World Academy of Science, Engineering and Technology
International Journal of Mechanical and Industrial Engineering
Vol:15, No:9, 2021

An E-Maintenance IoT Sensor Node Designed for Fleets of Diverse Heavy-Duty Vehicles

George Charkoftakis, Panagiotis Liosatos, Nicolas-Alexander Tatlas, Dimitrios Goustouridis, Stelios M. Potirakis

Abstract**—**E-maintenance is a relatively recent concept, generally referring to maintenance management by monitoring assets over the Internet. One of the key links in the chain of an e-maintenance system is data acquisition and transmission. Specifically for the case of a fleet of heavy-duty vehicles, where the main challenge is the diversity of the vehicles and vehicle-embedded self-diagnostic/reporting technologies, the design of the data acquisition and transmission unit is a demanding task. This is clear if one takes into account that a heavy-vehicles fleet assortment may range from vehicles with only a limited number of analog sensors monitored by dashboard light indicators and gauges to vehicles with plethora of sensors monitored by a vehicle computer producing digital reporting. The present work proposes an adaptable internet of things (IoT) sensor node that is capable of addressing this challenge. The proposed sensor node architecture is based on the increasingly popular single-board computer – expansion boards approach. In the proposed solution, the expansion boards undertake the tasks of position identification, cellular connectivity, connectivity to the vehicle computer, and connectivity to analog and digital sensors by means of a specially targeted design of expansion board. Specifically, the latter offers a number of adaptability features to cope with the diverse sensor types employed in different vehicles. In standard mode, the IoT sensor node communicates to the data center through cellular network, transmitting all digital/digitized sensor data, IoT device identity and position. Moreover, the proposed IoT sensor node offers connectivity, through WiFi and an appropriate application, to smart phones or tablets allowing the registration of additional vehicle- and driver-specific information and these data are also forwarded to the data center. All control and communication tasks of the IoT sensor node are performed by dedicated firmware.

Keywords**—**IoT sensor nodes, e-maintenance, single-board computers, sensor expansion boards, on-board diagnostics.

I. Introduction

THE management of a fleet of heavy-duty vehicles/ machinery, both in the day-to-day and long-term scale, is a quite challenging task. Usually, such fleets comprise hundreds or even thousands of members, characterized by a wide range of types, age, technologies etc. Moreover, each fleet member's condition depends on operators' competence and operating conditions (e.g., environmental temperatures, sand, sea). The conventional strategy of periodic maintenance, i.e., regular maintenance according to a specific schedule (e.g., upon completion of a specific number of working hours of the vehicle/machinery) is not time- and cost-efficient, while there is no alarm of a possible upcoming breakdown [1].

E-maintenance is a multidisciplinary field developing during the last two decades [2]. Despite the fact that different definitions have been proposed in the literature [3], [4], a generic enough one is that e-maintenance refers to the "maintenance management concept whereby assets are monitored and managed over the Internet" [3]. It should also be stressed that e-maintenance is considered to comply with Industry 4.0, offering an effective as well as efficient assets' management by means of applying mechanical, electrical, and computer engineering in a cooperative way [5], [6].

An e-maintenance solution for the demanding case of diverse heavy-vehicles fleet is the "InteligentLogger" system [7]. A fundamental component of InteligentLogger is the network of IoT sensor nodes that undertake the acquisition of vehicle/machinery-related data in real time, which are appropriately processed by artificial intelligence (AI) models in order to detect abnormal behaviors, which in turn feed business intelligence (BI) models for the production of reports, dashboard and alerts [7].

The present work presents a versatile IoT sensor node that was specifically designed for InteligentLogger. A number of works focusing on IoT sensors for vehicle applications have already been published, some of them are focusing on smart vehicles, e.g., [8]-[11], while others are focusing on fleet management, e.g., [1], [12]. However, the proposed IoT sensor node differentiates from other, already proposed, vehicle-oriented IoT sensors/sensor nodes because of its dedicated design aimed to serve fleets of heavy-duty vehicles/machinery of high diversity. It is capable of acquiring information from a wide range of possible sources such as on-board diagnostics (OBD), analog and digital sensors, as well as of providing position identification using the global navigation satellite system (GNSS), and is characterized by versatility, a quality essential for addressing the problem of being appropriate for all members of a highly diverse heavy-duty vehicles fleet.

The rest of the paper is organized as follows: Section II provides a brief presentation of the InteligentLogger system, while it also provides the specifications of the IoT sensor node; Section III presents the system level design of the proposed IoT sensor node; Section IV presents details of the hardware design; Section V presents details of the software design.

II. InteligentLogger Description: Specifications for the IoT Sensor Node

InteligentLogger implements a bottom-up architecture,

G. Charkoftakis, P. Liosatos, N.-A. Tatlas, and S. M. Potirakis are with the Electrical and Electronics Engineering Department of the University of West Attica, Ancient Olive Grove Campus, 250 Thivon and P. Ralli, Aigaleo, Athens GR-12244, Greece (e-mails: ntatlas@uniwa.gr, spoti@uniwa.gr).

