UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

INTELLIGENT POWER AWARE ALGORITHMS FOR TRAFFIC SENSORS

A THESIS

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

MASTER OF SCIENCE

in

Electrical and Computer Engineering

By

SIRAJ MUHAMMAD
Norman, Oklahoma
2018

INTELLIGENT POWER AWARE ALGORITHMS FOR TRAFFIC SENSORS

A THESIS APPROVED FOR THE
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

BY

_____
Dr. Hazem Refai, Chair

_____
Dr. Thordur Runolfsson

_____
Dr. Kam Wai C. Chan

© Copyright by SIRAJ MUHAMMAD 2018
All Rights Reserved.

To the soft-hearted and the sapient, my parents, Thanaa and Muhammad.

To the great and trustworthy, my dear brother, Obada.

To the beautiful little roses that adorn my life, my sisters, Einas and Israa.

To you all I dedicate this work,

Acknowledgements

I would like to offer my genuine appreciation and heartfelt gratitude to my dear advisor, *Dr. Hazem Refai* for his unparalleled support, mentorship, and constant encouragement to grow and succeed.

I extend my sincere appreciation to my committee members, *Dr. Thordur Runolfsson* and *Dr. Kam Wai Chan* for their time and insightful suggestions to improve this work.

I would also like to acknowledge Dr. Walid Balid for his assistance in the development of this research. Special thanks as well to Michelle Farabough for editing this thesis.

To my extended family at OU, professors, students, and staff, I am thankful to you all for everything you have provided me during my time here!

Table of Contents

List of Tables

List of Figures

Abstract

The Internet of Things (IoT) is reshaping our world. Soon our world will be based on smart technologies. According to IHS Markit forecasts, the number of connected devices will grow from 15.4 billion in 2015 to 30.7 billion in 2020. Forrester Research predicts that fleet management and the transportation sectors lead others in IoT growth. This may come as no surprise, since the infrastructure (roadways, bridges, airports, etc.) is a prime candidate for sensor integration, providing real-time measurements to support intelligent decisions. The energy cost required to support the anticipated enormous number of predicted deployed devices is unknown. Currently, experts estimate that 2 to 4% of worldwide carbon emissions can be attributed to power consumption in the information and communication industry [1].

This thesis presents several algorithms to optimize power consumption of an intelligent vehicle counter and classifier sensor (*i*VCCS) based on an event-driven methodology wherein a control block orchestrates the work of various components and subsystems. Data buffering and triggered vehicle detection techniques were developed to reduce duty cycle of corresponding components (e.g., microSD card, magnetometer, and processor execution). A sleep mode is also incorporated and activated by an artificial intelligence-enabled, reinforcement learning algorithm that utilizes the field environment to select proper processor mode (e.g., run or sleep) relative to traffic flow conditions. Sensor life was extended from 48 hours to more than 200 days when leveraging 2300 mAh battery along with algorithms and techniques introduced in this thesis.

Chapter 1 Introduction

Transportation systems are at the heart of our socio-economic environment and played a major role in shaping the trajectory of economic strength and quality of life in a given community. The opposite effect is also true. For a strong and continually prosperous economy, it is a necessity for underlying infrastructure to be equipped and up-to-date with technological advancements characteristic of other industry sectors. Transportation systems must integrate and interoperate with others.

Intelligent Transportation System (ITS) technology has been accelerated since the turn of the century due to a variety of technological forces that have caused an ever-increasing need to efficiently utilize the transportation infrastructure. In spite of the fact that other industries (e.g., information technology, communications) have evolved at a faster pace, the 2010-2011 ITS America Annual Report informs [2]:

- 77% of fixed bus route agencies have real-time arrival data.
- 94% of toll roads have electronic collection.
- 70% of the population is covered by 511 systems in 38 states.
- Thousands of miles of highway and arterial roads are managed under Traffic Management Centers surveillance.

ITS is one of the most promising sectors for Internet of Things (IoT) hype, primarily due to the impact transportation system elements will have on domains connected to the Internet cloud. Smart traffic signals and surveillance systems enable more convenient traffic planning by communicating traffic conditions and flow to connected vehicle on-board systems, which, in turn, will aid in saving an enormous amount of time due to traffic congestion delays. ITS America reports that each year the

average American spends 40 hours sitting in traffic [3]. Moreover, public safety vehicles, buses, and commercial vehicle fleets will benefit from a smart, connected infrastructure able to communicate valuable information about traffic and road conditions. Cisco Systems has already introduced a high-level system overview, namely Cisco Connected Roadways [4].

An essential ITS system component is vehicle detection and classification. However, the integral real-time traffic monitoring and analysis functionality proves problematic. Commuters, traffic administrators and agencies must rely on such systems for improving traffic control and management, as well as for trip planning and routing decisions.

Technologies used for vehicle detection and surveillance can be categorized into two types:

- In-roadway sensors, including Inductive Loop Detectors (ILD), magnetic sensors, piezoelectric sensors, and Weigh-In-Motion (WIM) sensors.
- Off-roadway sensors, including video image processors, microwave radars, infrared sensors, ultrasonic sensors, and passive acoustic arrays sensors.

Interested readers can find additional details about these technologies in [5]. Existing technologies, like Bluetooth, have also been used to detect vehicles and estimate highway travel time [6].

Regardless of the technology, the era of IoT is redefining the objectives and standards of all systems. IoT device are expected to be smart, reliable, and low-cost, without significant maintenance. For ITS systems to fit into the IoT paradigm, they must adhere to the inherent need for constrained resources for processing, memory,

power, and security functionality. These could impose further difficulties and challenges on ITS, especially given the nature of the applications they target. For instance, some Automatic Vehicle Classification (AVC) systems cannot save collected data for more than several days. WIM systems, on the other hand, can store data for up to a year. Problems with power consumption persist, whether at the processing or communication stage at the point when data is transmitted to an access point for further processing.

The work detailed in this thesis is an extension to a project focused on designing a low-cost, reliable, and low-power intelligent vehicle counter and classification sensor (*i*VCCS) system [7] in which the sensor exploits the physical phenomenon of magnetic field disturbance caused by ferrite materials in the body of a vehicle. The sensor relies on a triaxial magnetometer to measure magnetic field before measurements are transmitted to processing platform. Previous work (i.e., first generation *i*VCCS [*i*VCCS 1stG]), proved the design concept with high accuracy and reliability [7]–[9]. A second generation *i*VCCS (*i*VCCS 2ndG) was designed around a more powerful, 32-bit microcontroller equipped with an on-die digital signal processing (DSP) core. Special attention to power consumption was lead to the development of the new *i*VCCS 2ndG platform.

This research focuses on minimizing sensor power consumption while maintaining accurate vehicle detection, count, and speed estimation. Several algorithms were ported and modified from *i*VCCS 1stG to accommodate the new platform and reduce overall circuit current consumption of the system. A number of new techniques were also introduced to enable interrupt-based microcontroller operations and reduce

duty cycle. In addition, a dynamic power management was proposed, leveraging the merits of artificial intelligence.

The balance of this thesis is organized, as follows:

- Chapter 2 includes the theoretical concept of magnetometers and their application in vehicle detection and counting; a literature review of work related to low-power embedded systems and dynamic power management is also provided.
- Chapter 3 focuses on an overview of the system platform, describing hardware and specifications.
- Chapter 4 describes algorithm modifications ported from the *i*VCCS 1stG to *i*VCCS 2ndG.
- Chapter 5 introduces new techniques developed to further optimize sensor operations relative to interrupts in favor of a polling methodology for reducing duty cycle; a power analysis of components is also presented.
- Chapter 6 highlights the dynamic power management problem and proposes a reinforcement algorithm to predict activities to reduce power consumption.
- Chapter 7 reports results collected from tests and deployments obtained in both lab and real-world scenarios.
- Chapter 8 concludes the thesis and offers direction for further research.

Chapter 2 Background and Related Work

2.1 Magnetometer Theory of Operation

Geomagnetic field is a force field that surrounds the earth's surface and flows from the earth's interior into the space. It is similar to a field generated by a simple bar magnet tilted, currently estimated at an 11.5° angle. Although the force field's intensity is in constant flux and has experienced large-amplitude variations over the past 800 thousand years [10], it is considered rather uniform at any region on the earth's surface, ranging between 25 and 65 μT (0.25 to 0.65 gauss).

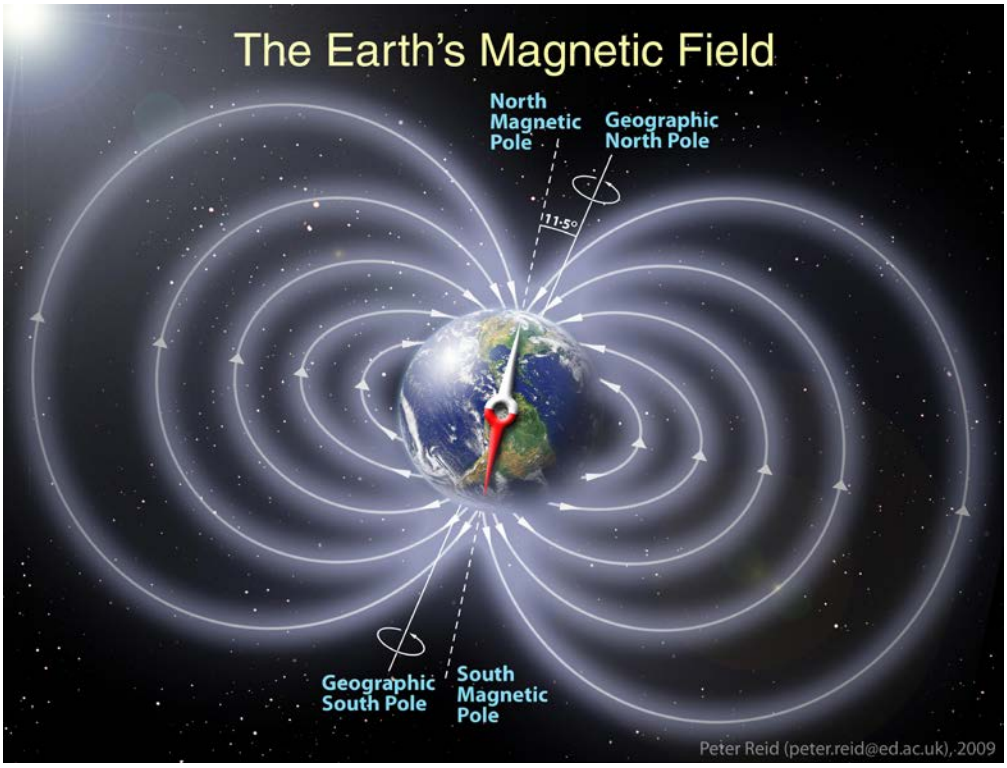

Figure 1. Magnetic field of the earth [11]**.**

Ferrous materials in traveling vehicles cause small, local disruptions in the geomagnetic field. Due to high magnetic permeability of these ferrous materials, the flux lines of the field are absorbed and distorted in a non-linear form as a vehicle passes

through the field, as shown in Figure 2. Several factors play a role in the resulting intensity magnitude and direction of field changes (e.g., speed, density, and size mechanical form).

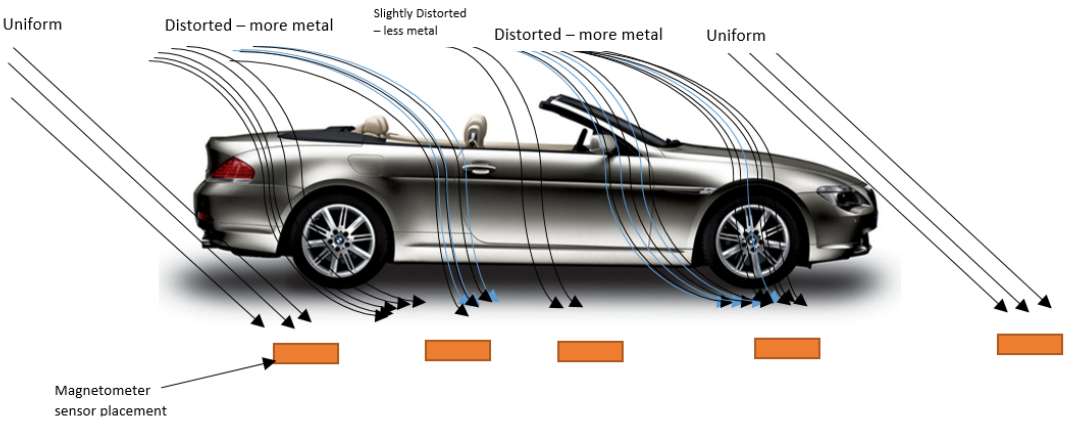

Figure 2. Disturbance in magnetic flux lines caused by a vehicle.

Measured disturbance is known as the vehicle magnetic signature, which is unique and differs from one vehicle to another. The disturbance can be modeled as a number of magnetic dipoles [12]. Signatures can be measured by magnetometers, which report three geomagnetic components of the field: north B_x, east B_y, and vertical B_z.

2.2 Related Research Work

Power consumption is a primary concern for wireless sensor node design, as each element of the design (e.g., processing unit, power management unit (PMU), processing algorithm, and, most importantly, system components) is affected. As such, it is critically important to carefully select each device and peripheral as a first step to sensor design [13]–[15]. *i*VCCS 2ndG was purposefully designed to optimize low power consumption.

While technological advancements have reached a point at which application performance is ensured, energy efficiency remains challenging, especially given IoT

integration. Modeling power consumption at the early stages of development and pre-deployment becomes decisive. Authors in [16] propose a general methodology to simulate wireless sensor node power consumption by profiling three essential aspects of WSNs: networking, sampling, and processing. Although electrical component characteristics and wireless interfaces play a major role in determining power consumption, a particular standard, component, and technology becomes crucial when developing an algorithm. A model for performance evaluation is necessary for optimization.

Literature has suggested building a predictor for estimating remaining power in the main energy storage and predicting the amount of power harvested during future time slots [17], [18]. Such mechanisms aid in determining which functionalities to disable and for how long based on a corresponding peripheral power profile. This stage of development can benefit from the power consumption model proposed in previous references. In [17], authors optimized RF transmission time — the most power-hungry system element — to determine an optimal transmission interval.

Software development is dependent upon energy consumption. Given that a nonoptimal algorithm is executed for energy consumption, selecting low-power components is inconsequential. In fact, several techniques must then be orchestrated to achieve a satisfactory performance. Most energy management implementations involve smart and intelligent use of peripherals and system sensors [17]–[20]. Indeed, the heart of the power management process should be an algorithm that determines when to enable and disable each module on the board, in addition to flexibly controlling and switching between power schemes. This is preferred over a fixed profile that runs the

system in its many situations. Such an algorithm is commonly referred to as Dynamic Power Management (DPM).

Researchers have defined and tried to resolve the DPM problem. Authors in [21] modeled system behavior as a time series of busy and idle periods, suggesting that various idle intervals require different sleep modes (see Figure 3), primarily because switching takes time and more energy than nominal consumption. They employed adaptive tree learning as a method for solving policy selection wherein a predictor is used to indicate when and for how long idle states occur.

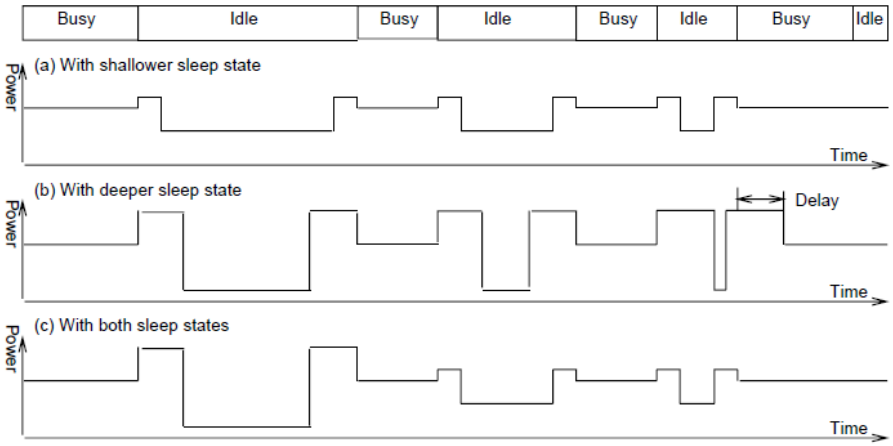

Figure 3. Changing system sleep mode according to load [21]**.**

In [22], authors used Reinforcement Learning (RL) to adaptively choose the optimal power management policy in a given idle state wherein an action is taken and objectives are evaluated. Based on outcomes, subject-to-constraints (e.g. performance) state-action pairs are rewarded/punished, assigning a Q-value. A variation of the Q-Learning algorithm, namely SARSA (State-Action-Reward-State-Action), is used to monitor values in a Q-table. Idle time is known and can be calculated according to assumptions.

