5.1). The prediction made by the algorithm is sent as soon as it is executed, while the sensor information data is sent as soon as it is read from the serial port.

Table 5.1. Endpoints supported by the server.

Endpoint	Method	Description	Response
'/'	GET	Main web page with a description of a system, and picture from collecting data.	.html
'/smart_parking'	GET	Information about parking spots occupancy.	json
'/smart_parking'	POST	Parking availability data sent by the algorithm.	json
'/smart_parking/spot/<node_id>'	GET	Information about particular sensor data, node_id defines number of sensor.	json
'/smart_parking/spot/<node_id>'	POST	Information about sensor data published after readings from serial port.	json

```
dataToJson = {
    "nodeID": "0",
    "timeStamp": datetime.now(),
    "xAxis": "0",
    "yAxis": "0",
    "zAxis": "0",
    "battVoltage": "0",
    "boardVoltage": "0",
    "battCurrent": "0",
    "battTemp": "0",
    "radioRSSI": "0",
    "radioTransmitLevel":"0",
    "boostEnable": "0",
    "rstSrc": "0",
    "debug": "0"
}
```

Fig. 5.1. Json with sensor data.

```
occupancy = {
    "timeStamp": 0,
    "44": 0,
    "45": 0,
    "46": 0
}
```

Fig. 5.2. Json with occupancy data.

5.1.2. Web page

The finished server was verified using all 3 sensors. A phone was placed over the sensors in very close proximity, causing a field change similar to that produced by small vehicles. Logs from a working server are presented in Fig. 5.3. It shows all requests sent to the server and their status. Fig. 5.5 shows the main web page. It contains a redirect to information about parking availability. Returned information about parking spots availability is shown in Fig. 5.4, it also contains a timestamp. Detailed data from sensors is shown in Fig. 5.6.

5.2. System verification

The completed system, which allows real-time monitoring of parking space occupancy, was tested to verify proper detection. These tests were conducted both in laboratory conditions and in the conditions of an actual parking area. In both cases, the test consisted of changing the value of the Earth's magnetic field over a given sensor and verifying that the system's response was correctly updated.

5.2.1. Verification in a laboratory environment

The verification tests performed under laboratory conditions were done with a slightly altered sensors arrangement. They were placed 1m apart from each other (instead of 3.6m). The change in field values was caused by placing a mobile phone just above the sensor. When the phone is very close, the sensor readings are similar to those caused by lighter cars. These test assumptions are due to the limited test space and the lack of any other way to influence the magnetic field.

Table 5.2. Result of tests under laboratory conditions.

	0_0_0	1_0_0	0_1_0	0_0_1	1_0_1	1_1_0	0_1_1	1_1_1	Total
Number of tests	10	10	10	10	10	10	10	10	80
Correct classifications	10	6	10	8	9	10	9	10	72
Misclassifications	0	4	0	2	1	0	1	0	8
Accuracy [%]	100%	60%	100%	80%	90%	100%	90%	10%	90%

The results of the tests are presented in Table 5.2. It lists the events that should be classified into a particular class and the percentage of correct classifications for each case. From the data obtained, it can be seen that the created system performed poorly in classifying the situation when the device was located only in the lower position (Node 44). This may be because the created training set was not balanced well enough and the small number of observations from this class caused classification problems for the real system. The rest of the tests performed well, there were single misclassifications in many classes, but these may have been due to the presence of a large number of electronic devices in the environment.

However, taking into account all the tests the result of correct classification was 90%, this result is good enough that it was decided to test the performance of the system in real parking area conditions.

5.2.2. Verification in a real parking

Similar tests to the laboratory ones were performed when the system was verified as a real parking system. The sensors were arranged in the parking area as they were during the training data collection (Fig. 4.1). Unfortunately, due to technical reasons, 3 vehicles were not available to complete the tests. This resulted in the fact that only classification into 4 of the 8 classes was tested, as only one vehicle was available to complete the tests. Its technical data are shown in Table 5.3.

Table 5.3. Vehicles specifications.

Nr.	Model	Curb weight [kg]	Engine Capacity [cm^3]	Year of production
1.	Renault Megane	1168	1598	2000

The results of the tests performed in the real parking area are shown in Table 5.4. Similar to the tests under laboratory conditions, a significant classification error can be observed when parking only in the bottom spot (Node 44). The repetition of this error in real conditions implies that, in order to improve the performance of the algorithm, the training set should be expanded to include more observations of this event. The remaining all other events were correctly classified, this means that the kNN algorithm performs well for the problem under consideration, however with a better-balanced training set. Even with the large misclassification caused by Node 44, the total correctness of the tests performed was 87.5%, however, it is possible to increase this by expanding the training set. Also, final tests should be performed with 3 vehicles to test all possible events.

Table 5.4. Results of tests conducted under actual parking area conditions.

	0_0_0	1_0_0	0_1_0	0_0_1	1_0_1	1_1_0	0_1_1	1_1_1	Total
Number of tests	10	10	10	10	N/A	N/A	N/A	N/A	40
Correct classifications	10	5	10	10	N/A	N/A	N/A	N/A	35
Misclassifications	0	5	0	0	N/A	N/A	N/A	N/A	5
Accuracy [%]	100%	50%	100%	100%	N/A	N/A	N/A	N/A	87.5%