D. Goustouridis is with the Thetametrisis SA, Christou Lada 40, Peristeri, Athens, GR-12132, Greece (e-mail: dgoustouridis@thetametrisis.com).

which can be briefly described as follows [7]:

- The IoT sensor node in each equipment acquires real time information directly from the machine and transmits them to the cloud ‘Sensor Consolidation Service’ (SCS).
- In case of lack of infrastructure networks at the worksite, a specific android application (that can run on any android device) downloads the IoT sensor node data and uploads them to the cloud as soon as the android device is moved to a location that a network is available. Furthermore, the android application can be used to register data that cannot be automatically recorded.
- The SCS collects the following data: device ID, timestamp, longitude, latitude and up to 8 analog and 3 digital sensors readings, depending on each sensor node configuration. It filters out invalid records and measurements and outputs equipment information and its operating data.
- These data are temporarily stored and then fed to a machine learning (ML) model for fault detection.
- Also, they are feed to a geofencing algorithm to identify cases of theft or inappropriate use.
- All results from both systems are feed to the data warehouse (DWH).
- The DWH is used as a data source to the BI suite, which finally produces corporate reports, worksite reports and sensor alerts.

As apparent, InteligentLogger relies on the IoT sensor nodes for the reliable acquisition of key information from the assets. In this sense, the IoT sensor nodes form the “frontline” in solving the problem of managing a fleet of wide variety. A key issue in successfully addressing this complicated problem is the careful selection of the main connectivity specifications as well as the minimum information that should be available from each heavy-duty vehicle/machinery.

TABLE I
MINIMUM SET OF EQUIPMENT INFORMATION AND OPERATING DATA ACQUIRED BY ANY IOT SENSOR NODE OF INTELIGENTLOGGER

Object	Note
GNSS position	{LAT, LON} Taken from GPS module
Time Stamp	{TIME, DATE} Included in every sample, taken from GPS module or NTP
Vehicle ID	{Vehicle ID} Set by the user
Battery Voltage	{Sensor Data} Taken from sensors connected to the IoT device through the Expansion Board Interface
Hydraulic Pressure	
Engine Cooling Lubricant Temperature and Pressure	
Throttle voltage	
Fuel Tank Level	
Fuel Flow Control	
Generic Sensor	
Vehicle Speed	{Sensor Data/Alarms} Taken from the OBD
Vehicle RPM	
Engine/Coolant Temperature	
Fault Alarms	
Generic Sensor	

The InteligentLogger IoT sensor node should meet the following requirements:

- GNSS position acquisition.
- Connectivity to OBD for sensor data/alarm acquisition.
- Cellular connectivity by means of 3G/long-term evolution (LTE) modem for the transmission of recorded data.
- WiFi connectivity to mobile devices equipped with an appropriate android application for transferring (on demand) the IoT sensor node recordings.
- On-board temperature/relative humidity sensor.
- Eight analog plus three digital sensor channels for direct connection to vehicle/machinery sensors.
- The minimum set of equipment information and operating data indications that each InteligentLogger IoT sensor node should record in real time are presented in Table I.

III. GENERAL DESCRIPTION OF THE PROPOSED SOLUTION

The proposed IoT sensor node comprises two subsystems: the Main Control Unit (MCU) and the Expansion Board Interface (EBI). Fig. 1 shows the general block diagram of the InteligentLogger IoT sensor node.

The EBI implements an eight-channel analog sensors acquisition system, providing with the appropriate matching circuits for 8 analog sensors, as well as the necessary signal conditioning, filtering, ending-up to the analog to digital conversion (A/D). Moreover, EBI provides connectivity to 3 digital sensors, as well as a built-in temperature/relative humidity sensor. The standard configuration of the 8 analog channels includes 4 voltage sensors and 4 current loop sensors 4-20 mA (2-wires). The versatility of the proposed IoT sensor node lies both in offering a variety of sensor type interfaces (analog voltage, analog current loop, digital), as well as in the ability to implement a wide variety of connected sensor configurations, with minimal modifications to the analog printed circuit. A basic condition for the smooth operation of the system is that for each connected analog sensor the proper external power supply has been ensured and its calibration table has been registered in the digital system. The connection between the EBI and the MCU is achieved through a serial I2C protocol interface.

The MCU is based on the very popular single-board computer Raspberry Pi. The MCU performs the required control, storage and management of the receiving data provided by the EBI. In standard mode, the MCU communicates with the SCS through a 4G/LTE modem, transmitting all digital/digitized sensor data, IoT device identity and position. Moreover, it offers connectivity to OBD of the heavy-duty vehicles as also WiFi connectivity to mobile devices equipped with an appropriate application for the manual registration of information about the vehicle and the driver that cannot be automatically acquired, as well as for the on demand local downloading of IoT sensor node recordings, e.g., in cases that cellular network is not available at the worksite. Dedicated firmware performs all communication and control tasks of the MCU. It is programmed with a high-level language (Python) on top of a modern operating system (Linux).