A more recent paper [23] used the Q-learning algorithm to optimize power consumption in a video encoder System-on-Chip (SoC).

Authors in [24] proposed a machine learning algorithm to select the optimal policy online. The algorithm considers a variant of RL with weights placed on policies and a loss function to update weights. Assumed idle times are known; the way in which loss function affects weight is not clear.

Chapter 3 System Platform Overview

*i*VCCS 2ndG is a fully autonomous system designed to achieve self-powered, low-power operation through energy harvesting without sacrificing performance. The new system is carefully crafted in a compact design (45×30×6 mm) that combines high-performance, energy-efficient components and that is equipped with a power management subsystem for minimizing energy consumption while maintaining accurate vehicle count, logging, and speed estimation. Figure 4 shows the interconnections among various components. A gauge monitors and reports battery capacity to the microcontroller unit (MCU). Nano-power load switches are placed at the power lines of certain energy-hungry sensor components, including the RF wireless module, GPS receiver, and SD card, among others. Energy-efficient algorithms were developed to accurately detect, count, estimate speed, and control various system components. Figure 5 shows top and bottom views of the printed circuit board.

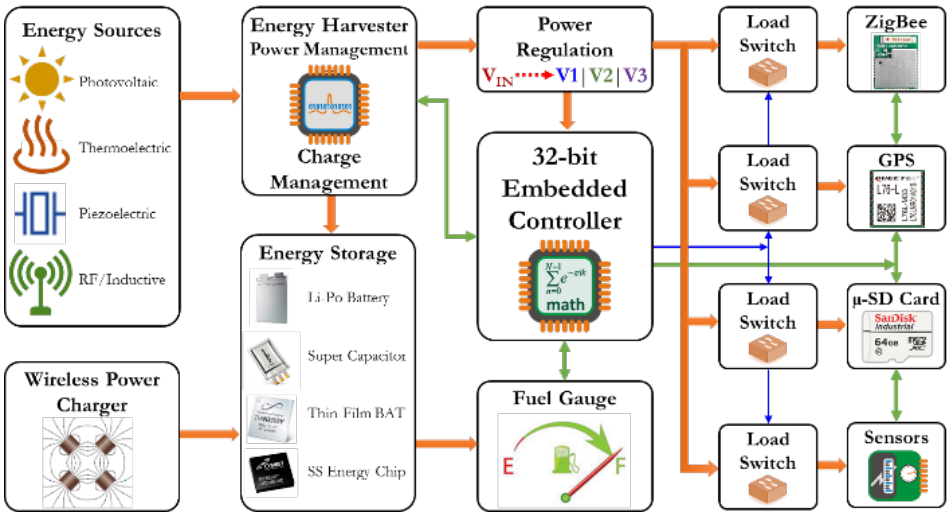

Figure 4. Overview of the system components [7].

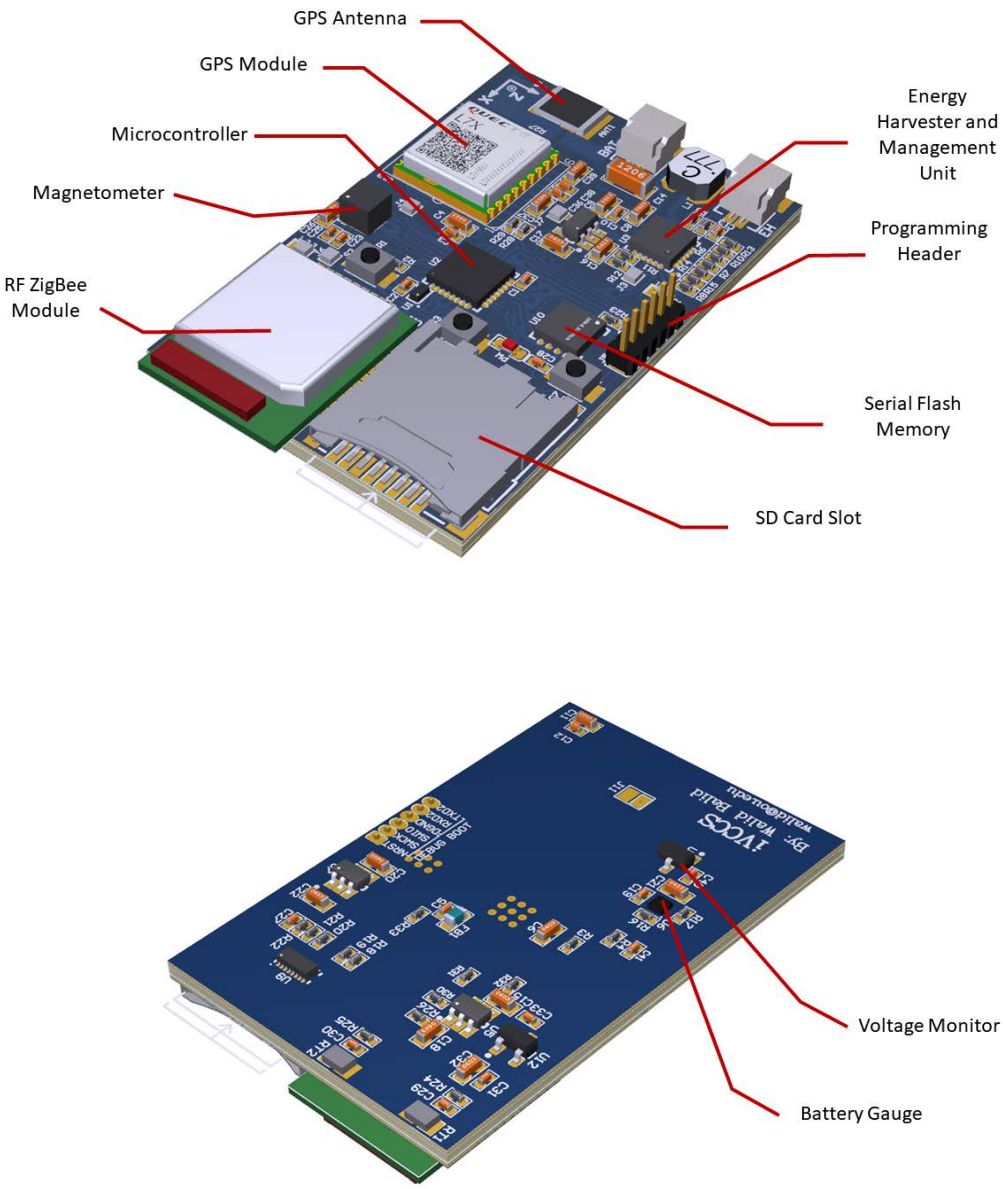

Figure 5. *i*VCCS PCB components.

3.1 Microcontroller

At the heart of the sensor system lies an ultra-low power platform manufactured by STMicroelectronics, equipped with an ARM CORTEX-M0+ 32-bit RISC core –

STM32L071 [25]. This microcontroller is characterized by number of features that make it appealing for low-power applications [26]. These include:

- 0.29 μA standby mode (3 wakeup pins)
- 0.43 μA stop mode (16 wakeup lines)
- 0.86 μA stop mode + RTC + 20 KB RAM retention
- Down to 93 μA/MHz in run mode
- 5 μs wakeup time (from flash memory)
- Core from 32 kHz up to 32 MHz max
- 1 to 25 MHz crystal oscillator
- 32 kHz oscillator for RTC with calibration
- Up to 192 KB flash memory with ECC
- 20 KB RAM

3.2 Magnetometer

Kionix KMX62 is a 6 degrees-of-freedom magnetometer/accelerometer inertial sensor system [27] that senses changes in the magnetic flux of earth's surface due to vehicles passing in its vicinity. KMX62 is a very reliable and power-efficient sensor, consuming 395 μA at high-resolution mode and 1 μA in standby mode. The magnetometer's full-scale range is ±1200 μT, and digital bit depth is 16 bits, resulting in a magnetic sensitivity of ±0.0366 μT compared to ±0.1 μT in *i*VCCS 1stG.

3.3 ZigBee Module

The radio frequency (RF) frontend utilizes a ZigBee transceiver from ZLG based on an NXP JN5168 microcontroller [28]. AW516x is a low-power, high-performance ZigBee module that incorporates the IEEE 802.15.4 standard and supports

a variety of protocols on top (e.g., FastZigBee, ZNET, ZigBee-PRO, and RF4CE). The AW5161P0CF module used in *i*VCCS 2ndG has a small footprint (13.5×16.5 mm) and features a ceramic antenna. Receiver sensitivity is -95 dBm, consuming 21 mA, and transmitter power is 2.5 dBm with 18 mA current consumption. The engine features a deep sleep mode with 100 nA typical current. Data transfer rate of the system is 2 Mbps.

3.4 GPS

Time synchronization is achieved via a GPS module manufactured by Quectel. L76-L GPS receiver module integrates GLONASS and GPS systems [29] and provides a built-in low-noise amplifier (LNA) for improved performance for 33 tracking channels, 99 acquisition channels, and 210 PRN channels. The module features EASY technology [29] and allows L76-L to automatically calculate and predict orbits using ephemeris data stored in the internal flash of the module. This reduces time-to-fix even in indoor situations with poor signal levels. Current consumption is 25 mA in acquisition mode and 7 µA in backup mode.

3.5 Real-Time Clock

STM32L071 also offers an internal Real-Time Clock (RTC) and features low power operation [26]. It has on-the-fly correction capability with a range of 1 to 32767 RTC clock pulses, allowing the system's 1 Hz clock to synchronize with the Pulse Per Second (PPS) signal of the GPS module subsequent to fix acquisition. The internal RTC is clocked by a 32.768 kHz external ultra-low power oscillator manufactured by SiTime [30], which offers a current consumption of <1 µA and frequency stability of ±5, ±10,

±20 ppm options over temperature, as well as the world's smallest footprint (1.5×0.8 mm CSP).

3.6 Data Storage

The system integrates a microSD card slot used to for raw data acquisition (e.g., vehicle magnetic signature or accelerometer), in addition to timestamps of vehicles arrival/departure and status messages. The micro-SD card is connected to the microcontroller through serial peripheral interface (SPI). In addition, an on-board 64 Mb serial NOR flash memory is available as a secondary storage medium. Macronix's MX25R64 [31] is an ultra-low power CMOS flash memory with a minimum of 100,000 erase/program cycles and 20-year data retention. MX25R64 features a typical 5 μA standby current, a maximum 4 mA read current, and 6 mA write current.

3.7 Battery Gauge

As part of the power management subsystem, a smart battery gauge is integrated for monitoring battery capacity and protecting it from deep discharge and over charge, especially for Li-Po batteries. Texas Instrument BQ27621-G1 [32] was used, as it provides advanced algorithms for calculating remaining battery capacity (mAh), state-of-charge (%), battery voltage (mV), and temperature (°C); it also requires little to no configuration and can be accessed through 400 kHz I^2C interface. Figure 6 shows a simplified schematic of gauge-battery pack connection.

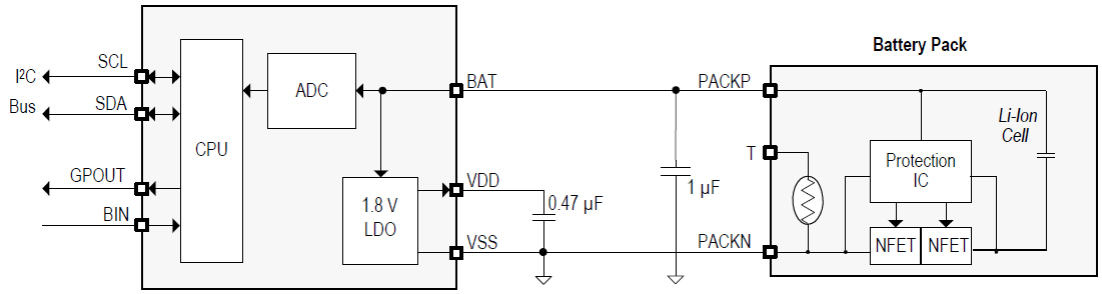

Figure 6. Simplified schematic of the battery gauge unit [32]**.**

3.8 Energy Harvester and Management Unit

ADP5091 [33] is an ultra-low PMU that harvests energy from photovoltaic (PV) and thermoelectric generators (TEG) sources. The unit can efficiently convert power from sources with range as low as 6 µW to 600 mW without losing significant energy (i.e., sub-microwatt). The chip features a cold-start circuit with input voltage as low as 380 mV. After cold-start, the regulator can operate at an input range of 80 mV to 3.3 V. Stable DC-to-DC boost conversion is realized through the use of a maximum power point tracking (MPPT) controller. Figure 7 shows a detailed functional diagram of the device.

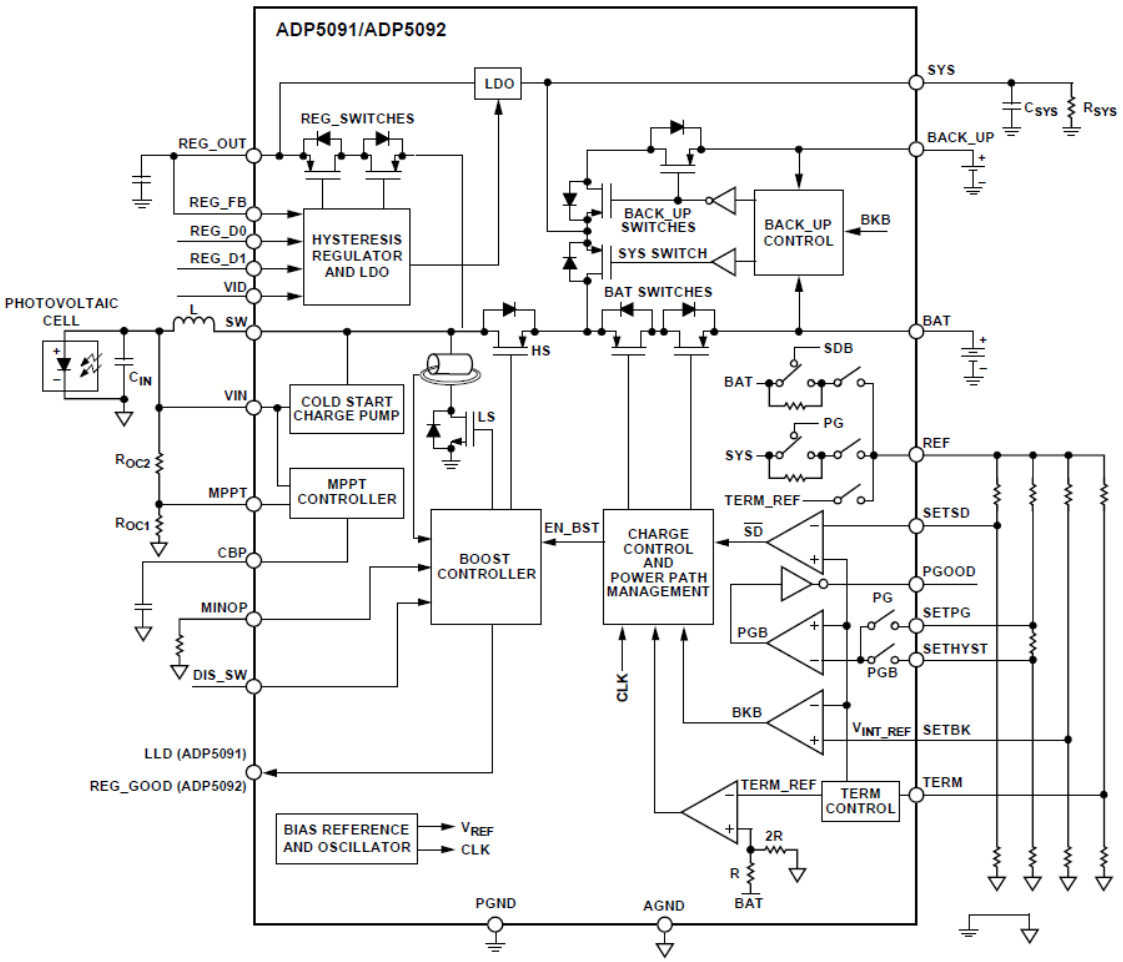

Figure 7. ADP5091 detailed functional block diagram [33].