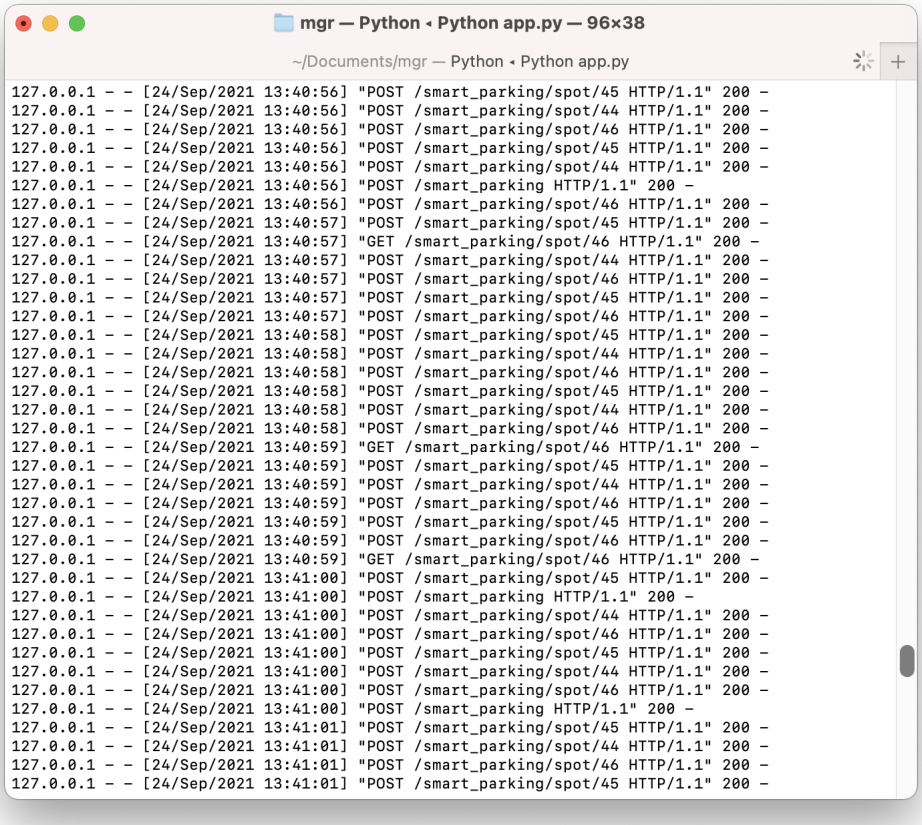

Fig. 5.3. Server logs.

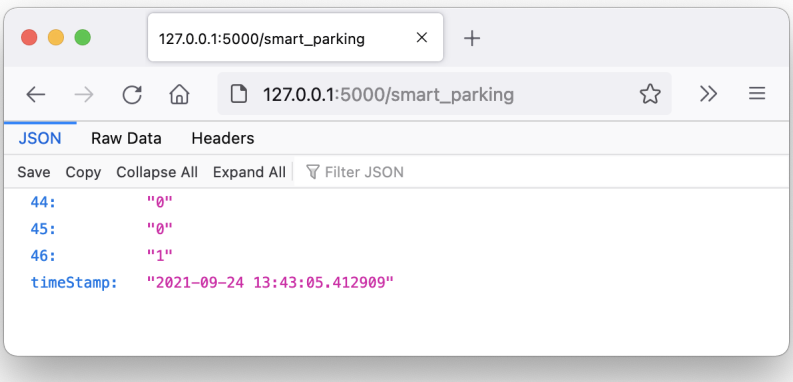

Fig. 5.4. Parking spots availability information.

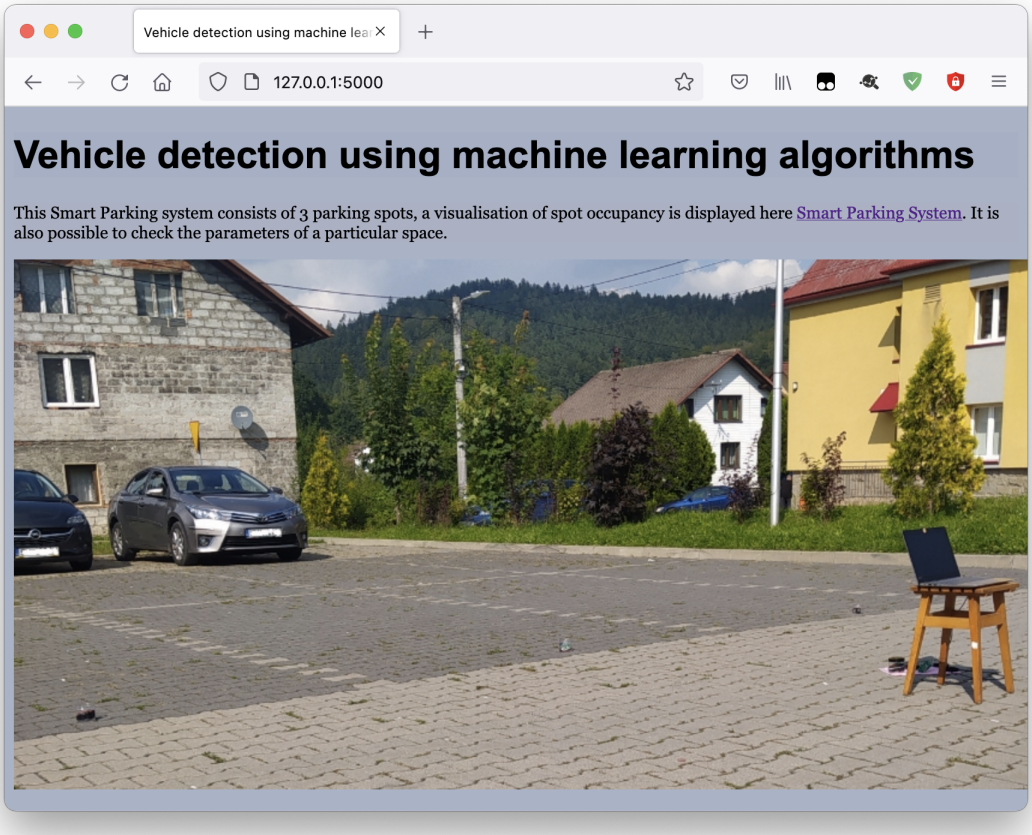

Fig. 5.5. Main web page.

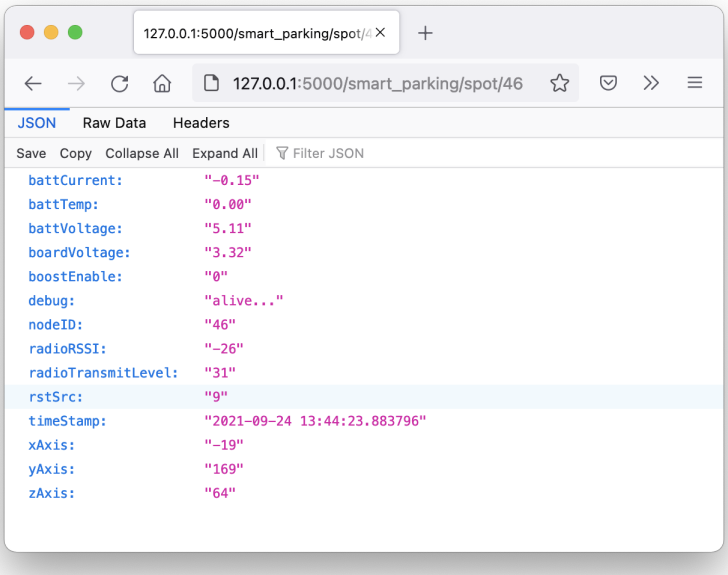

Fig. 5.6. Data from node 46 information.