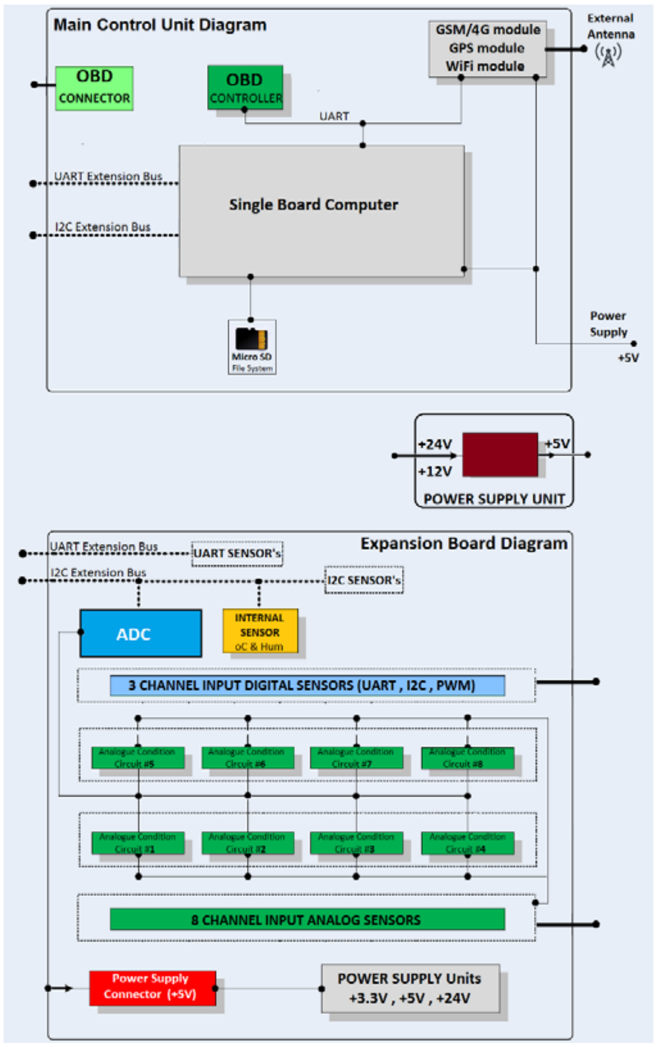

Main Control Unit Diagram
GSM/4G module
GPS module
WiFi module
External Antenna
OBD CONNECTOR
OBD CONTROLLER
UART
UART Extension Bus
I2C Extension Bus
Single Board Computer
Micro SD File System
Power Supply
+5V
+24V
+12V
+5V
POWER SUPPLY UNIT
UART Extension Bus
UART SENSOR's
I2C Extension Bus
I2C SENSOR's
Expansion Board Diagram
ADC
INTERNAL SENSOR oC & Hum
3 CHANNEL INPUT DIGITAL SENSORS (UART , I2C , PWM)
Analogue Condition Circuit #5
Analogue Condition Circuit #6
Analogue Condition Circuit #7
Analogue Condition Circuit #8
Analogue Condition Circuit #1
Analogue Condition Circuit #2
Analogue Condition Circuit #3
Analogue Condition Circuit #4
8 CHANNEL INPUT ANALOG SENSORS
Power Supply Connector (+5V)
POWER SUPPLY Units +3.3V , +5V , +24V

Fig. 1 General block diagram of the InteligentLogger IoT sensor node system

Both subsystems are powered by + 12 V or + 24 V, which is supplied via the heavy-vehicle /machinery battery.

The developed prototype of the InteligentLogger IoT sensor node is illustrated in Fig. 2.

IV. Hardware Design Description

The internal configuration of MCU is shown in Fig. 3 and includes a Raspberry Pi 3 board [13] equipped with an AutoPi shield board [14].

The main reason for choosing Raspberry Pi [14] compared to other microcomputers and micro-controllers, is that the specific computer is a complete, low cost, and low power "pocket" computer. It comprises a 64-bit quad core processor running at 1.4 GHz, dual-band 2.4 GHz and 5 GHz wireless LAN, Bluetooth 4.2/BLE, faster Ethernet and an electronic hardware connectivity via several available communication protocols (UART, I2C SPI). Thanks to the built-in programmable microcontroller with the Extended 40-pin GPIO header accessible, it forms a great development tool for integrated automation systems.