Chapter 4 Porting Algorithms from 1stG to 2ndG

*i*VCCS 2ndG uses most algorithms developed in *i*VCCS 1stG with necessary modifications for platform differences. *i*VCCS 1stG was based on an 8-bit microcontroller and firmware developed in BASCOM; *i*VCCS 2ndG is based on a 32-bit ARM Cortex microcontroller. Firmware was developed in C due to its efficiency and speed. In this chapter, essential algorithms and proposed flow charts are discussed, as well as differences between 1st and 2nd generations.

4.1 Detection and Counting Algorithm

The detection algorithm developed in *i*VCCS 1stG is ported to the new design with modifications to fit the new platform [8]. The algorithm processes the magnetic flux sampled by KMX62 and detects vehicle arrival/departure as it passes through the sensor zone. As aforementioned, vehicles have ferrous materials that cause disturbance in the local magnetic field, creating a push-and-pull effect in the flux lines as the vehicle passes. The result is fluctuations in the magnitude of the magnetic field. KMX62 is a tri-axial sensor that represents each sample point by three 16-bit words (i.e., x, y, z). Microcontroller calculates magnitude, and then communicates it to the algorithm for processing. Three thresholds and three debounce timers govern the algorithm. R_{TH} represents the baseline threshold; O_{TH} is the onset threshold; and H_{TH} is the holdover threshold. These parameters define vehicle arrival/departure. Figure 8 depicts a sample vehicle signature with illustrated thresholds and timers. Given that the magnitude reaches O_{TH} threshold, a timestamp of arrival is logged. If magnitude drops below H_{TH}, the vehicle is assumed departed, and another timestamp is logged, adding '1' to the vehicle counter. In the absence of a vehicle in the vicinity, magnitude is expected to

stay below R_{TH}. The state-machine of the algorithm is illustrated in Figure 9. C code can be found in Appendix A.

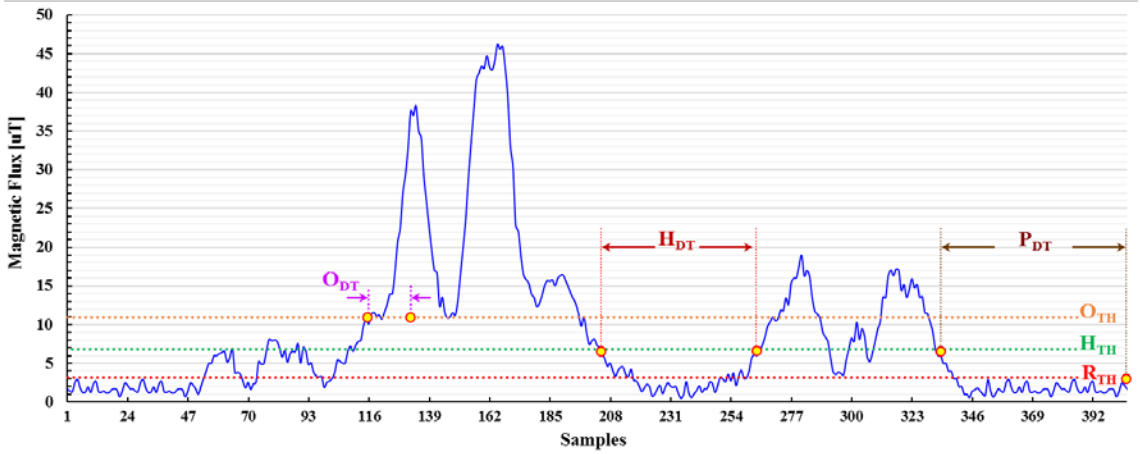

Figure 8. Detection algorithm parameters applied on a vehicle flux magnitude [7]**.**

Given magnitude rises between R_{TH} and H_{TH}, calibration is executed. Debounce timers play a crucial role in eliminating misdetections and double detections (i.e., counted twice). Onset debounce timer O_{DT} is used to filter glitches in a signature that result in misdetections. Double detections occur as a result of ferrous material distributed throughout the body of a vehicle. Long trucks, for instance, with extended separations between axles cause dips and steep fluctuations between O_{TH} and H_{TH}, resulting in a single vehicle counted twice. Holdover debounce timer H_{DT} is leveraged to overcome this problem.

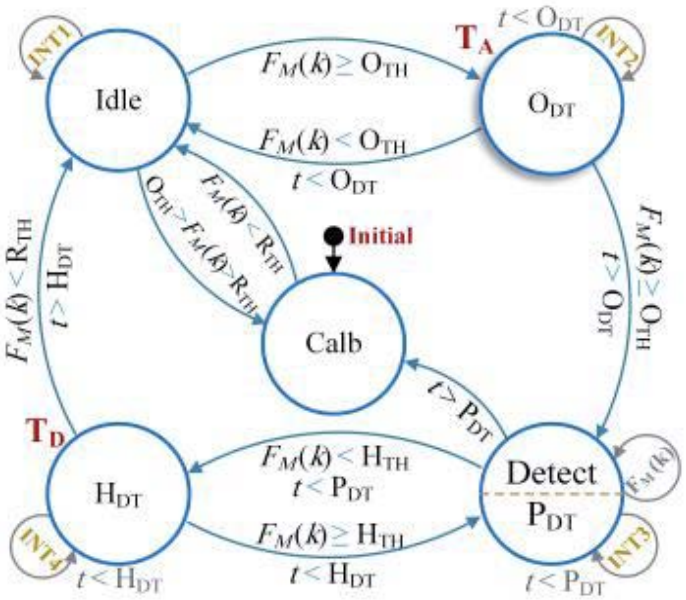

Figure 9. Detection algorithm state machine [7]**.** $\boldsymbol{F_M(k)}$ **is the field magnitude.**

4.2 Adaptive Compensation of Baseline Drift

Magnitude measured by magnetometer sensor (i.e., KMX62) can suffer from drifts due to temperature, displacement caused by vehicles hitting the sensor, and/or operating stress. Consequences of such result in misdetection and speed estimation errors. Figure 10 shows an example of such a drift over 240 minutes due to temperature [8]. Consequently, magnetometer baseline should be regularly tracked and re-calibrated using a Moving Average Filter (MAF). Given that magnitude drops below on-set threshold O_{TH}, calibration algorithm is executed. Moreover, calibration algorithm is carried out subsequent to confirmation of vehicle departure. Algorithm C code (See Appendix A) was redesigned in *i*VCCS 2ndG to make it more readable and intuitive. Previously, multiple flags and function calls were required for recalibration. A high-level description of the algorithm is shown in Figure 11.

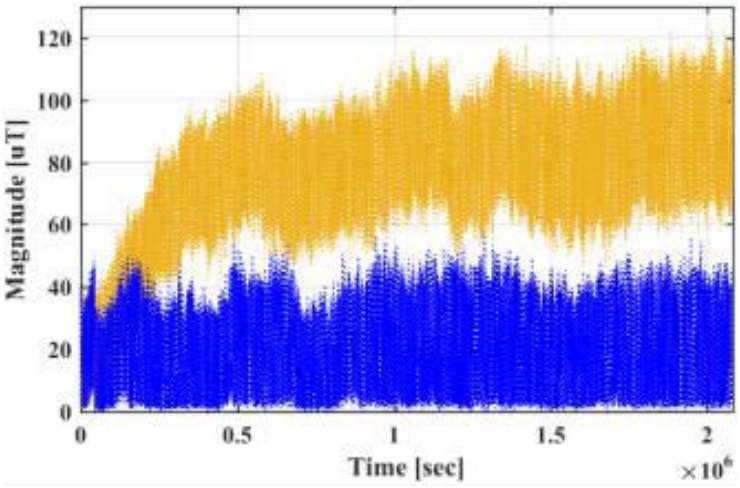

Figure 10. Baseline magnitude drift with (blue) and without (yellow) adaptive compensation [8]**.**

Inputs: x, y, z

Outputs: x_{ref}, y_{ref}, z_{ref}

1: Disable external interrupts except DRI

2: **FOR** i < M

3: **WAIT FOR** Magnetometer Data-Ready interrupt

4: $x_{acm} \leftarrow x_{acm}$ + READ x

5: $y_{acm} \leftarrow y_{acm}$ + READ y

6: $z_{acm} \leftarrow z_{acm}$ + READ z

7: **LOOP**

8: $x_{avg} \leftarrow x_{acm}$ / M

9: $y_{avg} \leftarrow y_{acm}$ / M

10: $z_{avg} \leftarrow z_{acm}$ / M

11: Enable external interrupts

Figure 11. High-level description of adaptive compensation algorithm. x, y, and z are the measured field's components. x_{acm}, y_{acm}, and z_{acm} are the accumulated values.

4.3 Adaptive Compensation of the RTC Frequency Drift

RTC skew can occur over time, desynchronizing local and remote clocks due to a number of factors, including variations in temperature, effect of passive board components, and tolerances specified by manufacturers. Because time synchronization is critical among system sensor nodes, especially for speed estimation, the drift phenomenon should be compensated. A GPS-based approach was developed to correct RTC clock drift in *i*VCCS 1stG and ported to C in *i*VCCS 2ndG. A block diagram of the algorithm is depicted in Figure 12.

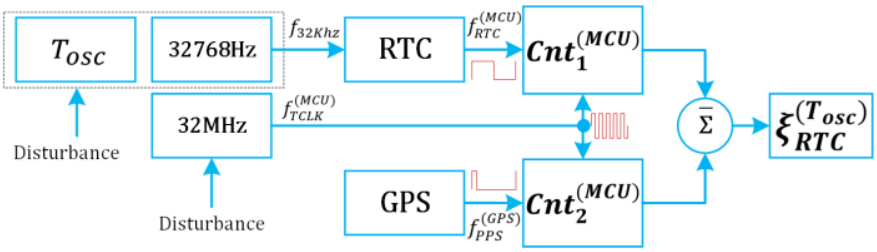

Figure 12. GPS-based RTC time drift correction system block diagram [8]**.**

After the GPS is locked to a fix, GPS module provides a 1 Hz signal, Pulse Per Second (PPS) routed to one of the GPIOs (General Purpose Input/Output) of the microcontroller. This signal is considered the accurate reference for RTC calibration. The RTC 1 Hz signal frequency $f_{RTC}^{(MCU)}$ is compared to PPS signal frequency $f_{PPS}^{(GPS)}$. The algorithm measures both signal durations using high frequency clock $f_{TCLK}^{(MCU)}$, driven from the MCU's 8 MHz oscillator. Error induced in measuring clock (i.e., oscillator) is canceled in the differentiation stage since both signals are measured at the same time using the same clock.

The algorithm commences by aligning RTC and PPS signals. Subsequently, two 32-bit MCU timers (i.e., Cnt1 and Cnt2) are configured in up-counting mode and controlled by RTC 1 Hz signal and PPS 1 Hz signal, respectively. Both originate at the first rising edge of 1 Hz signal and stop on the following rising edge. Time correction coefficient is defined as the difference between the two counters. Figure 13 and Figure 14 show a flow chart that explains the algorithm in detail.

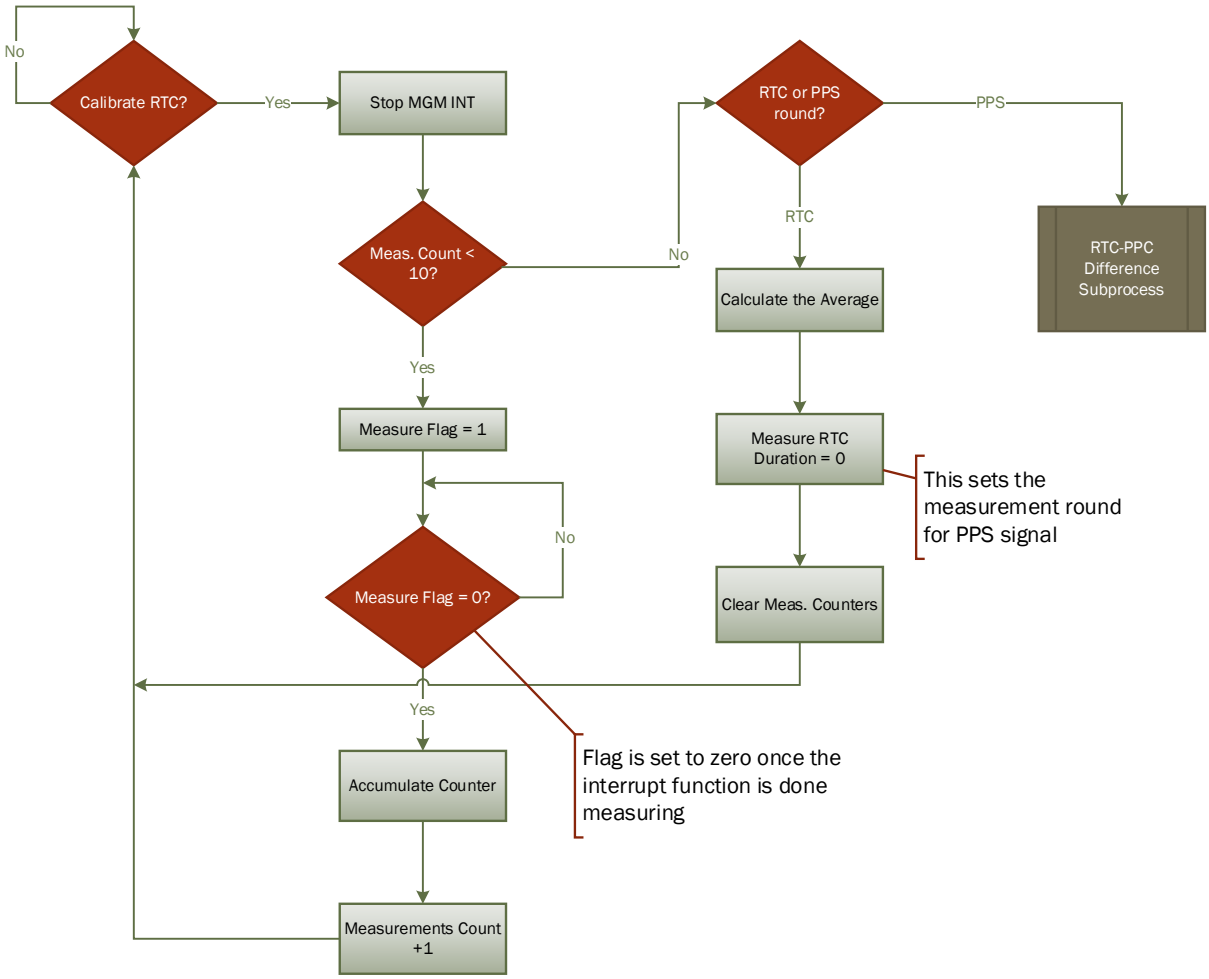

Figure 13. A flow chart depicting the algorithm for RTC drift calibration.

The algorithm sets a flag `measure_freq` to inform the interrupt function of RTC 1 Hz — called each time one second passes — to start counters, and the stop them on the next rising edge of the signal (i.e., one period of 1 Hz signal).

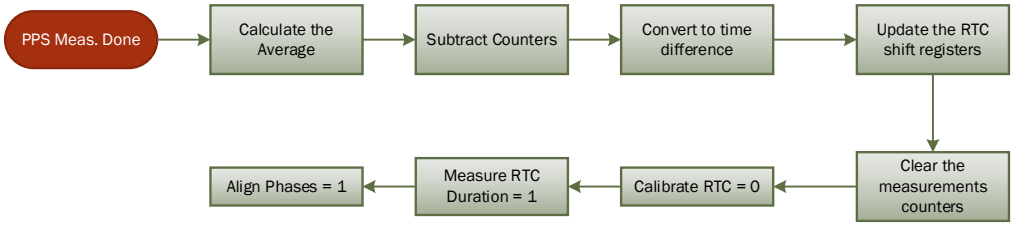

Figure 14. PPS-RTC difference calculation sub process.