6. Summary

This thesis examined the use of machine learning methods for efficient parking space occupancy detection, using magnetic field analysis. A system of three parking spaces was built for this purpose. The spaces were equipped with magnetic field sensors, integrated with a microcontroller. The microcontrollers connected via radio to the master, which was their default gateway. The microcontrollers handling the sensors and the master had ready code written in Arduino language in Sloeber environment, however it was modified for the purpose of this project. The master was communicating with the computer through the serial port. Initially, it was considered to use Raspberry Pi 4 as the computer unit, however, it was finally decided to use a portable laptop. The data received on the computer unit allowed for the collection of training and testing collection for machine learning algorithms. Six different algorithms were tested and based on the results the kNN algorithm was chosen as the most optimal one for the considered problem. After selecting the algorithm the implementation of the real system was started. A python script was created to read data from the serial port and predict the occupancy of parking spaces in real-time. In order to verify the system, a website was created using Flask framework and running on localhost. It allowed visualization of the data in the browser. Once the entire system was completed, it was tested under laboratory conditions and in a real parking lot, yielding efficiency results of 90% and 87.5%, respectively. The errors in the final tests were likely due to poor balancing of the training set, resulting in a large classification error for one class.

The next step to be taken to develop the platform is to expand the training set and run more tests in a real parking lot, this time using three vehicles to examine classifications for all classes. This was not accomplished, though, because of time limitations due to sensor problems. A delay in ordering the parts necessary to run them delayed the start of the work. Additionally, the Covid-19 virus pandemic situation made access to the university laboratory problematic. However, despite the problems encountered it was managed to create a system that meets all the requirements set out at the beginning.

The conducted research confirms that machine learning algorithms can be successfully applied to the problem of vehicle detection in parking spaces. They allow to eliminating interference from neighboring parking spaces in a good way. From the tested algorithms two that could be successfully used for this problem, they are kNN and SVM. Tests have shown that they work fast and effectively. However, a more balanced training set should be taken care of, which could allow for a significant increase in detection performance. It should also be updated on a regular basis, or automatically increased.

Magnetic field sensors in combination with machine learning algorithms are a good alternative to the problem of vehicle detection in parking spaces. Currently, systems based on image analysis or simple laser sensors are most commonly used. The first of these solutions is quite expensive for larger-scale deployments, while the second is often unreliable as it is very prone to detection errors. In the case of a Smart Parking-like system deployment, the use of magnetic field sensors would work well because it offers effective detection at a relatively low cost.

Several improvements are also possible in the developed system. A big problem was the limitation caused by radio communication. The size of one sent message allowed to send one data frame at a time. This caused the concept of operation of the communication of sensors with the master to be changed several times. Another problem with radio communication was the issue of communication breaking down when a large obstacle was placed between the sensor and the master. This necessitated a special setup of the sensors during training data collection and testing. An alternative to radio communication could be to use a Bluetooth module that is compatible with Arduino, and therefore supportable in the developed program code. This could allow for more stable communication and faster data transfer, which would result in lower latency of the system. Another alternative is to resign from wireless communication and connect sensors with master via cable, however a such solution would be less flexible and would require placing sensors in one location, and probably would cost more.

The created system determines the availability of parking spaces, however the use of magnetic sensors can have much greater application. With appropriately sensitive sensors it is possible to analyze all traffic within parking spaces, such as vehicles passing by. Another interesting concept is the recognition of entire parking events such as reverse parking, forward parking, or turning around in a parking space. In order to achieve such a result, it is necessary to analyze the received samples over time. Magnetic sensors could also be used for e.g. speed measurement, as it would be possible to measure the time it took to cover the distance between sensors A and B. Another interesting application in the Smart City aspect could be the measurement of the number of vehicles on the road, which would allow approximating the number of moving vehicles and to perform a traffic analysis in the city. The developed system can be therefore extended in several different directions, as there is a wide range of applications for it.

Bibliography

[1] P G Scholar Carlos Salazar Keyur K Patel Sunil M Patel. *Internet of Things-IOT: Definition, Characteristics, Architecture, Enabling Technologies, Application and Future Challenges*. 2016.

[2] *https://pl.freepik.com/premium-wektory/internet-of-things-icon-banner2401206.htm*.

[3] Jaiteg Singh Vidhi Gandhi. *IoT: Architecture, Technology, Applications, and Quality of Services*. 2019.

[4] Taewoo Namc J. Ramon Gil-Garcia Theresa A. Pardo. *What makes a city smart? Identifying core components and proposing an integrative and comprehensive conceptualization*. 2015.

[5] Emile Mardacany. *SMART CITY CHARACTERISTICS: Importance of Built Environment Components*.

[6] Departament of Economic and Social Affairs. *Word Urbanization Prospect*. Highlights. United Nations, 2018.

[7] Statista Research Department. *Registered cars in India FY 2017 by major city*. 2021.

[8] Ebru Karakose Ilhan Aydin Mehmet Karakose. *A Navigation and Reservation Based Smart Parking Platform Using Genetic Optimization for Smart Cities*. Computer Engineering Department, Civil Aviation School, Firat University, 2017.

[9] T K Ramesh Dharmini Kanteti D V S Srikar. *Intelligent Smart Parking Algorithm*. Bengaluru, India: Department od Electronics and Communication Engineering, Amrita University, 2017.

[10] Masahiko Mikawa Hiroyuki Suds Tatsuro Yano Takeshi Tsujimura. *Vehicle Distinction Using Laser Radar System*. Tottori, Japan, 2001.

[11] Lixiao S. Jianmin D. Kaihua Z. *Road and obstacle detection based on multi-layer laser radar in driverless car.* 2015.

[12] M. K. Muju S.Banerjee P. Choudekar. *Real time car parking system using image processing.* 2011.

[13] S. Syarif A. K. Jaya D. KManase Z. Zainuddin. *Car Detection in Roadside Parking for Smart Parking System Based on Image Processing.* 2020.

[14] K. Choeychuen. *Available car parking space detection from webcam by using adaptive mixing features.* 2012.

[15] Péter Sarcevic. “New methods in the application of inertial and magnetic sensors in online pattern recognition problems”. Ph.D. Thesis. UNIVERSITY OF SZEGED, 2018.

[16] Lijun Suna Yao Hea Yuchuan Dua. “Vehicle Classification Method Based on Single-Point Magnetic Sensor”. type. Shanghai, China: Tongji University, 2012.

[17] Xinghe Bao Fengrong Li Chang Xu Yingguan Wang. “Vehicle Classification Using an Imbalanced Dataset Based on a Single Magnetic Sensor”. Article. Shanghai, China: Chinese Academy of Sciences, 2018.

[18] Mindaugas Zilys Darius Andriukaitis Algimantas Valinevicius Mindaugas Cepenas ytautas Markevicius Dangirutis Navikas. “Dynamic Vehicle Detection via the Use of Magnetic Field Sensors”. Article. Kaunas, Lithuania: Kaunas University of Technology, 2016.