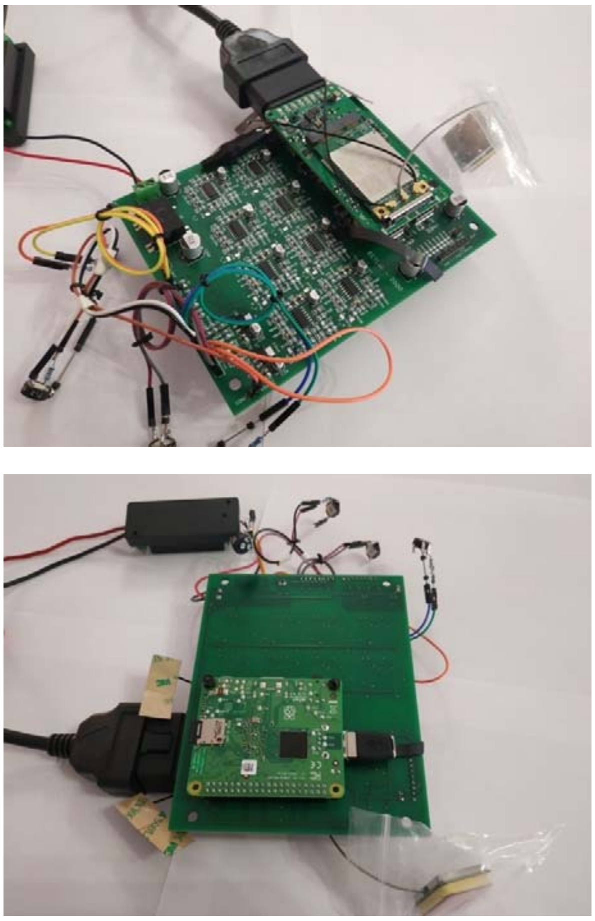


Fig. 2 Top and bottom view of the prototype of an InteligentLogger IoT sensor node

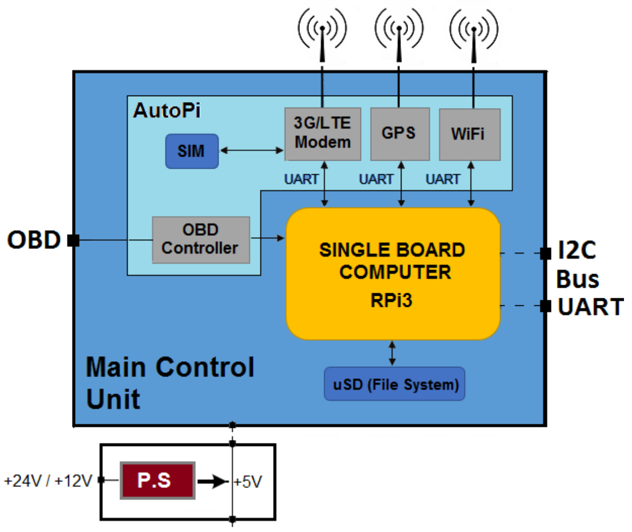


Fig. 3 Block diagram of the MCU subsystem

AutoPi [14] integrates a range of electronic circuits and sensors that adapt the MCU to the needs of a mobile data collection unit. AutoPi is a solution based on Raspberry Pi 3 (RPi3) technology, but it also uses several other high-tech tools that grant users access to valuable functions. AutoPi integrates a number of sensors that are needed for a vehicle tracking and telemetry application. The high speed 4G/LTE modem and the GNSS (GPS) are a single module while there is a popular ELM327 controller for OBD interface. The Wi-Fi is provided by RPi3 that boots from a microSD card. AutoPi is interfaced with RPi3 by a single USB port that provides a number of virtual communication ports per peripheral. AutoPi is providing all the necessary glue logic and power adaptation circuits for its peripherals, thus only single USB supply is needed.

The EBI (Fig. 4) offers 8 analog sensors input channels: 4 current sensor interface channels (channels no. 1 to no. 4) and 4 voltage sensor interface channels (channels no. 5 to no. 8). The sensor signal of each channel reaches the A/D through the following 4 stages (Fig. 5): matching circuit, level shifting, pre-amplification, anti-aliasing filter.

A/D converts the signal into a suitable digital format that can be received and processed by the microprocessor of the MCU. Two constant current source circuits are available whenever required (optional use). In addition, 3 digital output sensors can be routed to the MCU (channels no. 9 to 11), using one of the following communication protocols: UART or I2C. One digital sensor is already mounted on board of the EBI to measure temperature and relative humidity in the IoT sensor node.