Once interrupt measuring is complete, the function sets `measure_freq` to zero, indicating the completion of the measuring phase relative to the main function. The microcontroller proceeds with the algorithm (Figure 15). The process is repeated for the number of taps of a moving average filter (default value is 10), and then measurements are averaged. The same procedure is repeated for PPS from the GPS module after it is locked and synchronized. Information about C implementation is available in Appendix A.

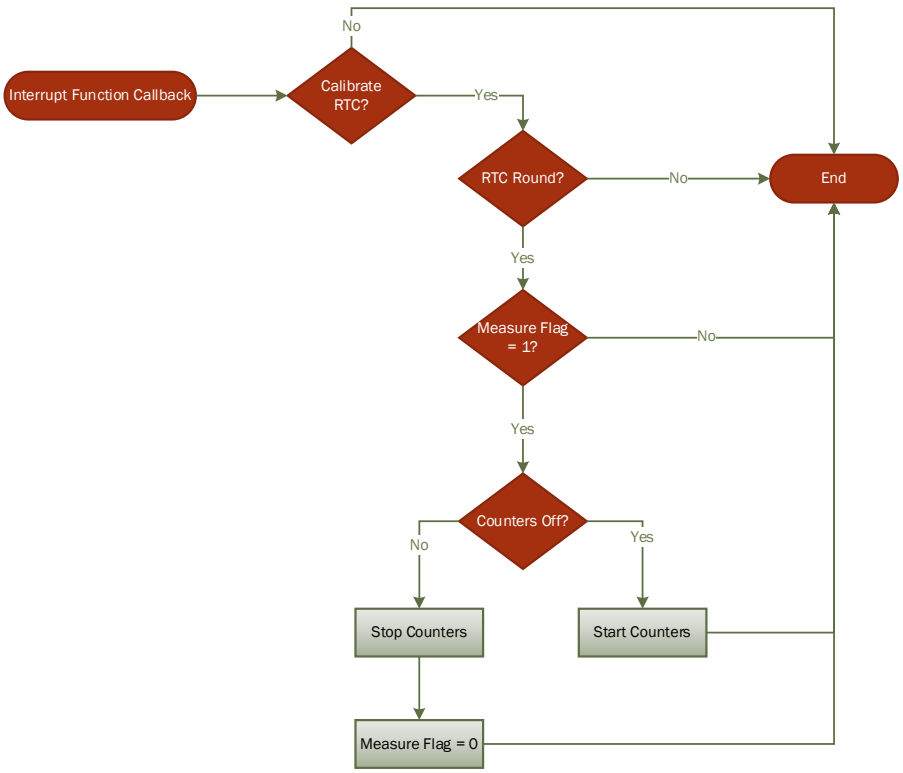

Figure 15. Measuring process flow chart inside RTC 1 Hz signal interrupt function.

4.4 Optimized ARM DSP Functions

Notably, the function for calculating magnitude was not optimized, requiring an average 0.75 milliseconds. Initially, generic C functions were used for floating point arithmetic, although these functions in generic C libraries are considered too slow and not optimized for a specific platform. ARM provides a DSP library that is optimized for some select microcontrollers, among them the on utilized for *i*VCCS 2ndG. ARM DSP cores provide 0.62 milliseconds improvement. Code is available in Appendix A.

4.5 Text Formatting Function

Text formatting required additional time — up to 3.5 millisecond. C-style text formatting function, namely sprintf, were used to convert floating point data into ASCII characters writable for a text file. To mitigate delay, float data were converted to 16-bit data, and an optimized function for converting integers to ASCII based on Terje Mathisen's algorithm was developed (See algorithm in Appendix A). Delay was reduced to 0.1776 milliseconds for single number conversion and 0.5625 milliseconds for complete reading from the magnetometer. Throughput of data collection was improved.

Chapter 5 System Algorithms and Power Analysis

Previous chapters have discussed how some algorithms were changed from 1st to 2ndG, porting them from the former to the latter 32-bit platform. In essence, the algorithms are identical, although two natures imposed a necessity for changing the operation to ensure accurate functionality. Another important aspect of *i*VCCS 2ndG features low-power operation. Notably, storage and communication are two of the most energy-hungry components of an embedded system. In this chapter, algorithms developed for mitigating power consumption induced by the microSD card, ZigBee, and continuous data fetching from the magnetometer KMX62 are discussed. An analysis of microSD card and ZigBee current consumption for the proposed algorithms is presented, as well as an estimate of system battery life at this stage of processing.

5.1 Data Buffering Technique

As indicated in Section 3.6, microSD card is used to log timestamps and store other status messages and raw data. However, microSD cards are energy inefficient and counter intuitive to the low power paradigm due to the fact that they simply dump data and messages directly to the card. According to SanDisk microSD card specifications [34], Read and Write procedures can take up to 100 mA in current consumption. Turning the card on/off when data must be stored is impractical due to required time delay for repeated card initialization each time the card is powered on. Specifications also refer to the automatic sleep feature wherein cards enter sleep mode, given no commands are received within 5 ms. In this mode, cards consume 350 μA. This indicates inefficiency for system duration. Consequently, the on-board ultra-low power flash memory is utilized to buffer data before shifting it to the microSD card, primarily

because the memory excels in energy efficiency and read/write performance when compared to the microSD card. The result is suitable for instantaneous data logging.

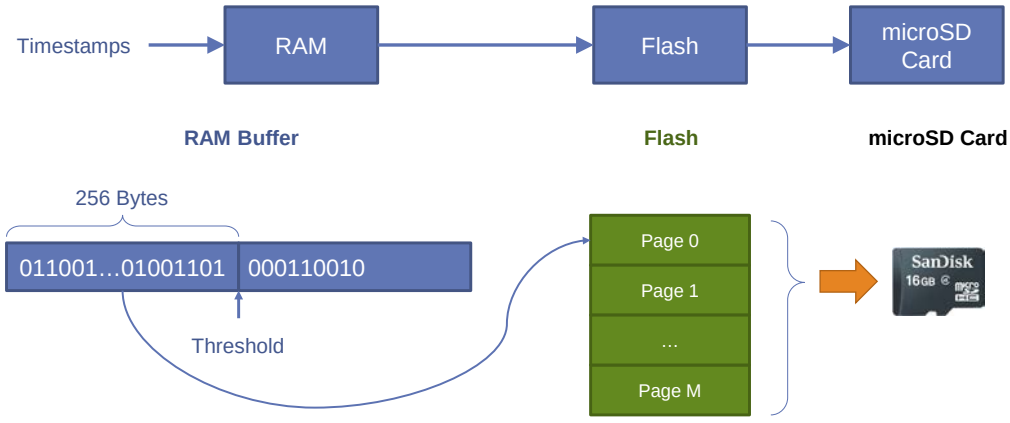

Figure 16. Block diagram illustrating data buffering technique.

MX25R64 exhibits a page basis write functionality, in which page size is 256 bytes. Host can commence reading at any byte address, although it can only write at the beginning of a page address. Hence, microcontroller must retain length of data required for the write function to avoid creating gaps in the flash memory. Figure 16 illustrates the procedure and Figure 17 provides a high-level description of the algorithm. C implementation can be found in Appendix B.

```
1:  IF New Sector THEN
2:      Erase Sector
3:  END IF
4:  RAM Buffer ← Data
5:  IF RAM Buffer Size ≥ 256 THEN
6:      Move data: Flash ← RAM
7:      Shift remaining data in RAM
8:  END IF
9:  IF Flash Full Pages ≥ Flash Threshold THEN
10:     Open new file in SD card
11:     FOR EACH Flash Page i DO
12:         SD card ← Flash page i
13:     END FOR
14:     SD card ← RAM Buffer
15:     Close file in SD card
16: END IF
```

Figure 17. High level description of data buffering technique,

5.2 Triggered Vehicle Detection

Typically, the magnetometer interrupts the microcontroller at any time in which a new sample is acquired in the buffer, in accordance with configured data rate (i.e., Data Ready Interrupt [DRI]). Desired behavior occurs only when an approaching vehicle interrupts MCU, thus, triggering execution of the detection algorithm. KMX62 magnetometer 2 is interrupted, as are DRI, Magnetometer Motion Interrupt (MMI), and Buffer Full Interrupt (BFI). MMI is issued when the difference between two consecutive samples on one axis reaches a programmed threshold in a specific direction (i.e., either increasing or decreasing) and stays above the threshold indicated for a

specific number of samples (i.e., time). The KMX62 buffer can operate in triggered mode and hold 64 samples of three components (x, y, z). Given a physical interrupt is caused a digital engine (i.e., magnetometer or accelerometer), a trigger event is asserted, and SMP_TH number of samples prior to the event are retained. Sample collection continues until the buffer is full. Data is reported in chronological order, as explained in Figure 18. MMI is configured to trigger the buffer, and BFI is routed to MCU through one of the GPIOs [27]. SMP_TH is set to 63, which causes KMX62 to immediately interrupt MCU subsequent to the reception of the first sample received after an MMI interrupt (i.e., vehicle approach/detection).

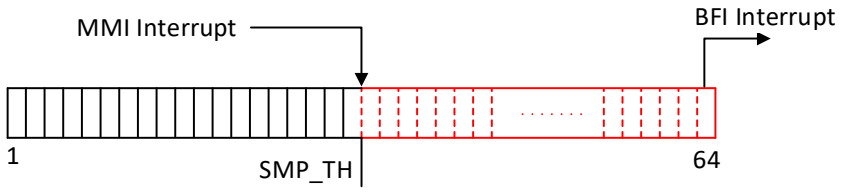

Figure 18. MMI and BFI operation in KMX62 magnetometer.

Figure 19 describes the way in which the algorithm reads and controls data flow from KMX62. MCU completes a dummy read for a number of samples, primarily because the first few samples might not relate to a vehicle's signature. The algorithm is configured to consider the last 24 samples prior to vehicle approach, discarding the first 40 samples from the buffer. After 24 samples are read and processed and given the state machine of the system still indicates detection, the MCU continues pulling new samples through Data Ready Registers until vehicle has departed.

```
1:  IF DRI Flag THEN
2:      Read data from DRI registers
3:  ELSE IF BFI Flag THEN
4:      Disable BFI
5:      Discard 40 samples
6:      WHILE Samples Counter < 24 THEN
7:          Read data from Samples Buffer
8:          Increase Samples Counter
9:      END WHILE
10:     IF State = Detection THEN
11:         Enable DRI
12:     ELSE
13:         Enable BFI
14:     END IF
15: END IF
```

Figure 19. High level description of triggered vehicle detection algorithm.

5.3 Communication Scheme

The primary role of the RF in this system is to report data (i.e., count and timestamps) to an access point (AP). This function is complete once each day, typically at midnight. However, the system is also capable of interacting with user requests. Notably, the ZigBee module is one of the most power-consuming units on board, and constantly turning it on will quickly deplete the battery.

MCU is not required to use RF interface, except for initial status reporting after sensor is initialized and data reporting at midnight. To accommodate user/AP requests, MCU must activate the ZigBee module for one minute, during which time it sends a status message reporting battery charge level and number of vehicles counted. Given that a command is received within this minute, the timer is reset, allowing for additional

commands. If none are received, RF module is shut down. Implementation is described in Appendix B.

5.4 microSD Card Power Analysis

In this section, microSD card power consumption and its effect on battery life are analyzed. Data written to a vehicle signature file is considered for two scenarios: 1) a regular, non-optimized scenario, wherein the microcontroller continuously writes data coming from the magnetometer as non-stop; and 2) an optimized scenario, wherein the microcontroller writes data coming from the magnetometer only when a vehicle is in the vicinity, thus eliminating ambient magnetic flux.

5.4.1 Continuous Data Transfer

Each sample consists of three components, x, y, and z in floating-point format. The microcontroller first converts data to ASCII to be written in a text file. Resulting data is a 25-byte chunk for each sample read from the magnetometer, as shown in Figure 20.

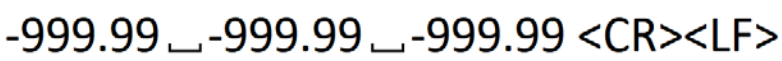

Figure 20. Data format for each sample.

Assuming that magnetometer sampling rate is 100 Hz, results for one hour (3600 seconds) are calculated, as follows:

$$3600 \times 100 \times 25 = 9 \times 10^6\ [Byte].$$

MicroSD card is interfaced with MCU via Serial Peripheral Interface (SPI) protocol. Interface clock is derived from the microcontroller system clock (8 MHz) with a pre-

scale of 2; therefore, SPI speed is 4 Mbit/s. Hence, time to write one sample to microSD card can be calculated as:

$$T_{sample} = \frac{25\ [byte]}{4\ [Mbit/s]\ = 524288\ [Bps]} = 47.68\ [\mu s]$$

Notably, MCU spends some time pulling and conditioning data from the magnetometer, as illustrated in Figure 21. Since sampling rate is 100 Hz, microcontroller waits 0.01 second for sample arrival and spends 3.5 ms for conditioning.

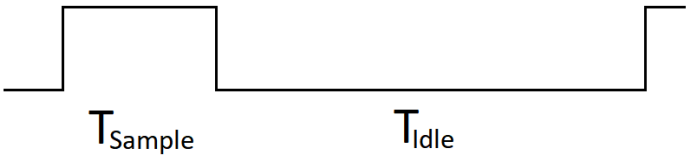

Figure 21. Timing diagram for writing to microSD card.

Total time for writing all bytes for one hour is determined by the following equation.

$$T_W^T = \frac{9 \times 10^6}{524288} = 17.1661\ [s]$$

Hence, hourly duty cycle for writing is:

$$DC_W = \frac{17.1661}{3600} = 0.004768 = 0.4768\%$$

Consequently, hourly duty cycle for idle time is:

$$DC_I = 1 - 0.004768 = 0.995231 = 99.5231\%$$

According to SanDisk datasheets [34], average current for writing to microSD card is 100 mA, and average current for idle (i.e., sleep) state is 150 µA. Average consumed current for write and idle states are calculated, as follows.

$$I_{W_{avg}} = 0.004768 \times 100\ mA = 0.4768\ [mAh]$$

$$I_{I_{avg}} = 0.995231 \times 150\ \mu A = 0.1493\ [mAh]$$

Total average consumed current is 0.6261 mAh.

5.4.2 Triggered Data Transfer

The same methodology can be applied to the second scenario for triggering data transfer. Assuming average speed of the vehicle is 65 mph and average vehicle length is 5 m, a sampling rate of 100 Hz would result in 18 samples for each vehicle. Assuming a length of 30 samples by taking into account samples before vehicle arrival and after departure:

$$Total\ number\ of\ bytes\ per\ vehicle = 30\ samples/vehc\ \times 25\ B/sample$$

$$= 750\ [B/vehc]$$

Therefore, time to write all bytes of a signature for one vehicle can be calculated as:

$$T_W = \frac{750}{524288} = 1.4305\ [ms]$$

Assuming traffic volume at a highway designated detection point is 1000 vehicles per hour, total time for write operation is determined by:

$$T_W^T = 1.4305\ [s]$$

Corresponding hourly duty cycle is determined by:

$$DC_W = \frac{1.4305}{3600} = 0.000397$$

Hourly duty cycle for idle time is, then:

$$DC_I = 1 - 0.000397 = 0.999603$$

Average consumed current can now be calculated in the following way.

$$I_{W_{avg}} = 0.000397 \times 100\ mA = 0.0397\ [mAh]$$

$$I_{I_{avg}} = 0.999603 \times 150\ \mu A = 0.1499\ [mAh]$$

Total average consumed current is 0.1896 mAh.

A reduction of 0.4365 mAh (or 69.6%) in consumed power is achieved using the new mechanism by refraining from writing ambient samples between vehicles arrivals/departures. It is worth noting that this number is further improved by reducing microSD card idle time. Buffering data through the serial flash is used for this purpose.