[19] Ke Zhu Chao Zheng Liu Xuyang K.H. Lam. “Overview of Spintronic Sensors, Internet of Things, and Smart Living”. article. The University of Hong Kong, 2016.

[20] F. Yuw S. Qiu H. Zhuand. “A remote test method for parking detection system based on magnetic wireless sensor network.” Article. 2017.

[21] V. Sinatra C. Vacirca E. Rossino L. Palermo S. Kurukunda C. Trigona B. Andò and S. Baglio. “Implementation and Characterization of a Smart Parking System based on 3-axis Magnetic Sensors.” Article. 2016.

[22] K. Gong B. Li M. Meng S. Song Q. Zhang G. Wang. “Wireless magnetic sensor node for vehicle detection using finite element simulation.” Article. Hong Kong, China, 2013.

[23] Susmita Ray. “A Quick Review of Machine Learning Algorithms”. Article. Faridabad, India: Manav Rachna University, 2019.

[24] Alan Davis Babu. *Artificial Intelligence vs Machine Learning vs Deep Learning (AI vs ML vs DL)*. 2019. URL: *https://medium.com/@alanb_73111/artificial-intelligence-vs-machine-learning-vs-deep-learning-ai-vs-ml-vs-dl-e6afb7177436*.

[25] Łukasz Więckowski. “Moduł czujnika zajętości miejsca postojowego/ Modułu stacji bazowej”. Dokumentacja Techniczna. Krakow, Poland, 2020.

[26] *8.5AH Lithium battery*. URL: *https://www.tme.eu/pl/details/eve-er26500_pr/baterie/eve-battery-co/er26500-fl/*.

[27] *19 AH extended lithium battery (D/R20 size)*. URL: *https://www.tme.eu/pl/details/eve-er34615_fl/baterie/eve-battery-co/er-34615-fl/*.

[28] *RFM69 Library*. URL: *https://github.com/lowpowerlab/RFM69*.

[29] *CmdMessenger*. URL: *https://github.com/thijse/Arduino-CmdMessenger*.

[30] Joanna Jaworek-Korjakowska. “Klasyfikatory w uczeniu maszynowym”. Lecture. Krakow, Poland: WEAIIB, Katedra Automatyki i Robotyki.

[31] *sklearn.neighbors.KNeighborsClassifier*. URL: *https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html*.

[32] *sklearn.linear_model.LogisticRegression*. URL: *https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html*.

[33] *Decision Tree Vector Art*. URL: *https://www.vecteezy.com/free-vector/decision-tree*.

[34] *sklearn.tree.DecisionTreeClassifier*. URL: *https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html*.

[35] *sklearn.ensemble.RandomForestRegressor*. URL: *https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html*.

[36] Agnieszka Duraj. "WYKRYWANIE WYJĄTKÓW PRZY UŻYCIU WEKTORÓW NOŚNYCH". Article. Instytut Informatyki, Politechnika Łódzka, 2017.

[37] *Support Vector Machines with Scikit-learn*. URL: *https://www.datacamp.com/community/tutorials/svm-classification-scikit-learn-python*.

[38] *sklearn.svm.SVC*. URL: *https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html*.

[39] IBM SPSS Statistics. *Czym jest siec neuronowa?* URL: *https://www.ibm.com/docs/pl/spss-statistics/27.0.0?topic=networks-what-is-neural-network*.

[40] Adrian Brückner Paweł Błaszczyk. *Podstawowe modele sieci neuronowych*. URL: *http://books.icse.us.edu.pl/runestone/static/ai/SztuczneSieciNeuronowe/PodstawoweModeleSieciNeuronowych.html#perceptron-wielowarstwowy*.

[41] *Multi level neural network. Artificial intelligence concept. Computer neuron net. Logical scheme of a ai perception. Vector illustration.* URL: *https://www.dreamstime.com/neural-network-artificial-intelligence-concept-computer-neuron-net-multi-level-neural-network-artificial-intelligence-concept-image133841766*.

[42] *sklearn.neural_network.MLPClassifier*. URL: *https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html*.

[43] *sklearn.metrics.accuracy_score*. URL: *https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html*.

List of Figures

List of Tables

A. Appendix

A.1. Master node

Main function of program uploaded to master node. The whole project is attached in Dejw_RM3100_644p_GatewayM2.zip.

```
void loop()
{

        //Serial.println("test3");
        wdt_reset();

        if (radio.receiveDone())
        {
                //Serial.println("test1");
                if(radio.DATALEN == sizeof (sensorDataFrame)) //sensor
                {
                        //Serial.println("test2");

                        theSensorData = *(sensorDataFrame*)radio.DATA;

                        // theSensorData = *(sensorDataFrame*)radio.DATA;

                        battVoltage = theSensorData.battVoltage /100.0;
                        boardVoltage = theSensorData.boardVoltage /1000.0;
                        battCurrent = theSensorData.battCurrent /100.0;
                        battTemp = theSensorData.battTemp /100.0 ;

                        //battTemp = ((theSensorData.battTemp) - 65535)/100.0 ;

                        //Serial.print("CarSensor no: ");
                        //44;0;4;12;9;3.54;3.32;1.40;-17.09;-51;31;0;1;alive...
                        Serial.print(theSensorData.timeStamp);
                        Serial.print(";");
                        Serial.print(theSensorData.nodeId);
                        Serial.print(";");
                        Serial.print(theSensorData.carFlag);
                        Serial.print(";");
```

```
                Serial.print(theSensorData.data0);
                Serial.print(";");
                Serial.print(theSensorData.data1);
                Serial.print(";");
                Serial.print(theSensorData.data2);
                Serial.print(";");
                Serial.print(battVoltage,2);
                Serial.print(";");
                Serial.print(boardVoltage,2);
                Serial.print(";");
                Serial.print(battCurrent,2);
                Serial.print(";");
                Serial.print(battTemp,2);
                Serial.print(";");
                Serial.print(radio.RSSI);
                Serial.print(";");
                Serial.print(theSensorData.radioTransmitLevel);
                Serial.print(";");
                Serial.print(theSensorData.boostEnable);
                Serial.print(";");
                Serial.print(theSensorData.rstSrc);
                Serial.print(";");
                Serial.println((char*)theSensorData.debug);

                if (radio.SENDERID  == nodeId)
                {
                        keepAlive = theSensorData.sensorStatus;
                        if (keepAlive)
                        {
                                isRecieve = true;
                        }
                }
        }
        if (radio.ACK_REQUESTED)
        {
                radio.sendACK();
                //Serial.print("DBG;ACK sent...");0
        }
        //Serial.println();
        Blink(LED,5);
}

if(startUpCalibrate){
        for(int i=0;i<sensorsNumber;i++){

                radio.send(sensorID[i],&startUpCalibrate,sizeof(bool));
                delay(100);
```

```
                }
                startUpCalibrate = false;
        }


        for(int i=0;i<sensorsNumber;i++){
                currPeriod[i] = (millis()+(i+1)*2000)/8000;
                if (currPeriod[i] != lastPeriodSend[i])
                {
                        lastPeriodSend[i]=currPeriod[i];
                        timeSync = millis();
                        radio.send(sensorID[i],&timeSync,sizeof(unsigned long));
                        delay(10);
                }
        }


        /* wait for keep alive packiet*/
        if (sth2Send && isRecieve)
        {
                int currPeriod = millis()/1000;
                if (currPeriod != lastPeriod)
                {
                        lastPeriod = currPeriod;
                        if (count <= 4)
                        {
                        //delay(10);
                        Serial.println("DBG;Transmit new config data...");
                        radio.send(theConfigData.nodeId,
                        (const void*)(&theConfigData), sizeof(theConfigData));
                        }

                        if (count > 4)
                        {
                                sth2Send = false;
                                isRecieve = false;
                                count = 0;
                        }

                        count++;
                }
                Blink(LED,5);
        }
        cmdMessenger.feedinSerialData();
}
```