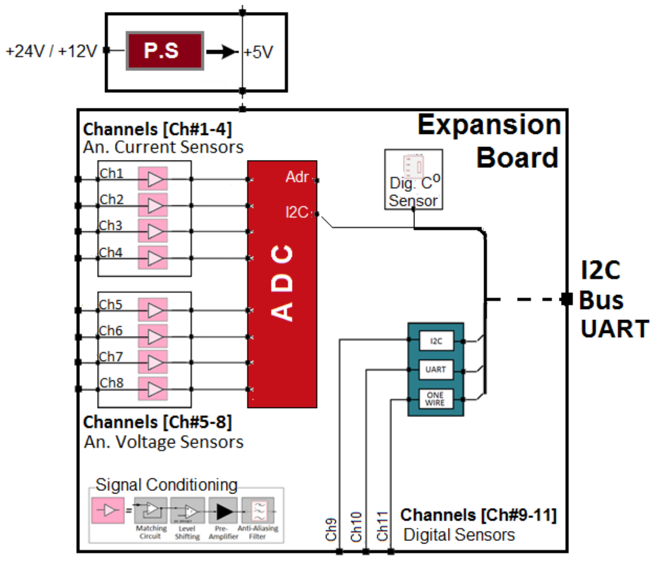


Fig. 4 Block diagram of the EBI subsystem

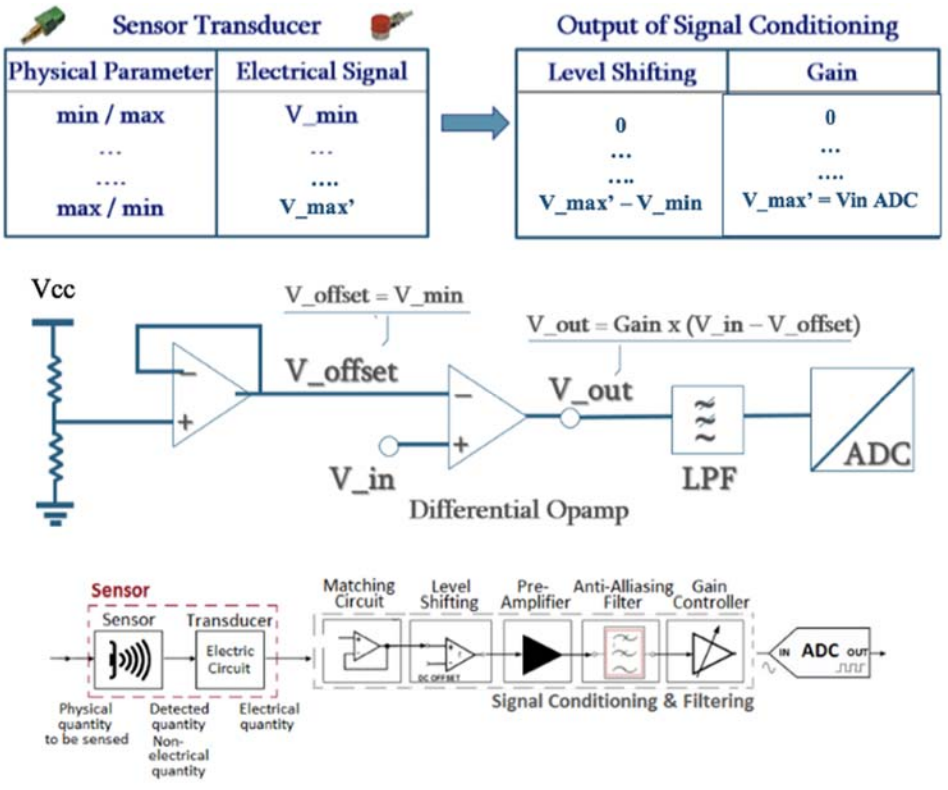


Fig. 5 Block diagram of analog signal digitization circuitry (one channel)

The InteligentLogger IoT sensor node system is powered from the battery of the heavy vehicle (+12 V or +24 V dc). On board there are also two dc-to-dc converters: one to provide the main power supply of +5 V/2 A of the EBI and a second one to provide +24 V/ 125 mA, required for the driving circuits of the 4-20 mA current sensors.

The voltage divider in Fig. 5 creates the required V_{offset}. This voltage corresponds to the minimum input voltage (V_{offset} = V_{min}) received by the analog sensor and which must be removed. The level shifting and pre-amplifier stages are based on an operational amplifier used as a differential amplifier with adjustment gain. The output voltage of the analog circuit (V_{out}) is equal to the voltage received by the sensor (V_{in}), minus V_{offset}, multiplied by the appropriate gain: V_{out} = (V_{in} - V_{offset}) x Gain).

This output voltage should not exceed the maximum input voltage specified by the A/D. In Fig. 5 the first operational amplifier is used as a buffer to completely isolate the V_{offset} voltage regulation from the rest of the circuit, while the last stage of the analog circuit, before the A/D converter, is a third-order Butterworth Low Pass Filter.

The proposed IoT sensor node offers a number of adaptability features, with the ability to interconnect and drive a number of analog output sensors for heavy-duty vehicles, based on appropriate zero-ohm resistor placement and appropriate value selection for limited number of passive components of the signal conditioning circuit of each channel of EBI.

V. Software Design Description

The MCU (Fig. 3) is using a number of wired plus wireless interfaces. The wired interfaces are I2C plus USB interfaces. The wireless interfaces are used to transmit data using LTE or Wi-Fi. A number of virtual UART interfaces are created over USB in order to interface with the GNSS subsystem and the OBD controller. A virtual QMI over USB interface is used to connect to the high speed 4G/LTE modem. The heart of the computing system is the RPi3 single board computer running Linux Ubuntu distribution. The RPi3/Linux ecosystems allow virtually any type of high-level server/application running using any external interface or peripheral. This versatility and plethora of options made RPi3/Linux combined with AutoPi the best option for our application.