5.5 ZigBee Power Analysis

This section describes ZigBee power consumption when the module is mounted on the *i*VCCS sensor. First, a theoretical analysis of the consumed current based on the datasheet of the module (AW516P0) is presented [28]. Next, empirical data for three scenarios are provided, the first of which is a baseline, wherein all components are shut down and battery capacity is logged every minute for an hour. An intermittent transmission scenario with module going back to active mode is reported, and finally, the module is configured in receive mode.

5.5.1 Theoretical Analysis

To calculate module power consumption in various modes for different functionalities, scenario timing is required along with corresponding current consumed in each stage. Typically, these figures are provided by the manufacturer and published in their datasheets. The ZigBee module utilized in this research is based on NXP JN516 ZigBee platform, and the following analysis is based on their documentation.

Consider an example wherein the sensor continuously performs the following actions to send status beacons:

1. Start the module.
2. Receive data frame from the host CPU/MCU.
3. Perform Clear Channel Assessment (CCA).
4. Transmit data frame containing a payload of 60 bytes.
5. Transmission is complete; reception is off; module is active.
6. After 10 minutes, repeat from Step 2.

Duration and current corresponding to the aforementioned steps are determined below.

Module startup

JN516x devices start up using a fast RC oscillator before executing bootloader code at 26 MHz. After an additional 170-740 μs, the faster 32 MHz crystal becomes stabilized and a glitch-less switchover occurs. Application code is expected to wait until the crystal is stable for radio transmission, which occurs 230 μs later; thus, requiring **1 ms** from reset/wake event. During the time in which transceivers are not operating, the module is still considered in active mode, assuming the module is working at default clock rate 16 MHz. In this case, current consumed will be **4.98 mA**.

UART Data Transmission

According to the datasheet, receiving data from the host MCU consumes **5.04 mA**. The example used in this work assumes 60 bytes data payload. Hence, it is necessary to determine the time required for the UART peripheral to send the following frame:

$$60_{Byte} \times \left(8_{bits} + 1_{start\ bit} + 1_{stop\ bit}\right)/57600_{Baud\ rate} = 10.42\ [ms]$$

Each byte has a start bit and a stop bit — thus, 8+1+1. Default baud rate of the module is 57600 bps. Time taken to receive the frame is **10.42 ms**.

Performing CCA

Assuming a channel is clear after CCA and that random back-off period is two, the time required for executing the CSMA/CA algorithm in a non-beacon enabled network is: back off period = **0.96 ms**, and CCA period = **0.128 ms**.

The application operates throughout the back-off period, and transceiver remains on, even though it is neither transmitting nor receiving. Current drawn during this

period is **5.16 mA**. During CCA, the radio receiver is on and, therefore, a **20.28 mA** current is drawn.

Data Transmission

The medium access control (MAC) layer header size (H_{MAC}) is set to 25 bytes for 64-bit source and destination addresses. The physical (PHY) layer header size (H_{PHY}) is fixed and equal to 6 bytes. Payload frame size ($F_{Payload}$) depends on the amount of data transmitted — loaded up to 114 bytes. In the example reported a 60-byte payload was assumed:

$$T = \frac{H_{MAC} + H_{PHY} + F_{Payload}}{250\ kbps} \times 8 = \frac{25 + 6 + 60}{250\ kbps} = 2.912\ [ms]$$

Therefore, time required to transmit a data frame is **2.912 ms**, during which the transceiver consumes a current of **18 mA**.

Idle Time

Per the example provided, the sensor will refrain from transmission for **10 minutes**, during which time the transceiver remains on, even though there is no TX or RX. Current consumed by the ZigBee module is **5.16 mA**. Figure 22 illustrates the timing diagram of this process.

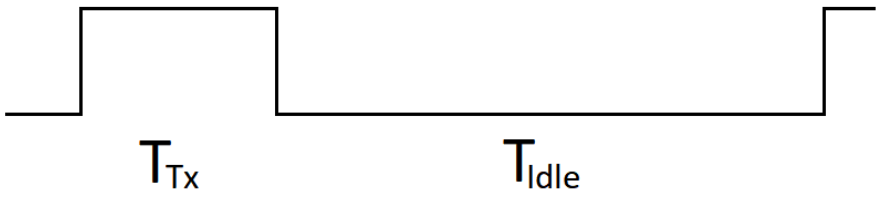

Figure 22. ZigBee transmit period (T_{TX}) and receive period (T_{Idle}).

It is necessary to calculate time and current consumed during the TX cycle.

Table 1. ZigBee Phases, Timings, and Currents

Phase	Current (mA) × Time (ms)	Charge (µC)
UART Data Transmission	$5.04\ mA\ \times\ 10.42\ ms$	52.52
CCA Back-off	$5.16\ mA \times 0.96\ ms$	4.95
CCA Period	$20.28\ mA\ \times 0.128\ ms$	2.6
Transmit Data	$18\ mA\ \times\ 2.912\ ms$	52.416
Total		**112.486**

Total time = **14.42 ms**. Average current consumed during transmission = Total Charge / Total Time = **7.8 µA**.

Since a transmission is assumed to occur once every 10 minutes, hourly duty cycle of RF transmission is given by:

$$DC_{TX} = \frac{0.01442 \times \left(\frac{60}{10}\right)}{3600} = 0.0024\%$$

Consequently, hourly duty cycle for idle time is:

$$DC_{Idle} = 100 - 0.0024 = 99.9976\%$$

Average hourly consumption can be calculated in the following way:

$$I_{TX} = 0.000024 \times 7.8\ \mu A = 0.0001872\ [\mu Ah]$$

$$I_{Idle} = 0.999976 \times 5.16\ mA = 5.16\ [mAh]$$

Total hourly consumption is:

$$I_{Total} = 5.16\ [mAh]$$

Notably, this value does not take into account the current drawn by the sensor or the self-discharge of the battery.

5.5.2 Empirical Data Collection

Data were collected for three scenarios: a baseline, wherein all components where powered down or put in sleep mode if there is no power switch (e.g. magnetometer/accelerometer), as well as TX and RX. For TX, a message is sent to AP once every 10 minutes. For RX, a message is sent from AP to the sensor mode once every 10 minutes. The following figure depicts battery capacity over time, logged every minute.

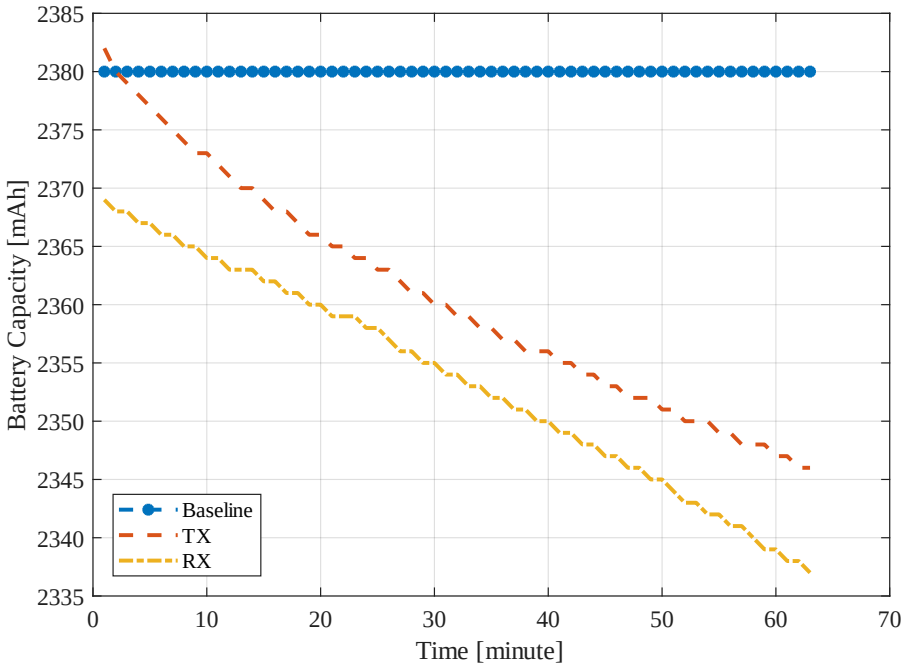

Figure 23. Power consumption of ZigBee TX and RX compared to baseline test.

Transmit consumption curve decays in a slightly steeper norm than receive mode. Therefore, caution should be taken when implementing the communication protocol. Obviously, ZigBee should not be constantly turned on; thus, a dynamic switching approach should be employed.

5.6 System Level Power Consumption Analysis

In this section the system's power consumption is empirically analyzed by dividing the sensor's one-hour cycle into states, and then measuring the amount of current drained in each state. Assuming the sensor is deployed on a highway with 65 mph speed limit and following distance of 1 second between vehicles, the result flow rate for a single lane will be 3089 passenger vehicles per hour. Assume flash threshold (See Section 5.1) is 10 pages (i.e., 2560 bytes), at which MCU buffers data from flash to microSD card. Sensor logs time of arrival (TA), time of departure (TD), and number of vehicles in the counter (N) for each detection. Additionally, sensor logs battery status once every minute. During a one-hour period, sensor will transition through the following states: battery status log, vehicle log, flash-to-microSD card buffering, status beacon, and standby.

a) `[BAT: VOLT = 4021 mV, CAP = 2480 mAh, SOC = 95 %]<CR><LF>`

b)
```
N01_TA@21225109.160156<CR><LF>
N01_TD@21225109.519531<CR><LF>
N01_N#73<CR><LF>
```

Figure 24. Example for battery status log line (a) and Vehicle timestamp (b). VOLT, CAP, and SOC are battery voltage, capacity, and state-of-charge, respectively.

During one hour, sensor will log 60 lines of battery status, each for 51 bytes. Figure 24 (a) shows an example line.

$$B_{bat} = 60 \times 51 = 3060\ [Bytes/hour]$$

For each detected vehicle, the timestamps and counter compose 54 bytes of logged data (or 56 bytes when counter is four digits), as shown in Figure 24 (b):

$$B_{veh} = 3089 \times 54 = 166806\ [Bytes/hour]$$

According to the aforementioned assumptions, the sensor will collect a total of 169,866 bytes during a one-hour time period. Since MCU moves data from flash to microSD card every 10 flash pages (2560 bytes), the result in 66 transfers.

Timings and drained current for each state is measured using high accuracy Fluke 289 multimeter [35]. Figure 25 depicts sensor current consumption for various states. Each data transfer is 1 second and consumes, at most, 30 mA.

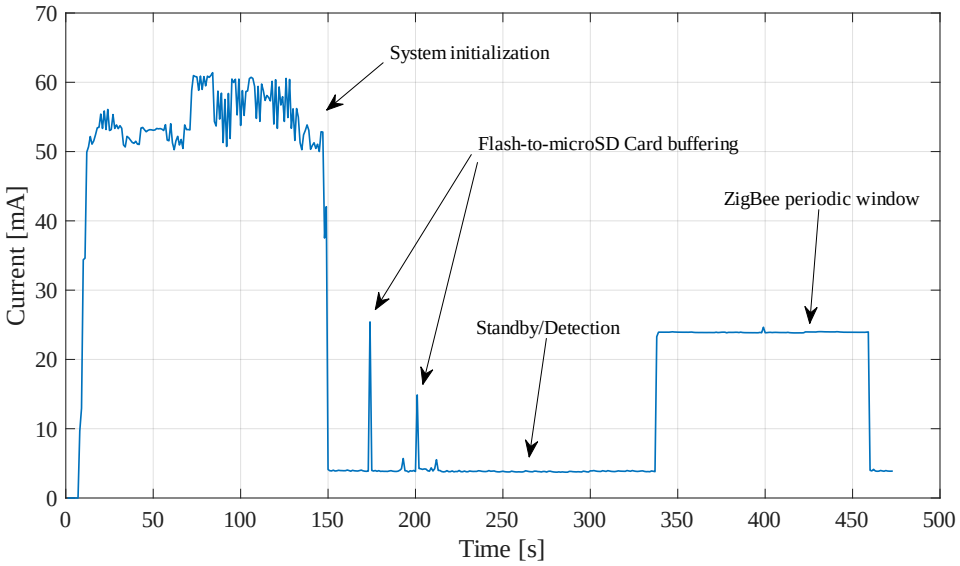

Figure 25. Current consumption at various states.

Each hour the sensor sends a status beacon message and activates the ZigBee module for one minute. Current drained is 24 mA. Sensor spends the majority of the time in standby state, consuming an average of 4 mA. Let's consider two scenarios: 1) sensor shuts down the ZigBee module after system initialization and does not incorporate the status beacon messages, and 2) status beacon messages used to periodically enable the ZigBee module. In the first scenario, average current for one hour is determined by:

$$I_{avg} = \frac{T_B I_B + (3600 - T_B) I_d}{3600} = \frac{66 \times 30 + 3534 \times 4}{3600} = 4.48\ [mA]$$

where T_B, I_B are buffering time and current, respectively, and I_d is the detection/standby current. Thus, the life of a 2300 mAh battery with up to an 80% derating can be calculated as:

$$T_{Bat} = \frac{2300}{4.48} \times 0.8 = 411\ [h]$$

In the second scenario, average current for one hour can be determined as:

$$I_{avg} = \frac{66 \times 30 + 60 \times 24 + 3474 \times 4}{3600} = 4.81\ [mA]$$

Therefore, the life for a 2300 mAh battery with up to an 80% derating factor can be calculated as:

$$T_{Bat} = \frac{2300}{4.81} \times 0.8 = 383\ [h]$$

Chapter 6 Reinforcement Learning for Power Management

The optimization methods and algorithms introduced in the previous chapters can be deemed hard-coded workarounds to minimize power consumption, as they lack flexibility and resilience for adapting to varying conditions. Although some policies lead to significant energy savings, cost to the system is response time and accuracy. Hence, proper orchestration between alternate methods and policies is necessary. With the problem of DPM, an algorithm controls power/performance tradeoff according to workload on the system. In this chapter, DPM and RL are described in an effort to further improve system power consumption.

6.1 Introduction to Reinforcement Learning

Reinforcement Learning (RL) is a branch of Artificial Intelligent (AI) algorithms that mimics a natural way of gaining knowledge and experience (i.e., learning through trial and error). Typically, this technique is tended to classify algorithms into one of two AI categories: supervised and unsupervised learning. Supervised learning agents are trained on a set of labeled data provided by an external expert, while unsupervised learning agents are trained on data that has no labels with a goal of finding hidden structure within unlabeled data [36]. Many researchers argue that RL falls somewhere between these two categories. In fact, RL can be considered an unsupervised method because it does not rely on expert correct-based examples of data sets [36]. Instead of finding hidden structures, RL attempts to maximize sparse rewards, based on what the agent must learn to behave in a given environment.

Although the concept is natural and intuitive, several challenges are characteristic of RL that are not characteristic of other types of machine learning (e.g.,

dilemma of exploration and exploitation). For example, should the agent *exploit* the knowledge gained by selecting actions to maximize its rewards? Or should the agent *explore* by selecting alternative untried actions for building a more realistic estimate and understanding of the environment? With regard to the credit assignment, how should one determine rewards and penalties associated with actions that lead to an arbitrary state?

The most common way of formalizing an RL problem is utilizing Markov Decision Processes (MDPs) [37], consisting of:

- finite state space S,
- set of available actions A,
- reward function $R: S \times A \rightarrow R$, and
- system dynamics function $P: S \times S \times A \rightarrow [0,1]$.

  This can be written as $P(s'|s,a)$ where:

$$\sum_{s' \in S} P(s'|s,a) = 1 \quad \forall\, s \in S, \forall\, a \in A$$

In other words, function P determines the probability of transitioning to state s', given that the agent remains in state s and performs action a. However, in an RL problem the agent has no initial or prior knowledge of the rewards function nor the system dynamics function.

Q-Learning is a standard RL algorithm, in which the agent's experience is expressed as sequences of states, actions, and rewards:

$$< s_0, a_0, r_1, s_1, a_1, r_2, \ldots >$$

In state s_0, the agent performed action a_0 and was rewarded r_1, resulting in transitioning to state s_1, and so on. Interactions with the environment form experience

the agent accumulates over time. The agent aims to maximize experience value — typically realized as a discounted future reward. In Q-Learning, this value is given by a function $Q^*(s, a)$ defined as the expected value of the cumulative discounted reward of performing action a in state s, and, hence, following the optimal path.