A.2. Slave node

Main function of program uploaded to sensors. The whole project is attached in Dejw_RM3100_644p_GatewayM2.zip.

```
void loop()
{
        wdt_reset();

        if (counter >= theEepromData.sensorSleepMux)
        {
                if(ctr >= 1000) //ms
                {
                        counter = 0;
                        ctr = 0;
                }
                else
                {
                        ctr++;
                }
                delay(1);

                if (radio.receiveDone())
                {
                if (radio.TARGETID  == (uint16_t)theEepromData.nodeId)
                {
                //Blink(LED, 3);
                        if (radio.DATALEN == sizeof (sensorConfigFrame))
                        {
                                theConfigData = *(sensorConfigFrame*)radio.DATA;
                                if (theConfigData.cmd == SET_CFG) // Set new configuration
                                {
                                theEepromData.sensorThreshold=theConfigData.sensorThreshold;
                                theEepromData.sensorCyclesCount=theConfigData.sensorCounter;
                                theEepromData.sensorSleepTime=theConfigData.sensorSleepTime;
                                theEepromData.sensorSleepMux=theConfigData.sensorSleepMux;

                                #ifdef DEBUG
                                Serial.println("Receive new Configuration: ");
                                Serial.print("Threshold: ");
                                Serial.println(theEepromData.sensorThreshold);
                                Serial.print("Dump cycles: ");
                                Serial.println(theEepromData.sensorCyclesCount);
                                Serial.print("Sleep time: ");
                                Serial.println(theEepromData.sensorSleepTime);
                                Serial.print("Sleep mux: ");
                                Serial.println(theEepromData.sensorSleepMux);
```

```
        delay(50);
        #endif
        // Write config data to  EEPROM
        eeprom_write_block((void*)&theEepromData,
        (void*)0, sizeof(sensorEepromData));
        // Calibrate
        carCounter = 0;
        counter = 0;
        ctr = 0;
        calibrateSensor();
        for (int i = 0; i < 10; i++)
        {
                digitalWrite(LED_PIN, !digitalRead(LED_PIN));
                delay(30);
        }
        digitalWrite(LED_PIN, LOW);
        }
else if (theConfigData.cmd == GET_CFG)
{
        theConfigData.cmd = 0;
        radio.send(theEepromData.gatewayId,
        (const void*)(&theConfigData), sizeof(theConfigData));
}
}
else if(radio.DATALEN == sizeof (sensorDataFrame)) //sensor
{
/*theSensorData = *(sensorDataFrame*)radio.DATA;
if (theSensorData.cmd == GET_CFG) // Get current sensor data
{
                        ;
}
}
else if(radio.DATALEN == sizeof(unsigned long)){
masterTime = *(unsigned long*)radio.DATA;
slaveTime = millis();
//theSensorData.timeStamp = masterTime+millis()-slaveTime;
//radio.send(theEepromData.gatewayId,
(const void*)(&theSensorData), sizeof(sensorDataFrame));

}else if(radio.DATALEN == sizeof(bool)){
setup();
calibrateSensor();
calibrated = true;
}
else
{
;// data is not valid
```

```
                }
        }
        if (radio.ACKRequested())
        {
                radio.sendACK();
        }
        // CheckForWirelessHEX(radio, flash, false);
}
}
else
{
        checkCompassState();

        //goToSleep(true);
}


if(calibrated){
        int currPeriod1 = millis()/500;

        if (currPeriod1 != lastPeriod1)
        {
                lastPeriod1=currPeriod1;
                checkCompassState();
                theSensorData.timeStamp = masterTime+millis()-slaveTime;

                radio.send(theEepromData.gatewayId,
                (const void*)(&theSensorData), sizeof(sensorDataFrame));

                delay(100);
        }
}
Serial.print("ID: ");
Serial.println(theEepromData.nodeId);

/*int currPeriod2 = millis()/100;
if (currPeriod2 != lastPeriod2)
{
        lastPeriod2=currPeriod2;
        checkCompassState();
}*/
if(mfpPinTriggered)
{
        mfpPinTriggered = false;
}

}
```