The easy to use and popular Python version 3 with a number of libraries (Table II) was the best programming choice.

TABLE II
Python Libraries Used

Library	Usage
pyserial	Generic UART interface
pynmea2	NMEA-0183 protocol handling (GNSS)
sqlite3	sqlite3 interfacing
ADCDifferentialPi	MCP3424 interfacing (ADC)
python-obd	ELM327 interfacing (OBD)
request	HTTP/POST/REST interfacing

The combined use of a high-level operation system (Linux) with a versatile language (Python 3) allowed us to:

- Use existing drivers for all peripherals. Almost every peripheral has a driver for Linux OS.
- Use existing libraries for high level protocols. Python is a programming language that is notorious for the enormous pool of readily available libraries and support base.
- Debug and maintain everything with ease, since text file debug logs and printed messages could be combined with exception errors.
- Simplify maintenance and upgrade by using remote access through Wi-Fi or 4G/LTE. A dynamic DNS service with OpenVPN and SSH is integrated to allow secure access even from remote locations.

The GNSS subsystem is using the GPS position system with the standard NMEA-0183 protocol. This protocol is an ASCII protocol over UART that needs to be parsed in order to obtain machine to machine interaction, meaning that the longitude and latitude values should be obtained as floating numbers to be communicated to the SCS interface. The A/D's are using the I2C subsystem that needs a specialized library (ADCDifferentialPi). The OBD is also using a library (python-obd) that interacts with the vehicle on board computer, specialized OBD codes are issued to get responses. The OBD is interfaced using the popular ELM327 IC that is using a UART interface with RPi3. The python-obd library allows a generic approach to almost any vehicle type and low-level physical interface. Each vehicle supports a number of "common" OBD commands and a number of "custom" OBD commands that are usually undocumented. The "common" OBD commands have initially be used in our application to acquire the following parameters:

- Vehicle speed
- Vehicle engine rpm
- Coolant temperature
- Diagnostic Trouble Codes (DTC).

The trouble codes provide an important maintenance information. To ease software development an OBD emulator was used that could also emulate a number of simultaneous DTC codes.

Finally high-level libraries such as sqlite3 were used to maintain a local record database. The intuition behind this functionality is simple, we assume that there is not always connection with the SCS. This can be for long times in case that the vehicle operates in remote and network-isolated locations. In this case all measurements are stored in a local database and recalled when there is a valid SCS connection or on demand by a mobile device running an appropriate android application.

At first a number of Python scripts were developed, one script per peripheral. Each peripheral operation could be readily checked for proper operation. These scripts were also the building stones for the main application. During the initialization of the application, the proper operation of each peripheral could be verified. For the wireless communication interfaces, proper driver interfaces and network routing were verified. The Linux is implementing a full featured TCP/IP stack with full networking capabilities. Python can exploit these capabilities to build a full-blown network application. The developed software application has the following features:

- Wireless communication with Wi-Fi and 4G/LTE interfaces.
- Parsing and accessing the positioning information obtained by the GNSS subsystem.
- Acquiring sensor information from the A/D's interface.
- Interfacing with the OBD ELM327 IC, acquiring operating parameters and fault codes.
- Storing each measurement as a record in a local database (SQLite)
- Storing diagnosing information in case of sensor or subsystem malfunction.
- Storing each record in a structured manner so that it can be acquired by a machine-to-machine interface.
- Being able to keep connection with SCS. If there is no

connection the records are kept until safely transmitted.

- Keeping connection with SCS and sending records using a high-level protocol based on HTTP/POST/REST API.

The flow of the IoT sensor node software application is shown in Fig. 6. For convenience, the flow is shown for 2 sensors only. At first an initialization phase is performed; the sensors and their interfaces are initialized to their operating parameters. Any existing malfunctions are detected and are recorded as faults for transmission to the SCS. Next the sensors are sampled in a periodic manner according to their specification. The sampling time is configurable. Every 15 seconds an RMC message is received from the GNSS, this message contains the longitude, latitude, speed and validity of the information. Every 60 seconds the information from the A/D's, Temperature/Humidity sensor, Digital I/O and OBD is acquired.