In general, the discounted future reward is defined as the immediate reward the agent receives for the current action taken and the future reward. Given a finite series of experiences:

$$< s_0, a_0, r_1, s_1, a_1, r_2, \dots, s_{n-1}, a_{n-1}, r_n, s_n >$$

total reward would be:

$$R = r_1 + r_2 + r_3 + \cdots + r_n$$

Therefore, future reward accumulated from time t onward is:

$$R_t = r_t + r_{t+1} + r_{t+2} + \cdots + r_n$$

However, due to the stochastic nature of the environment, the future reward for a given action is not guaranteed the same when chosen next. Thus, a discount factor is introduced to impart less significance on future rewards when compared a current reward, as shown below:

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{n-t} r_n$$

where $0 \leq \gamma < 1$ is the discount factor. If $\gamma = 0$, then the algorithm will be short-sighted, relying on only the immediate reward. If $\gamma = 1$, total future reward would be the same, thus, it would make sense if identical actions consistently reap the same rewards in a deterministic environment. A more balanced approach would be $\gamma = 0.9$. In this case, the discounted reward can be rewritten as:

$$R = \sum_{i=1}^{\infty} \gamma^{i-1} r_i$$

$$= r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots + \gamma^{i-1} r_i + \cdots$$

$$= r_1 + \gamma(r_2 + \gamma(r_3 + \cdots))$$

Let R_t be the reward accumulated from time t

$$R_t = r_t + \gamma\big(r_{t+1} + \gamma(r_{t+2} + \cdots)\big)$$

$$= r_t + \gamma R_{t+1}$$

Q-function will be discussed in more detail in subsequent sections of Chapter 6.

6.2 RL for *i*VCCS

Originally, the sensor exhibited an all-on policy, wherein all on-board components were turned on. This approach was not intuitive approach, and the sensor lasted barely 40 hours. A more flexible operation was added wherein certain components were turned on and off following a predetermined action flow. This resulted in reduced current consumption, as shown in Figure 25. To further enhance power consumption, sleep mode of ARM Cortex-M0 microprocessor was incorporated into the operation. However, this caused misdetections and/or double counting vehicles that had less than 2 seconds following distance as a consequence of long wake-up time from the sleep mode following the detection of a passing vehicle. Sensor power cycle can be divided into several phases, namely:

1. System initialization,
2. Data buffering,
3. ZigBee periodic communication, and
4. Standby/detection.

System initialization is a one-time phase and, therefore, has a negligible effect on long-term power consumption. Data buffering is part of the data-driven phase and related to the number of detected vehicles and the traffic flow. This phase was incorporated to reduce current consumed by microSD card and operation duty-cycle. ZigBee periodic communication was also implemented to reduce operation duty-cycle. Although sleep mode drains current at 2 mA, standby phase drains at 4 mA. The problem with sleep mode, as mentioned earlier, is that system response time increases and causes misdetections. Notably, in a low traffic condition — where vehicle following distance is more than 5 seconds — sleep mode is expected to perform faultlessly, as well as conserve more power. A DPM was introduced to solve the problem.

6.3 Problem Formulation

An RL algorithm based on Q-Learning was proposed as an approach to intelligently optimize the selection of power policies in a given state.

Assume the following:

- System can
    - observe the environment (i.e., traffic congestion) and
    - measure power consumption.
- Environment behavior can be modeled as high traffic (HT)/low traffic (LT).
- Power policies available to the agent (i.e., sensor) include:
    - High Power (HP) mode and
    - Low Power (LP) mode.

The system is modeled as a Markov Decision Process, with four-state space: High-Power High-Traffic (HP-HT), High-Power Low-Traffic (HP-LT), Low-Power High Traffic (LP-HT), and Low-Power Low-Traffic (LP-LT). Figure 26 shows the state

transition diagram of the system. Solid arrows are transitions made by the agent, and dashed arrows are transitions made by environment. Although agent is not penalized for transitions made by the environment, transitions should have high penalty in the reward matrix to prevent agent from taking such an action.

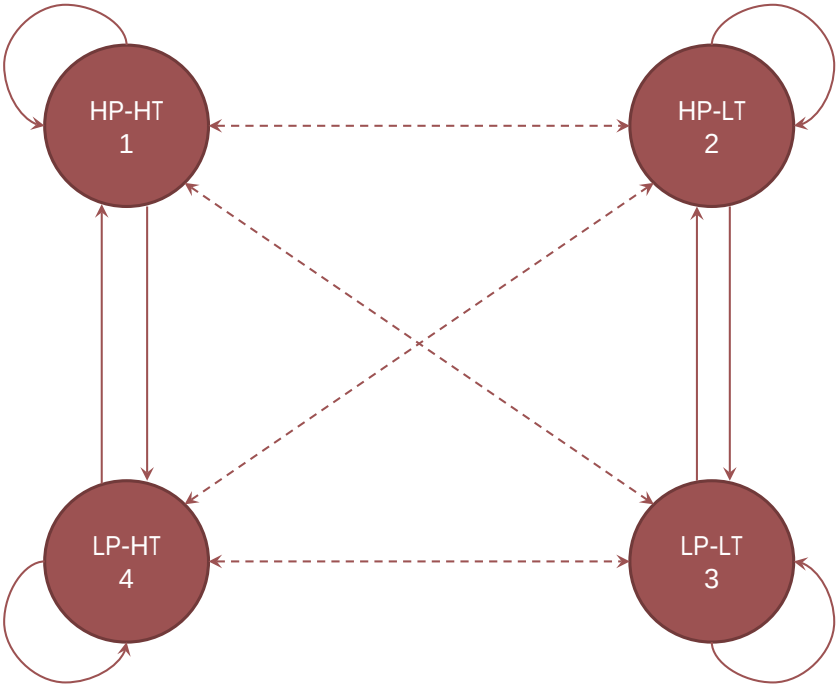

Figure 26. State transition diagram.

The agent (i.e., *i*VCCS) can switch to either high-power or to low-power mode. Reward function is defined as an R matrix, as indicated below.

$$R = \begin{bmatrix} 15 & -50 & -50 & -50 \\ -50 & -50 & 15 & -50 \\ -50 & -20 & 15 & -50 \\ 15 & -50 & -50 & -50 \end{bmatrix}$$

A tuple of $< s, a, r >$ forms an experience in the table. Q-function constructs the Q-table by giving the expected value of reward (i.e., Q-value). The Q-table can be either randomly initialized or zero initialized, as indicated below.

$$Q = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

6.4 Bellman Equation vs. Temporal Differences Equation

The proposed model consists of four states, as discussed earlier: HP-HT, HP-LT, LP-HT, and LP-LT. The agent can control its state in HP or LP. The environment controls sensor state through traffic, placing it in high or low traffic. Sensor should read information for either to determine its state. When the agent performs an action, a corresponding reward from R matrix is selected, and a new experience (i.e., tuple of state, action, reward $< s, a, r >$) is formed. This result is the Q-value $Q^*(s, a)$, which is the expected value of action a in state s, and then following optimal policy, defined as:

$$Q^*(s, a) = \sum_{s\prime} P(s'|s, a)[\mathcal{R}^a_{ss\prime} + \gamma V^*(s')]$$

where $V^*(s)$ is the expected value of following an optimal policy from state s. Bellman equation is used to estimate this value, as follows.

$$Q(s, a) = r + \gamma \max_{a\prime} Q(s', a')$$

where $\max_{a\prime} Q(s', a')$ is the maximum Q-value in the experience table (Q-table) for future state s' over all possible actions a'. Bellman equation provides the maximum future reward as the reward received for the action taken in the current state s plus the maximum future reward for the next state s'. In some RL problems, newer values of Q can be given additional weight to increase their influence as they are deemed more accurate. To weigh later experiences, the following Temporal Differences (TD) equation can be used:

$$Q(s, a) = Q(s, a) + \alpha \left(r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right)$$

- $0 < \alpha \leq 1$ determines the weight of newer values compared to older ones.
- γ is the discount factor.
- r is the reward corresponding to an action from R matrix.

Both cases were simulated to study their convergences. Figure 27 shows that Bellman equation converges faster than the TD equation by a factor of 10 times.

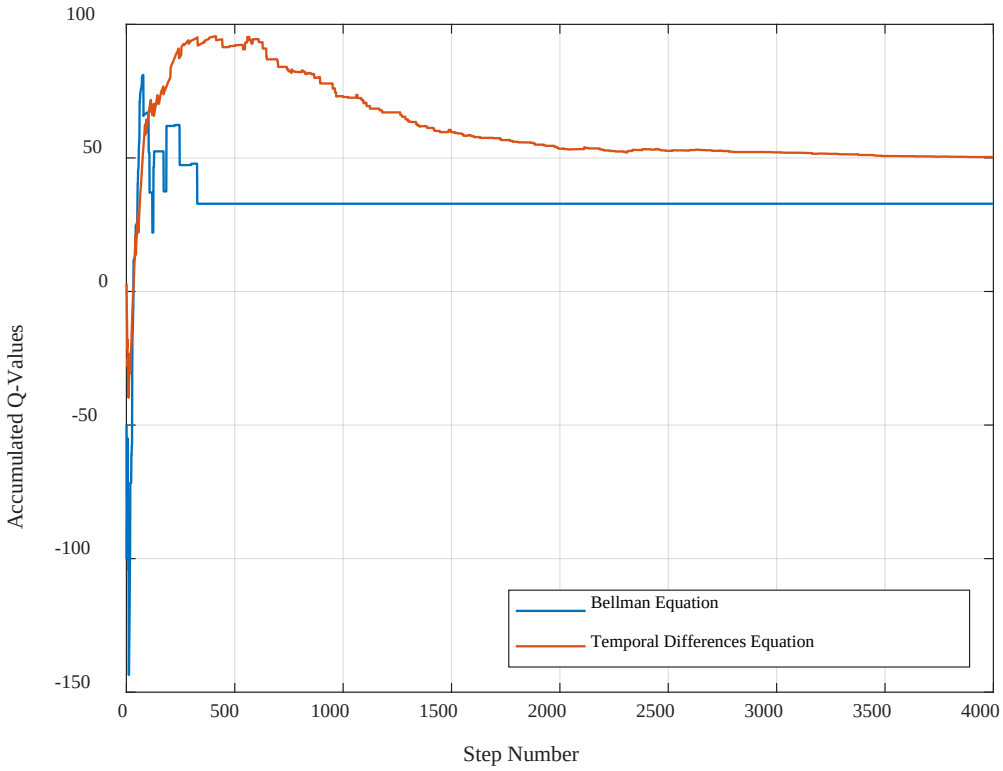

Figure 27. Q-Matrix convergence: Bellman vs. TD equation.

By looking at the normalized Q-matrix for both cases, it is clear that the agent was rewarded for going to the optimal states 1 and 3, which correspond to HP-HT, and LP-LT, respectively. Although it is not optimal, some rewards were given for transitioning into state 2 from state 3, since it does not affect the detection and counting accuracy at

the expense of power consumption. Figure 28 and Figure 29 depict the normalized experience matrices (i.e., Q-Matrix).

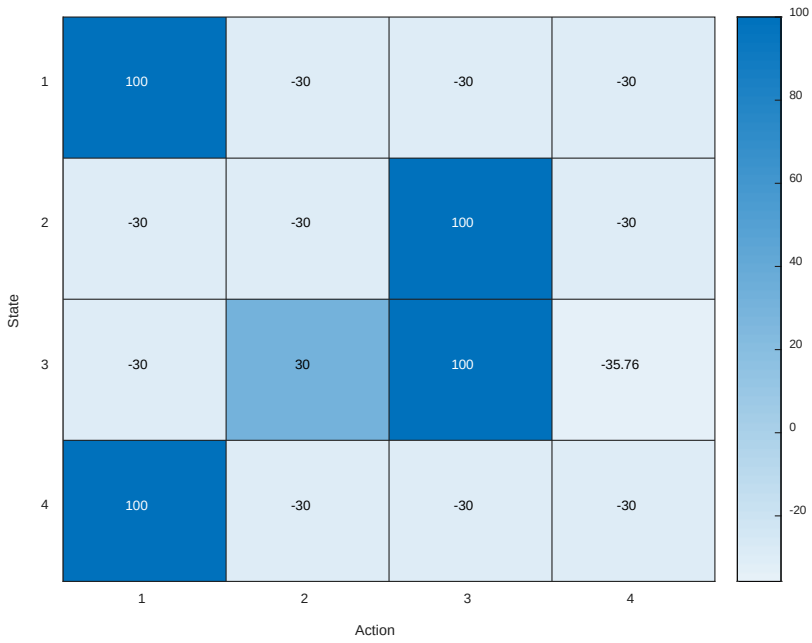

Figure 28. Normalized Q-Matrix for Bellman equation.

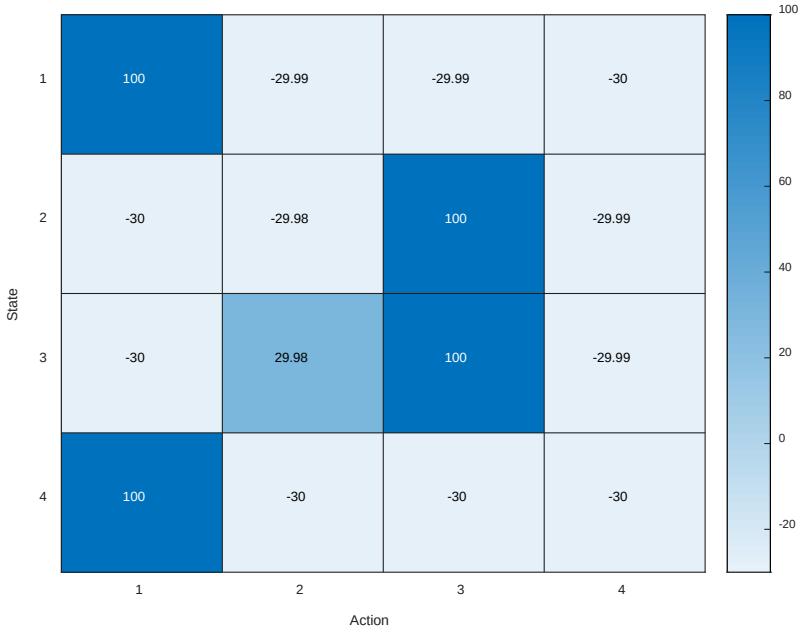

Figure 29. Normalized Q-Matrix for TD equation.

A small difference can be observed in the Q-Matrix and accumulated Q-values of TD equation. This is attributed to the fact that TD is a suboptimal solution to Bellman functional equation [38]. States and actions 1, 2, 3, and 4 refer to HP-HT, HP-LT, LP-LT, and LP-HT, respectively. Given the fast convergence of Bellman equation, it is prudent to be chosen as the system Q-function.

6.5 Reducing Number of Actions

In the previous section, the simulation was accomplished when agent had the option to move to any state. Notably, it was penalized for choosing actions that resulted in states controlled by the environment. In this section, the number of actions is reduced to Low-Power and High-Power.

This approach eliminates the possibility of choosing non-permitted actions such as the ones controlled by the environment. It also enhances convergence time by a

factor of half, compared to results obtained earlier. Figure 30 and Figure 31 show Q-Matrix convergence and Q-Matrix normalized.

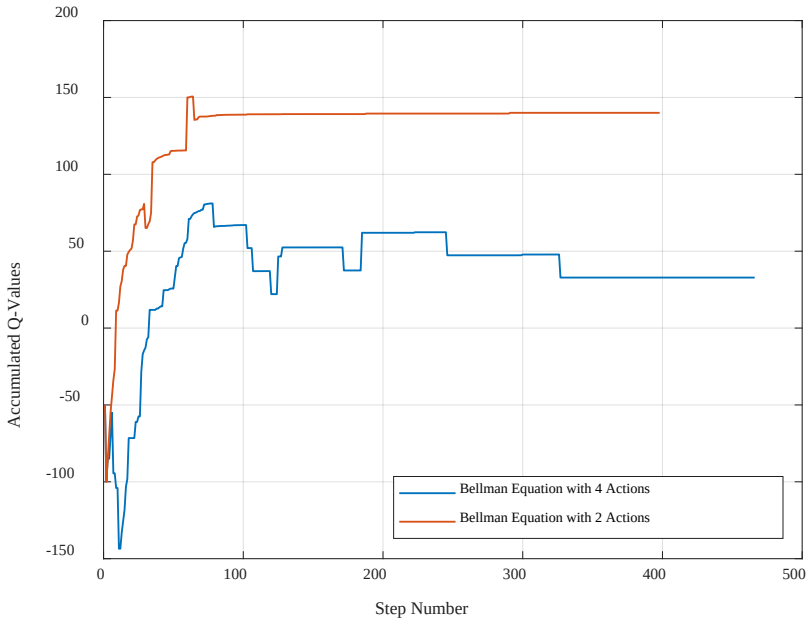

Figure 30. Bellman equation convergence: 2 actions vs 4 actions.