A.3. Algorithm selection

A python script for selecting the most optimal algorithm. The posted code is located in the algorithm_selection.ipynb file.

```
# load data

import pandas as pd
from sklearn import datasets
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
import time
from sklearn import preprocessing

import numpy as np

path = r'C:\Users\adria\pomiary'

header = ['Module-44','Power-44','Module-45','Power-45','Module-46',
'Power-46','Value']

df = pd.read_csv(path+"/Data.csv", sep=";",header=None,names=header)
df=pd.DataFrame(df)
data = df.iloc[: , :-1]
target = df['Value']
features_train, features_test, labels_train, labels_test =
train_test_split(data,target, train_size=0.7,random_state=1)

scaler_train = preprocessing.StandardScaler().fit(features_train)
scaler_test = preprocessing.StandardScaler().fit(features_test)

features_train = scaler_train.transform(features_train)
features_test = scaler_test.transform(features_test)

#output char to int for random forest
from sklearn.preprocessing import LabelEncoder

enc = LabelEncoder()
enc2 = LabelEncoder()

enc.fit(labels_train)
enc2.fit(labels_train)

randomForest_labels_train = enc.transform(labels_train)+1
randomForest_labels_test = enc.transform(labels_test)+1
```

```
test_Number = 100

print(features_train)

# kNN

# k choose

import matplotlib.pyplot as plt
#plt.subplots(figsize=(10,8))
plt.grid()
k = []
for i in range(1,12):

start = time.time()
classifier = KNeighborsClassifier(n_neighbors=i)
classifier.fit(features_train, labels_train)
clas_pred = classifier.predict(features_test)
end = time.time()
accurancy = accuracy_score(labels_test, clas_pred)
k.append(accurancy)
plt.plot(k)
plt.xlabel("Value of k")
plt.ylabel("Accurancy")
plt.savefig('kChoose.png')


kNN_time = 0;
for i in range(1,test_Number):
start = time.time()

classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(features_train, labels_train)
clas_pred = classifier.predict(features_test)

end = time.time()

kNN_time = kNN_time + end - start
kNN_time = kNN_time/i
kNN_accuracy = round(accuracy_score(labels_test, clas_pred) * 100,2)

print("Accuracy:",kNN_accuracy)
print("Time:",kNN_time)

i = 0
state = [0,0,0,0,0,0,0,0]
```

```
for x in range(0,len(clas_pred)):

if clas_pred[x] != np.array(labels_test)[x]:
i=i+1
print("nr. ",i,".",clas_pred[x],np.array(labels_test)[x])

if(np.array(labels_test)[x]=='0_0_0'):
state[0] = state[0] + 1
elif(np.array(labels_test)[x]=='1_0_0'):
state[1] = state[1] + 1
elif(np.array(labels_test)[x]=='0_1_0'):
state[2] = state[2] + 1
elif(np.array(labels_test)[x]=='0_0_1'):
state[3] = state[3] + 1
elif(np.array(labels_test)[x]=='1_0_1'):
state[4] = state[4] + 1
elif(np.array(labels_test)[x]=='1_1_0'):
state[5] = state[5] + 1
elif(np.array(labels_test)[x]=='0_1_1'):
state[6] = state[6] + 1
elif(np.array(labels_test)[x]=='1_1_1'):
state[7] = state[7] + 1

print(i+kNN_accuracy*778/100)
print(state)

logTrain = []
for x in range(0,len(labels_train)):
#print(labels_test.iloc[x][2])
logTrain.append(labels_train.iloc[x][2])
#print(logTrain)

logTest = []
for x in range(0,len(labels_test)):
#print(labels_test.iloc[x][2])
logTest.append(labels_test.iloc[x][2])
#print(logTest)

# logistic regression

from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression

logReg_time = 0;
for i in range(1,test_Number):
start = time.time()
```

```
clf = LogisticRegression(random_state=0).fit(features_train, logTrain)
s = clf.predict(features_test)

end = time.time()

logReg_time = logReg_time + end - start
logReg_time = logReg_time/i
logReg_accuracy = round(accuracy_score(logTest, s)*100,3)

print("Accuracy:",logReg_accuracy)
print("Time:",logReg_time)

# decision tree

import pandas as pd
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
from sklearn.model_selection import train_test_split # Import train_test_split function
from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation

dTree_time = 0
for i in range(1,test_Number):
start = time.time()

clf = DecisionTreeClassifier(random_state=1)

# Train Decision Tree Classifer
clf = clf.fit(features_train,labels_train)

#Predict the response for test dataset
y_pred = clf.predict(features_test)

end = time.time()
dTree_time = dTree_time + end - start
dTree_time = dTree_time/i
dTree_accuracy = round(accuracy_score(labels_test, y_pred)*100,3)

print("Accuracy:",dTree_accuracy)
print("Time:",dTree_time)

i = 0
state = [0,0,0,0,0,0,0,0]

for x in range(0,len(y_pred)):

if y_pred[x] != np.array(labels_test)[x]:
i=i+1
print("nr. ",i,".",y_pred[x],np.array(labels_test)[x])
```

```
if(np.array(labels_test)[x]=='0_0_0'):
state[0] = state[0] + 1
elif(np.array(labels_test)[x]=='1_0_0'):
state[1] = state[1] + 1
elif(np.array(labels_test)[x]=='0_1_0'):
state[2] = state[2] + 1
elif(np.array(labels_test)[x]=='0_0_1'):
state[3] = state[3] + 1
elif(np.array(labels_test)[x]=='1_0_1'):
state[4] = state[4] + 1
elif(np.array(labels_test)[x]=='1_1_0'):
state[5] = state[5] + 1
elif(np.array(labels_test)[x]=='0_1_1'):
state[6] = state[6] + 1
elif(np.array(labels_test)[x]=='1_1_1'):
state[7] = state[7] + 1

print(i+dTree_accuracy*778/100)
print(state)

clf.get_depth()

clf.get_n_leaves()

from sklearn import tree
fig = plt.figure(figsize=(15,15))
_ = tree.plot_tree(clf,
filled=True)
fig.savefig("decistion_tree.png")

k = []
for i in range(1,10):
start = time.time()

clf = DecisionTreeClassifier(random_state=1,max_depth=i)

# Train Decision Tree Classifer
clf = clf.fit(features_train,labels_train)

#Predict the response for test dataset
y_pred = clf.predict(features_test)
accurancy = accuracy_score(labels_test, y_pred)
k.append(accurancy)
plt.plot(k)
plt.grid()
plt.xlabel("Value of k")
```

```
plt.ylabel("Accurancy")

# random forest

from sklearn.ensemble import RandomForestRegressor
import numpy as np

RF_time = 0
for i in range(1,test_Number):
start = time.time()

rf = RandomForestRegressor(n_estimators = 200, random_state = 1)
rf.fit(features_train, randomForest_labels_train);
predictions = rf.predict(features_test)

# Calculate the absolute errors
errors=abs(np.array(predictions.astype(int))-np.array(randomForest_labels_test.astype(int)))

# Calculate mean absolute percentage error (MAPE)
mape = 100 * (errors / randomForest_labels_test)

end = time.time()
RF_time = RF_time +end - start
RF_time = RF_time/i
RF_accuracy = round(100 - np.mean(mape),3)

# Calculate and display accuracy
print('Accuracy:', RF_accuracy)
print("Time:",RF_time)

k = []
for i in range(1,5):
rf = RandomForestRegressor(n_estimators = i, random_state = 1)
rf.fit(features_train, randomForest_labels_train);
predictions = rf.predict(features_test)

#Predict the response for test dataset
# Calculate the absolute errors
errors=abs(np.array(predictions.astype(int))-np.array(randomForest_labels_test.astype(int)))
# Calculate mean absolute percentage error (MAPE)
mape = 100 * (errors / randomForest_labels_test)
accurancy = round(np.mean(mape),3)
k.append(accurancy)



plt.plot(k)
```

```
plt.grid()
plt.xlabel("Number of trees")
plt.ylabel("Mean absolute percentage error [%]")
plt.savefig('randomForestN.png')

# SVM

from sklearn import svm
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

print(features_train[:,0])
df2s = pd.DataFrame({'col1': features_train[:,0],
        'col2': features_train[:,2],
        'col3': features_train[:,4]})

df3s = pd.DataFrame({'col1': features_test[:,0],
        'col2': features_test[:,2],
        'col3': features_test[:,4]})

print(df2s)

SVM_time = 0
for i in range(1,test_Number):
start = time.time()

#Create a svm Classifier
clf = svm.SVC(kernel='rbf',random_state=1) # Linear Kernel

#Train the model using the training sets
clf.fit(df2s, labels_train)

#Predict the response for test dataset
y_pred = clf.predict(df3s)

end = time.time()
SVM_time = SVM_time + end - start

SVM_time = SVM_time/i
SVM_accuracy = round(accuracy_score(labels_test, y_pred)*100,3)

print("Accuracy:",SVM_accuracy)
print("Time:",SVM_time)

i = 0
state = [0,0,0,0,0,0,0,0]
```

```
for x in range(0,len(y_pred)):

if y_pred[x] != np.array(labels_test)[x]:
i=i+1
print("nr. ",i,".",y_pred[x],np.array(labels_test)[x])

if(np.array(labels_test)[x]=='0_0_0'):
state[0] = state[0] + 1
elif(np.array(labels_test)[x]=='1_0_0'):
state[1] = state[1] + 1
elif(np.array(labels_test)[x]=='0_1_0'):
state[2] = state[2] + 1
elif(np.array(labels_test)[x]=='0_0_1'):
state[3] = state[3] + 1
elif(np.array(labels_test)[x]=='1_0_1'):
state[4] = state[4] + 1
elif(np.array(labels_test)[x]=='1_1_0'):
state[5] = state[5] + 1
elif(np.array(labels_test)[x]=='0_1_1'):
state[6] = state[6] + 1
elif(np.array(labels_test)[x]=='1_1_1'):
state[7] = state[7] + 1

print(i+SVM_accuracy*778/100)
print(state)

#  multi-layer perceptron (MLP) algorithm

from sklearn.neural_network import MLPClassifier

MLP_time = 0
for i in range(1,test_Number):
start = time.time()

clf = MLPClassifier(solver='lbfgs', random_state=1)
clf = clf.fit(features_train,labels_train)
y_pred = clf.predict(features_test)

end = time.time()
MLP_time = MLP_time + end - start
MLP_time = MLP_time/i
MLP_accuracy = round(accuracy_score(labels_test, y_pred)*100,3)

print("Accuracy:",MLP_accuracy)
print("Time:",MLP_time)

i = 0
```

```
state = [0,0,0,0,0,0,0,0]

for x in range(0,len(y_pred)):

if y_pred[x] != np.array(labels_test)[x]:
i=i+1
print("nr. ",i,".",y_pred[x],np.array(labels_test)[x])

if(np.array(labels_test)[x]=='0_0_0'):
state[0] = state[0] + 1
elif(np.array(labels_test)[x]=='1_0_0'):
state[1] = state[1] + 1
elif(np.array(labels_test)[x]=='0_1_0'):
state[2] = state[2] + 1
elif(np.array(labels_test)[x]=='0_0_1'):
state[3] = state[3] + 1
elif(np.array(labels_test)[x]=='1_0_1'):
state[4] = state[4] + 1
elif(np.array(labels_test)[x]=='1_1_0'):
state[5] = state[5] + 1
elif(np.array(labels_test)[x]=='0_1_1'):
state[6] = state[6] + 1
elif(np.array(labels_test)[x]=='1_1_1'):
state[7] = state[7] + 1

print(i+MLP_accuracy*778/100)
print(state)

results_data = {'Accuracy':[kNN_accuracy,logReg_accuracy,
        dTree_accuracy,RF_accuracy,SVM_accuracy,MLP_accuracy],
        'Average Time':[kNN_time,logReg_time,dTree_time,RF_time,SVM_time,MLP_time]}

results = pd.DataFrame(results_data, index =
['kNN', 'Logistic regression','decision tree', 'random forest', 'SVM','MLP'])

print(results)
```