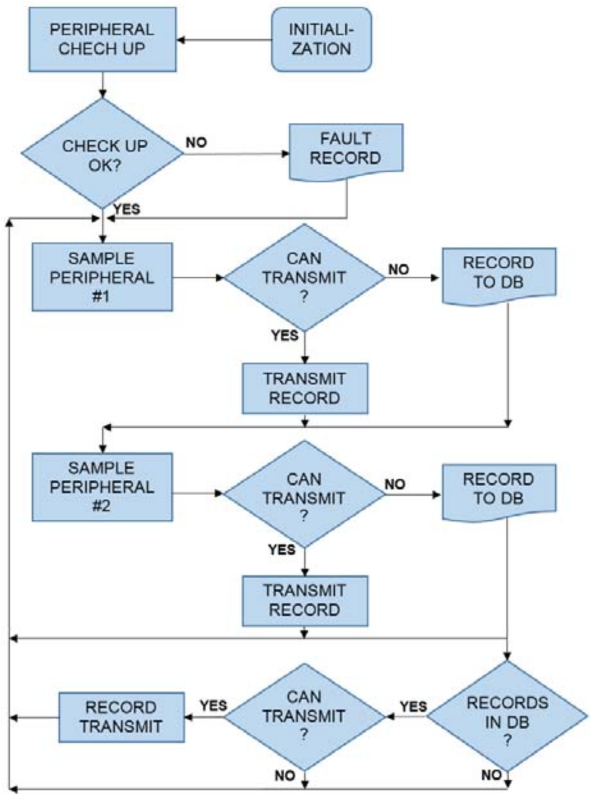
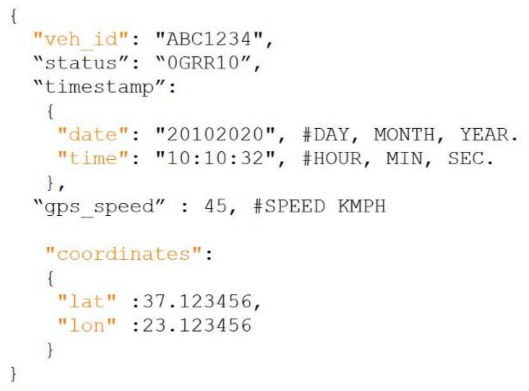


Fig. 6 Software application flow

After each sensor sampling the system checks the SCS connection, if it is valid the record is transmitted immediately, if not it is written in the local database. Periodically the system checks if there are records in the database, these are records that have not been transmitted. Then a transmission is attempted using the oldest record first. The database handling is performed by using SQL high level commands, thus record manipulation can include searching, adding and deleting.

The transmission to SCS is done with cloud messaging by using the *azure.servicebus* protocol. This messaging protocol includes also the confirmation that the record is successfully delivered to the server. If confirmation is received the record should not be written to the database if it is just acquired by a sensor or it should be deleted from the database if it is retrieved for transmission.

The messages format is JSON, this is a human readable format that is also a very common in use. Fig. 7 shows an example of how the vehicle location information is transmitted using this format. "veh_id" is a unique code for every vehicle and "status" depict specific status information at the time information was acquired. "date" and "time" are the timestamp of the sample. "gps_speed", "lat" and "lon" where directly acquired by the GPS sensor. In similar way, we have specified three more messages. A generic sensor message may contain all the A/D sensor values, the OBD acquired values plus any other generic sensor value. An OBD fault message may contain DTC codes and a diagnostic message may contain any IoT Sensor Node diagnostic codes.


```
{
  "veh_id": "ABC1234",
  "status": "0GRR10",
  "timestamp":
   {
    "date": "20102020", #DAY, MONTH, YEAR.
    "time": "10:10:32", #HOUR, MIN, SEC.
   },
  "gps_speed" : 45, #SPEED KMPH

   "coordinates":
   {
    "lat" :37.123456,
    "lon" :23.123456
   }
}
```

Fig. 7 A GNSS record with JSON format

An important system function is how the timestamp is obtained. The operating system should always use a valid time service. The RPi3 is not having a real time clock circuit so the valid time should be obtained every time the system boots. When the system boots the application, it is testing for a valid internet connection, in this case the system time is updated using the NTP service. If for any reason there is no internet connection, the system time is acquired from a valid GNSS signal. The GPS is a provider of accurate clock information. The application halts into a loop trying to acquire a valid system time, until a valid clock is acquired any other operation is prohibited.

At the end of our development, we were confident that we could use a high-level OS and low-cost hardware for a demanding telemetry application. Linux is open source and free of charge, it has a huge support base. RPi3 is a very low-cost computing multicore platform that can exploit the features of Linux OS. We could integrate everything with a plethora of interfaces and maintenance/debugging options. We could appreciate all the high-level functionality of networking interfaces and database management. They offer the highest flexibility with proven software packages. Telematics hardware is usually using embedded microcontrollers with or without an RTOS. Still such level of versatility is not existent. For example, remote access with OpenVPN/SSH is almost impossible to achieve due to memory constraints and database is limited to serial records or linked lists in non-volatile memory. RPi3 is using a multi gigabyte micro-SD that gives in practice unlimited space.