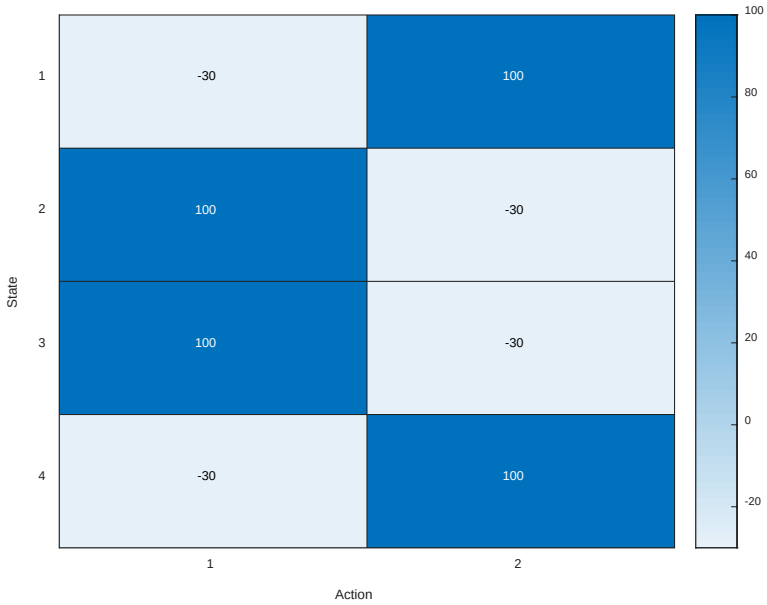

Figure 31. Q-Matrix using two actions.

6.6 Power Consumption Analysis

When employing an RL algorithm, circuit current drained by the system can be calculated utilizing a similar approach to the analysis detailed in Section 5.6. In this scheme, the algorithm places the microcontroller core in sleep mode during standby state, reducing current consumed to 2 mA. Given that a vehicle approaches the sensor zone, microcontroller experiences an interruption, exits sleep mode, and switches to detection state. Consequently, drained current becomes a function of the number of vehicles detected by the sensor. Figure 32 represents current consumed when a vehicle is detected. Vehicle time spent traveling over the sensor determines length of detection state, which was statistically calculated as an average 0.5 seconds.

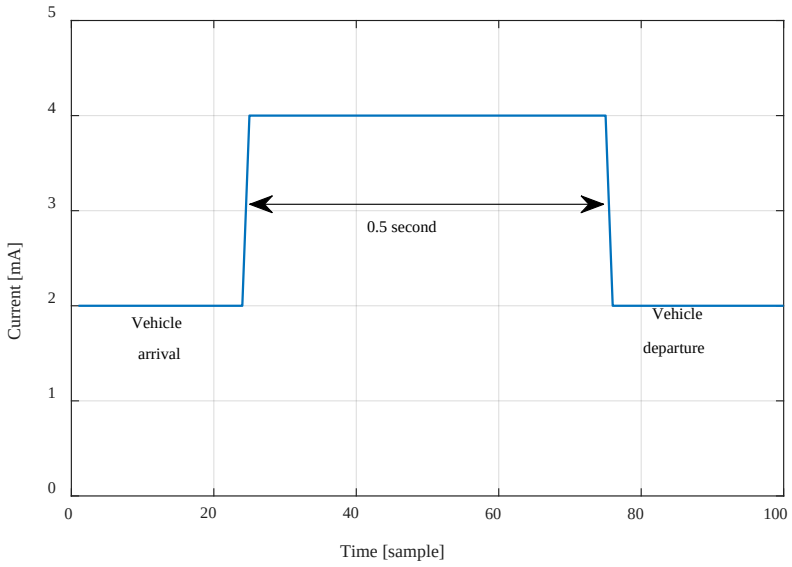

Figure 32. Current consumed when MCU wakes from sleep mode.

Acknowledging previous assumptions, average current equation in Section 5.6 can be rewritten to express current consumption, leveraging the RL algorithm, as shown below.

$$I_{avg} = \frac{T_B I_B + N_v T_v I_d + (3600 - T_B - N_v T_v) I_s}{3600} = 3.71\ [mA]$$

where N_v is number of vehicles detected per hour; T_v is the detection period; and I_s is the sleep state current. Given a 2300 mAh battery, battery life is extended, as follows:

$$T_{Bat} = \frac{2300 \times 0.8}{3.71} = 496\ [h]$$

6.7 Simulation using Real-World Data

Simulations detailed above were provided with synthetic data, where the agent is randomly assigned a state and is required to take appropriate action. This process was repeated over 5000 episodes, and each episode commences in a random state and terminates when the agent reaches the goal state.

To simulate real-word scenarios, a code was developed that uses the same sensor (*i*VCCS 2ndG) to combine vehicle timestamps collected during past deployments for forming the state of the agent using two variables — power policy (LP or HP) and traffic trend (HT or LT). Traffic trend is defined as vehicles detected per minute. A threshold of 10 vehicle/min separates high from low traffic. The algorithm was tested using two data sets — one collected on campus for 24 hours and another on Britton Highway for 3 hours. Power policy 1 represents low power; 2 represents high power. The following figures show the traffic trend and power policy chosen over time for both data sets. Since campus traffic is rather low, the agent chose to keep power policy on low (i.e., 1). High traffic on Britton Highway caused the agent to maintain high power (i.e., 2). The switch between HP and LP can be observed at the beginning and end of the graphs for Britton Highway, on which traffic increases and decreases due to lane closure by traffic control for installation.

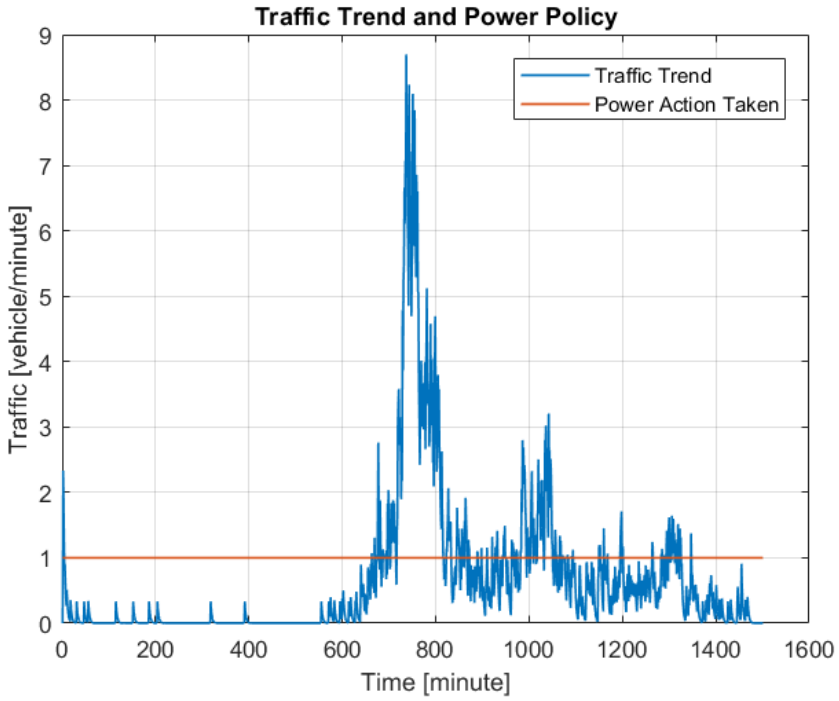

Figure 33. Traffic trend and power policy on campus.

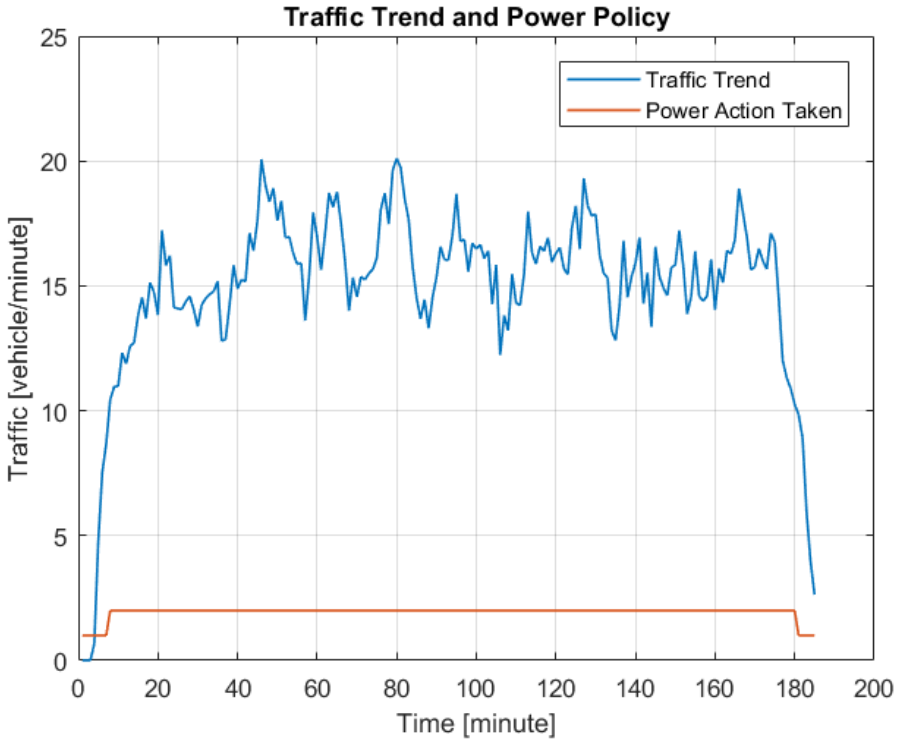

Figure 34. Traffic trend and power policy on Britton Highway.

Chapter 7 Experiments and Results

7.1 Detection Algorithm Validation

*i*VCCS 2ndG was deployed on Britton Highway to test functionality and to verify the algorithm worked as expected in a real-world scenario. Magnetometer sampling rate was configured to 100 Hz, and the sensor was configured to collect timestamps of vehicle arrivals and departures, as well as magnetic signatures. Battery capacity was logged every minute to study power consumption of regular polling-based detection and triggered vehicle detection. A camera was set up to record all vehicles for ground-truth validation. Figure 35 shows the deployment setup.

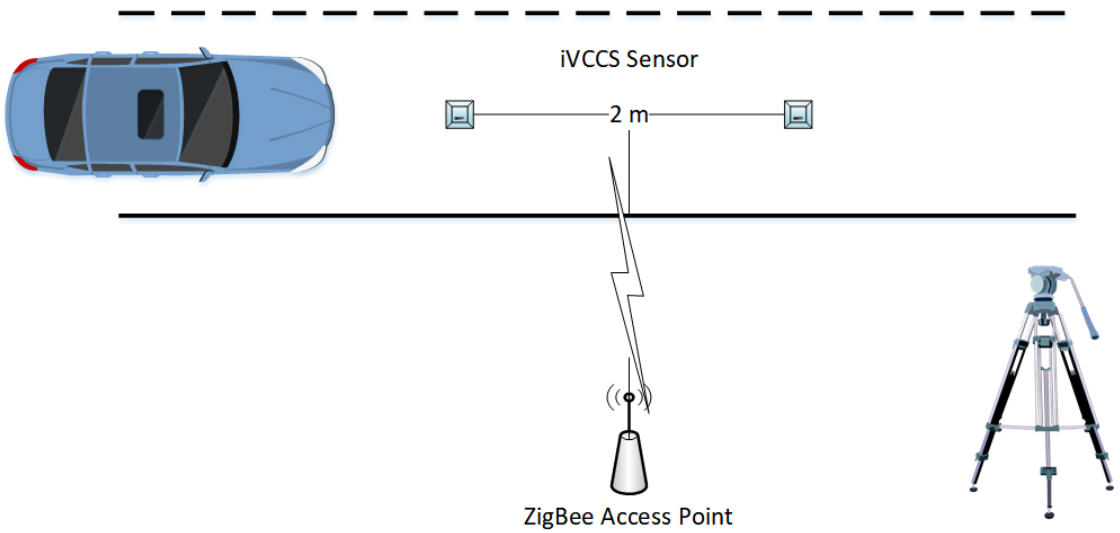

Figure 35. *i*VCCS setup on Britton Highway.

Sensors were deployed at 11:26 AM for two hours. The first sensor reported 1765 detected vehicles; the second detected 1756. During the same time period, video

recording captured 1766 vehicles. Separation distance between the two sensors was 2 meters. Difference in the number of detected vehicles between the two sensors resulted from cars changing lanes while traveling over the sensors. In general, an accuracy of more than 99% was achieved for vehicle detection.

Power information were collected during the deployment (e.g., battery capacity, battery voltage, and state of charge) and logged every minute. A third sensor was also deployed on the same lane as the other two for testing the triggered detection mechanism (See Section 5.2), which notifies the microcontroller only when a vehicle approaches and passes through the sensible magnetic field of the *i*VCCS. The following figure compares power consumption of two sensors — one with triggered detection and the other without — for a period of over 180 minutes.

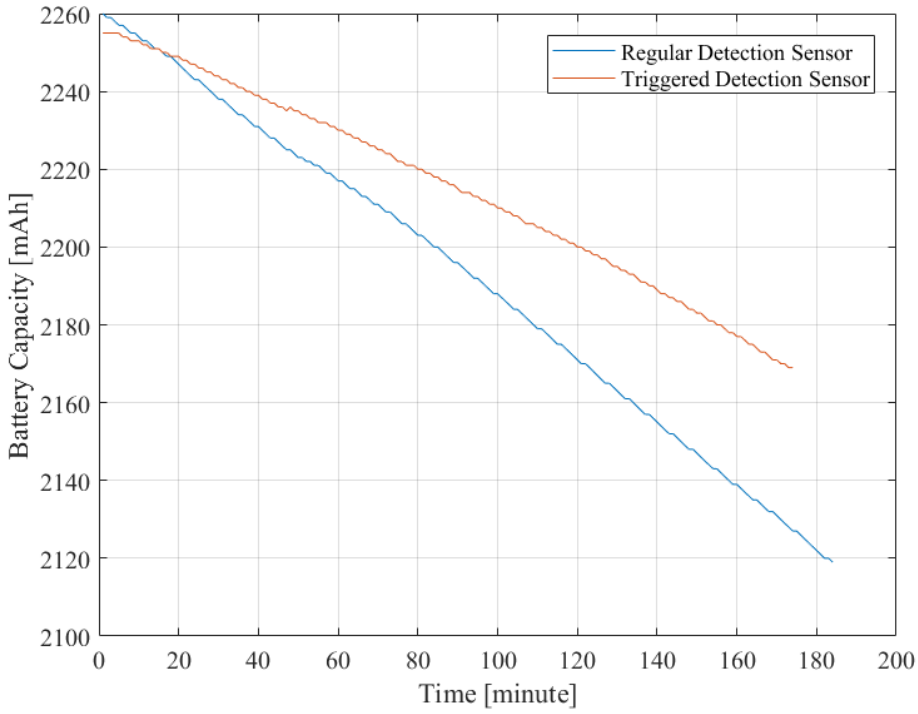

Figure 36. Battery capacity of regular and triggered detection sensors over a 180-minute time period.

As the figure illustrates, depletion rate for the triggered detection sensor is lower than regular detection. During the 180-minute period, regular detection firmware consumed 133 mAh; triggered detection consumed 86 mAh, resulting in a 35% improvement. It is worth noting that this enhancement was achieved only by stopping the detection algorithm when there's no vehicle in the vicinity. The microcontroller was still in active mode, and all other peripherals (i.e., RF transmission, microSD card, UARTs, and others) were switched on.