A.4. kNN algorithm

An implementation of the kNN algorithm that reads the collected data from a serial post and sends it to a flask server. The posted code is located in the engine.py file.

```
import serial
import time
import pandas as pd
import numpy as np
```

```
from datetime import datetime
import json
import requests

from sklearn import datasets
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import preprocessing

def determineOccupancy(x):
x_scaled = scaler_train.transform(np.array(x).reshape(1, -1))
clas_pred = classifier.predict(x_scaled)
return clas_pred

path = r'C:\Users\adria\pomiary'

header = ['Module-44','Power-44','Module-45','Power-45','Module-46','Power-46','Value']

df = pd.read_csv("./Data.csv", sep=";",header=None,names=header)
df=pd.DataFrame(df)
data = df.iloc[: , :-1]
target = df['Value']
features_train, features_test, labels_train, labels_test =
train_test_split(data,target, train_size=0.99,random_state=1)
print(features_test)

scaler_train = preprocessing.StandardScaler().fit(features_train)
scaler_test = preprocessing.StandardScaler().fit(features_test)

features_train = scaler_train.transform(features_train)
features_test = scaler_test.transform(features_test)

classifier = KNeighborsClassifier(n_neighbors=4)
classifier.fit(features_train, labels_train)

ser = serial.Serial('/dev/cu.usbserial-A100NJNH', baudrate=9600, timeout=1)

for i in range(1,10):
data = ser.readline()

new_data = [0,1,2,3,4,5]
old_data = [0,0,0,0,0]

dataToJson = {
        "nodeID": "0",
        "timeStamp": datetime.now(),
```

```
        "xAxis": "0",
        "yAxis": "0",
        "zAxis": "0",
        "battVoltage": "0",
        "boardVoltage": "0",
        "battCurrent": "0",
        "battTemp": "0",
        "radioRSSI": "0",
        "radioTransmitLevel":"0",
        "boostEnable": "0",
        "rstSrc": "0",
        "debug": "0"
}


while(1):
data = ser.readline().strip()
data1 = data.decode().split(';')
dataToJson["nodeID"] = data1[1]
dataToJson["timeStamp"] = datetime.now()
dataToJson["xAxis"] = data1[3]
dataToJson["yAxis"] = data1[4]
dataToJson["zAxis"] = data1[5]
dataToJson["battVoltage"] = data1[6]
dataToJson["boardVoltage"] = data1[7]
dataToJson["battCurrent"] = data1[8]
dataToJson["battTemp"] = data1[9]
dataToJson["radioRSSI"] = data1[10]
dataToJson["radioTransmitLevel"] = data1[11]
dataToJson["boostEnable"] = data1[12]
dataToJson["rstSrc"] = data1[13]
dataToJson["debug"] = data1[14]

res = requests.post('http://localhost:5000
/smart_parking/spot/'+dataToJson["nodeID"],
json=json.dumps(dataToJson,default=str))

if(data1[1] == "44" and new_data[0]!= (abs(float(data1[3]))
+ abs(float(data1[4])) + abs(float(data1[5])))):
new_data[0] = abs(float(data1[3])) + abs(float(data1[4]))
+ abs(float(data1[5]))
new_data[1] = float(data1[3])**2 + float(data1[4])**2 + float(data1[5])**2
parkingOccupancu = determineOccupancy(new_data)
parkingOccupancu = parkingOccupancu[0].split('_')
dJson={"timeStamp":datetime.now(),"44":parkingOccupancu[0],
        "45":parkingOccupancu[1],"46":parkingOccupancu[2]}
res = requests.post('http://localhost:5000/smart_parking',
```

```
json=json.dumps(dJson,default=str))
print(new_data)
print(parkingOccupancu)
elif(data1[1] == "45" and new_data[2]!= (abs(float(data1[3]))
+ abs(float(data1[4])) + abs(float(data1[5])))):
new_data[2] = abs(float(data1[3])) + abs(float(data1[4]))
+ abs(float(data1[5]))
new_data[3] = float(data1[3]) ** 2 + float(data1[4]) ** 2 + float(data1[5]) ** 2
parkingOccupancu = determineOccupancy(new_data)
parkingOccupancu = parkingOccupancu[0].split('_')
dJson={"timeStamp":datetime.now(),"44":parkingOccupancu[0],
        "45":parkingOccupancu[1],"46":parkingOccupancu[2]}
res = requests.post('http://localhost:5000/smart_parking',
json=json.dumps(dJson,default=str))
print(new_data)
print(parkingOccupancu)
elif (data1[1] == "46" and new_data[4]!= (abs(float(data1[3]))
+ abs(float(data1[4])) + abs(float(data1[5])))):
new_data[4] = abs(float(data1[3])) + abs(float(data1[4]))
+ abs(float(data1[5]))
new_data[5] = float(data1[3]) ** 2 + float(data1[4]) ** 2
+ float(data1[5]) ** 2
parkingOccupancu = determineOccupancy(new_data)
parkingOccupancu = parkingOccupancu[0].split('_')
dJson={"timeStamp":datetime.now(),"44":parkingOccupancu[0],
        "45":parkingOccupancu[1],"46":parkingOccupancu[2]}
res = requests.post('http://localhost:5000/smart_parking',
json=json.dumps(dJson,default=str))
print(new_data)
print(parkingOccupancu)

ser.close()
```