VI. Conclusion

In the present paper, we presented the design of an IoT sensor node appropriate for the InteligentLogger system, which is an e-maintenance system for heavy-duty vehicles/machinery. The specific IoT sensor node provides compatibility with both old and new technology vehicles through a variety of available sensor interfaces and OBD interfaces while at the same time it provides positioning information. All information is transmitted in real-time through a cellular network when/where network coverage is provided. Alternatively, one can directly download the information through WiFi by using a smartphone or a tablet running an appropriate application and transmitted to the database as soon as the smartphone/tablet connects to the internet. More information about parameters that cannot be automatically acquired can also be inserted through the specific application. A functional prototype of the specific IoT sensor node has already been produced, tested at the laboratory, and has currently been put in on-vehicle test operation. The specific IoT sensor node is expected to play a decisive role in the successful solution of the challenging task of diverse heavy-duty vehicles/machinery fleets by the IntelligentLogger system.

Acknowledgment

This research has been co-financed by the European Union and Greek national funds through the Operational Program Competitiveness, Entrepreneurship and Innovation, under the call RESEARCH—CREATE—INNOVATE (project code: T1EDK- 01359, IntelligentLogger).

References

[1] S. Mohammad, M. A. A. Masuri, S. Salim, and M. R. Abdul Razak, “Development of IoT Based Logistic Vehicle Maintenance System,” in *Proc. IEEE 17th International Colloquium on Signal Processing & Its Applications (CSPA),* Langkawi, Malaysia, 2021, pp. 127-132.

[2] B. Iung, E. Levrat, A. Crespo Marquez, and H. Erbe, “Conceptual framework for e-Maintenance: Illustration by e-Maintenance technologies and platforms,” *Ann. Rev. Control*, vol. 33, pp. 220-229, 2009.

[3] E. Levrat, B. Iung and A. Crespo Marquez, “E-maintenance: review and conceptual framework,” *Prod. Plan. Control*, vol. 19, no. 4, pp. 408-429, 2008.

[4] A. Muller, A. Crespo Marquez and B. Iung, “On the concept of e-maintenance: Review and current research,” *Reliability Eng. Syst. Safety*, vol. 93, pp. 1165-1187, 2008.

[5] A. Bousdekis and G. Mentzas, “Condition-Based Predictive Maintenance in the Frame of Industry 4.0,” in *Advances in Production Management Systems. The Path to Intelligent, Collaborative and Sustainable Manufacturing*, H. Lödding et al. (eds.), Cham: Springer, 2017, pp. 399-406. https://doi.org/10.1007/978-3-319-66923-6_47

[6] R. S. Velmurugan and T. Dhingra, “Asset Maintenance: A Primary Support Function,” in *Asset Maintenance Management in Industry*, R. S. Velmurugan and T. Dhingra (eds.), Cham: Springer, 2021, pp. 1-21. https://doi.org/10.1007/978-3-030-74154-9_1

[7] D. Goustouridis, A. Sideris, I. Sdrolias, G. Loizos, N. A. Tatlas, and S. M. Potirakis, “IntelligentLogger: A Heavy-Duty Vehicles Fleet Management System Based on IoT and Smart Prediction Techniques,” *World Academy of Science, Engineering and Technology, Open Science Index 176, International Journal of Mechanical and Industrial Engineering*, vol. 15(8), pp. 336 - 340, 2021. https://publications.waset.org/10012185/pdf

[8] A. BinMasoud and Q. Cheng, "Design of an IoT-based Vehicle State Monitoring System Using Raspberry Pi," *in Proc. 2019 International Conference on Electrical Engineering Research & Practice (ICEERP), 2019*, pp. 1-6.

[9] S. K. Singh, A. K. Singh, and A. Sharma “OBD - II based Intelligent Vehicular Diagnostic System using IoT,” *in Proc. ISIC’21: International Semantic Intelligence Conference*, Delhi, India, 2021, pp. 511-515.

[10] N. Goyal, V. Goel, M. Anand, and S. Garg, “Smart Vehicle: Online Prognosis for Vehicle Health Monitoring,” J. Innovation in Computer Sci. Eng., vol. 9, no. 2, pp. 12-22, Jan-June 2020.

[11] B. C. Nithin, S. Pooja, K. G. Sampath, and S. S. Sharmila, “On-Board Vehicle Fault Monitoring System”, *pices*, vol. 4, no. 5, pp. 82-84, Sep. 2020.

[12] S. Hussain, U. Mahmud and S. Yang, "Car e-Talk: An IoT-Enabled Cloud-Assisted Smart Fleet Maintenance System," *IEEE Internet of Things J.*, vol. 8, no. 12, pp. 9484-9494, 15 June15, 2021.

[13] R. Barnes, “Raspberry Pi 3: Specs, benchmarks & testing,” *The MagPi Magazine*, 2016; https://magpi.raspberrypi.org/articles/raspberry-pi-3-specs-benchmarks (accessed on 26/06/2021).

[14] L. Hattersley, “Build a car computer 'carputer' with Raspberry Pi,” *The MagPi Magazine*, 2019; https://magpi.raspberrypi.org/articles/build-car-computer-raspberry-pi (accessed on 26/06/2021).