A.5. Flask server

mplementation of a server running on localhost using the Flask framework. The posted code is located in the app.py file.

```
from flask import Flask, redirect, url_for, render_template, request, jsonify
import json

app = Flask(__name__)

dataFromSensors = {
        "44": {
                "nodeID":"44",
```

```
                "timeStamp": 0,
                "xAxis": "0",
                "yAxis": "0",
                "zAxis": "0",
                "battVoltage": "0",
                "boardVoltage": "0",
                "battCurrent": "0",
                "battTemp": "0",
                "radioRSSI": "0",
                "radioTransmitLevel":"0",
                "boostEnable": "0",
                "rstSrc": "0",
                "debug": "0"
        },
        "45": {
                "nodeID":"45",
                "timeStamp": 0,
                "xAxis": "0",
                "yAxis": "0",
                "zAxis": "0",
                "battVoltage": "0",
                "boardVoltage": "0",
                "battCurrent": "0",
                "battTemp": "0",
                "radioRSSI": "0",
                "radioTransmitLevel":"0",
                "boostEnable": "0",
                "rstSrc": "0",
                "debug": "0"
        },
        "46": {
                "nodeID":"46",
                "timeStamp": 0,
                "xAxis": "0",
                "yAxis": "0",
                "zAxis": "0",
                "battVoltage": "0",
                "boardVoltage": "0",
                "battCurrent": "0",
                "battTemp": "0",
                "radioRSSI": "0",
                "radioTransmitLevel":"0",
                "boostEnable": "0",
                "rstSrc": "0",
                "debug": "0"
        }
}
```

```
dataFromSensors = json.dumps(dataFromSensors)

dataFromSensors = json.loads(dataFromSensors)

occupancy = {
        "timeStamp": 0,
        "44": 0,
        "45": 0,
        "46": 0
}

occupancy = json.dumps(occupancy)
occupancy = json.loads(occupancy)
@app.route('/')
def index():
return render_template('mainPage.html')

@app.route('/smart_parking', methods=['GET'])
def smart_parkingGET():
return jsonify(occupancy)

@app.route('/smart_parking', methods=['POST'])
def smart_parking():
content = json.loads(request.json)
for x in content:
occupancy[str(x)] = content[x]
return jsonify(request.json)

@app.route('/smart_parking/spot/<node_id>',methods=['GET'])
def spotGET(node_id):
return jsonify(dataFromSensors[str(node_id)])


@app.route('/smart_parking/spot/<node_id>', methods=['POST'])
def spot(node_id):
content = json.loads(request.json)
for x in content:
dataFromSensors[str(node_id)][x] = content[x]

return jsonify({"node_id":node_id})

if __name__ == '__main__':
app.run(debug=True, host='0.0.0.0')
```

A.6. Main page

Main page used by flask server. The posted code is located in the mainPage.html file.

```
<!DOCTYPE html>
<html>
<head>
<!-- HTML Codes by Quackit.com -->
<title>
Vehicle detection using machine learning algorithms</title>
<meta name="viewport" content="width=device-width, initial-scale=1">
<meta name="keywords" content="smart parking, vehicle detection, machine learning">
<meta name="description" content="This page is a visualization of
a test parking system created for my master thesis">
<style>
body {background-color:#a9b2c7;background-image:url
        (blob:https://kleki.com/a63efc5b-f317-554d-a538-ae1216a57b31);
        background-repeat:no-repeat;background-position:top center;
        background-attachment:fixed;}
h1{font-family:Arial, sans-serif;color:#000000;background-color:#a4afc5;}

p {font-family:Georgia, serif;font-size:14px;font-style:normal;font-weight:
        normal;color:#000000;background-color:#aab1c5;}
</style>
</head>
<body>
<h1>Vehicle detection using machine learning algorithms</h1>
<p>This Smart Parking system consists of 3 parking spots, a visualisation of
spot occupancy is displayed here <a href="/smart_parking">Smart Parking System</a>.
It is also possible to check the parameters of a particular space. </p>
<p></p>
<img src=" {{url_for('static', filename='parkingSystem.png')}}">

</body>
</html>
```