7.2 Power Optimization Validation

The system was tested in two scenarios: a lab test using a train continuously running for 24 hours and a field test wherein two sensors were deployed at the south entrance of campus for 24 hours. The sensor captured vehicle count and speed estimates for individual cars. Reported speeds harmonized with the expectations and nominal values of a real-world setting.

7.2.1 Lab Test

In the lab test, a sensor was placed under a miniature train track with a train operating for 24 hours at varying speeds. Figure 37 illustrates the setup. The detection algorithm was validated, and battery life was examined. Results were compared with the empirical analysis detailed in Section 5.6.

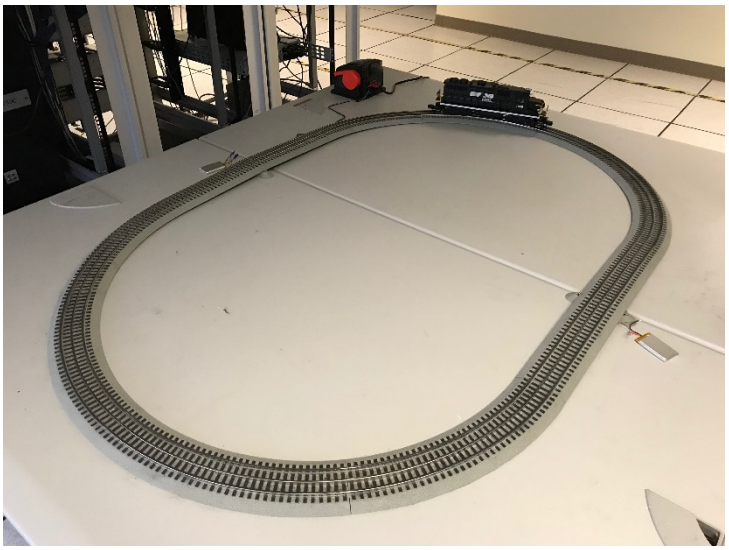

Figure 37. Lab test using a train and two sensors placed under the track.

Figure 38 lists the frequency at which the sensors detected the train, along with reported speed, which was calculated using distance between the two sensors.

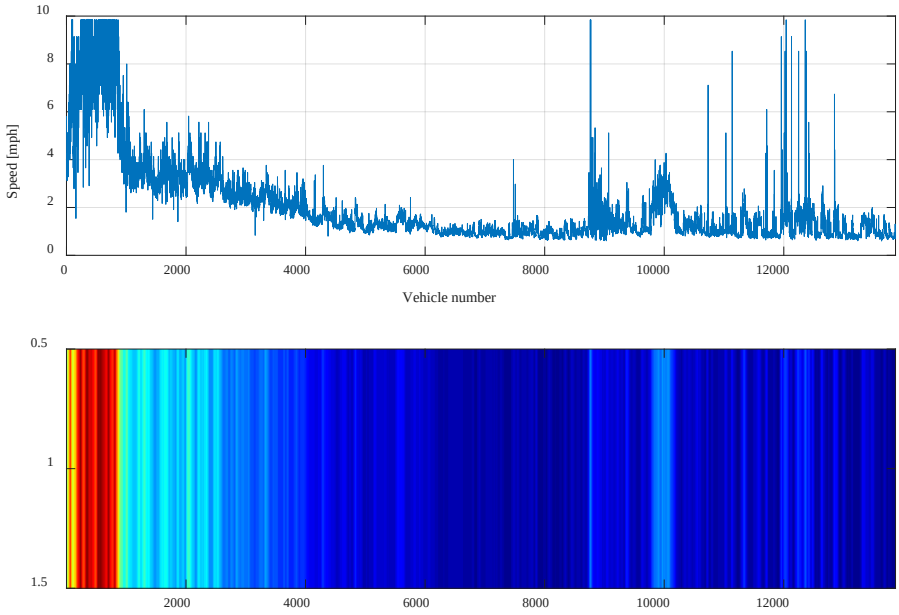

Figure 38. Lab test: Speed estimates of the train for a time period over 24 hours.

The sensor consumed only 11 mAh in a time period over 24 hours (i.e., 0.458 mAh per hour) as Figure 39 shows, which is 10 times less than predicted in the analysis described in Section 5.6.

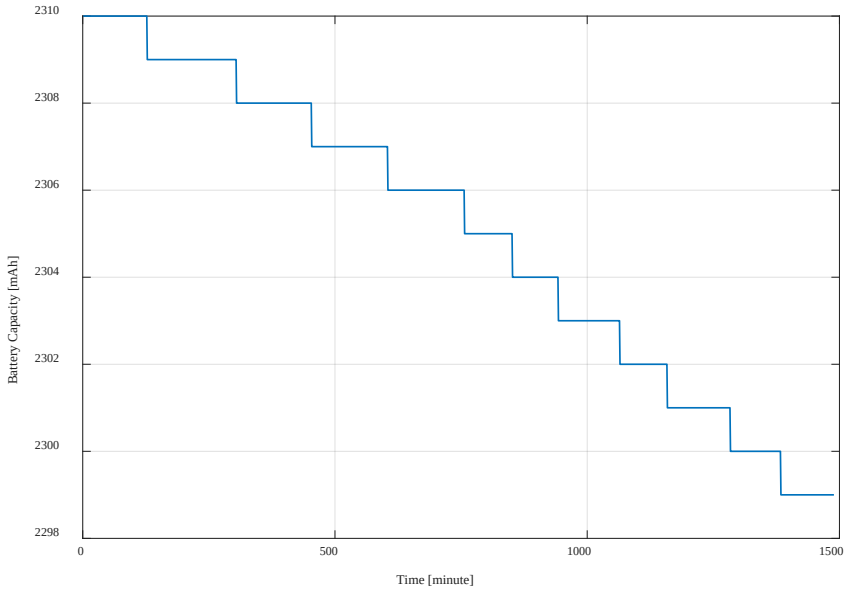

Figure 39. Lab test: Battery capacity of the sensor for a time period over 24 hours.

7.2.2 On-Campus Test

For the field test, two sensors were deployed at the south entrance of campus: one with the proposed DPM algorithm managing the power policy and the other running a plain version of the firmware. Figure 40 depicts the sensors' locations. Sensors were deployed for 24 hours, collecting the number of vehicles that entered the campus, time of arrival and departure, as well as battery capacity for comparing power consumption of both versions of firmware. Speed was calculated in a postprocessing stage, wherein detected vehicle timestamps were used with known separation distance (e.g., 2 meters) between sensors.

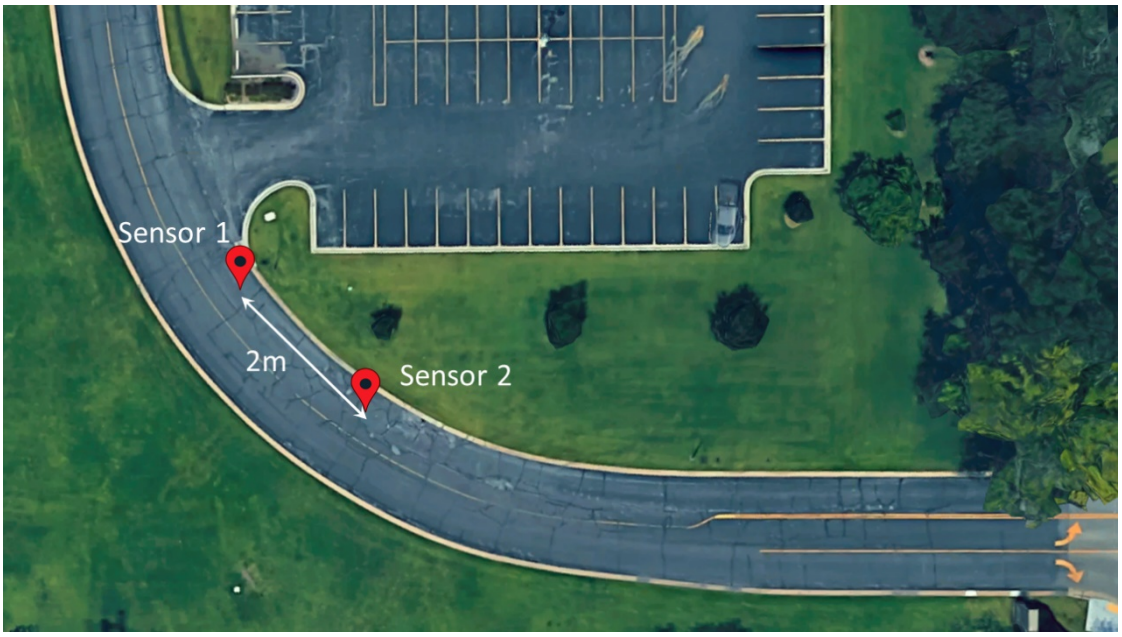

Figure 40. Campus field test setup.

Average speed was 6 mph, which is fairly logical given sensor location at the campus entrance and speed limit of 10 mph. Processing the collected power data revealed that the sensor running original firmware (sans DPM) consumed approximately 18 mAh, resulting in a battery life of over 100 days. Alternatively, the sensor executing the DPM algorithm consumed only 4 mAh over 24 hours, indicating a battery life of over 400 days. Both estimates assumed a 2300 mAh battery with a derating factor of 80%. The original sensor detected only 11 more vehicles (2.9%) than the sensor executing the DPM algorithm.

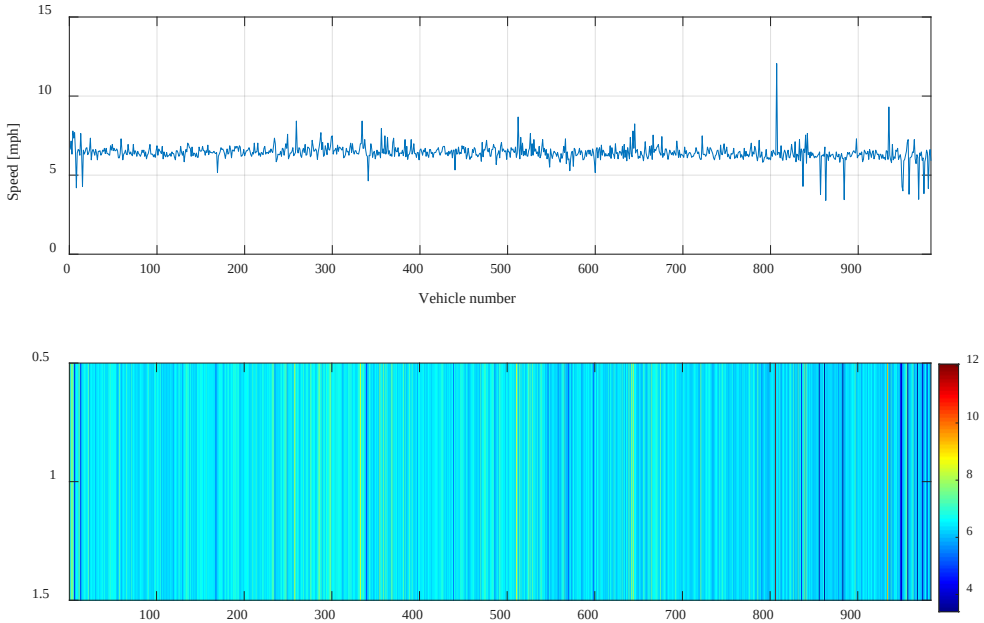

Figure 41. Campus field test: Number of vehicles detected and speed estimates.

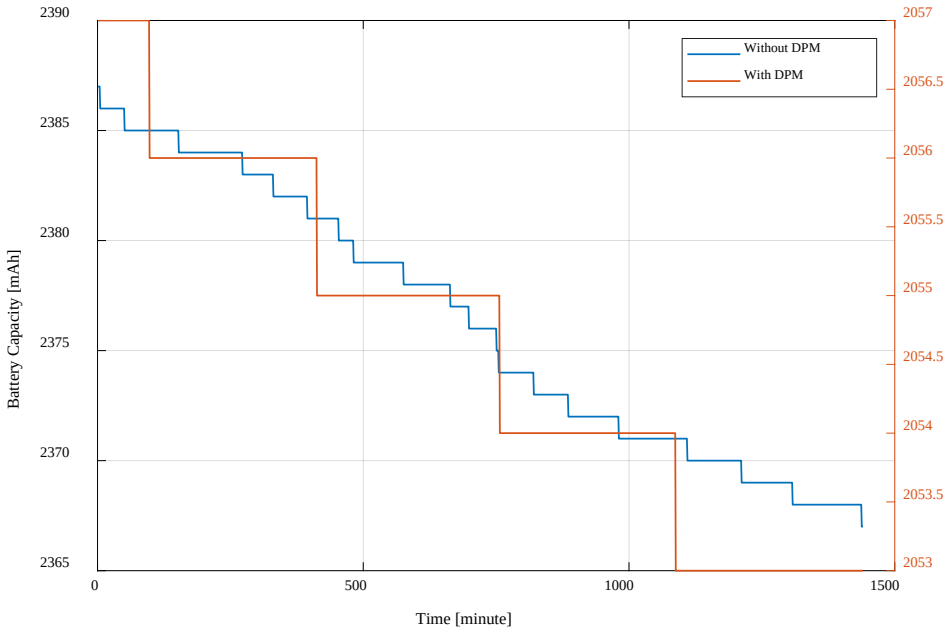

Figure 42. Battery capacity of sensor with and without DPM for a time period over 24 hours.

Chapter 8 Conclusion and Future Work

8.1 Conclusion

This research presented algorithms and methods to optimize system response and reduce power consumption of an *i*VCCS designed by the WECAD (Wireless and Electromagnetic Compliance and Design) Center at the OU-Tulsa campus in Tulsa, OK. The first prototype of the sensor adopted an all-on policy, in which all components were active, thus depleting battery life in nearly 48 hours.

The second generation of the system (*i*VCCS 2ndG) was built around a more powerful processing platform utilizing a 32-bit microcontroller. The first-generation firmware was ported to the new platform in C language to accommodate robustness and speed, which is critical for an improved response time without an increase in power consumption. As part of the process, some algorithms were modified, while others were introduced to shift the operation methodology from a polling mode to an event-driven, interrupt-based procedure, wherein the system responds to interactions from the surrounding environment.

Two operation modes (i.e., HP and LP) were designed to leverage a tradeoff between consumed energy and response time, which affects detection accuracy relative to traffic flow. An RL approach was proposed to design a DPM algorithm for observing traffic and controlling the system's power policy according to the environment and agent states.

An analysis of energy utilization by communication and storage subsystems were performed, in addition to a system level study. A conservative analysis revealed a

battery life of approximately 18 days. Experimental results showed that overall battery life of the system was extended to over 200 days for a 2300 mAh battery.

8.2 Future Work

*i*VCCS 2nd G design includes a port to host energy harvesting technology. Several energy sources have been considered by the research community; harvesting technology is also a hot topic in the IoT domain. The final goal for the project is designing self-powered, maintenance-free, wireless sensor nodes. Table 2 [39] shows a comparison of energy sources and the power outputs for typical energy scavenging devices based on published studies and experiments.

Table 2. Comparison of energy sources

Energetic Source	Power Density
Solar (outdoors)	15 mW/cm^2 (direct sun)
	0.15 mW/cm^2 (cloudy day)
Vibrations	0.01-0.1 mW/cm^3
Thermoelectric – 10 °C gradient	40 μW/cm^2
Acoustic noise	3×10^{-6} mW/cm^2 at 75 dB
	9.6-4 mW/cm^2 at 100 dB
Passive human-powered systems	1.8 mW (shoe inserts)

In general, three sources of energy can be distinguished from the ambient environment: Photovoltaic Cells (PV), Mechanical Vibration (MV), and Electromagnetic Generators (EG) [40], [41]. PV cells are perhaps the most common energy harvesting technique for WSNs. This fact is not a big surprise, as it is well-established in the literature [42], [43]. Advancement in this field has reached a level in

which systems can be solely powered by PV cells sans energy storage or converters [44]. Although such an approach is not feasible for experiments like those detailed in this research, the example provided assures that an ample amount of energy can be harvested from solar power. In view of the importance and efficiency of PV cells, the *i*VCCS 2ndG is designed to connect a solar panel for energy harvesting.

Literature suggests the use of mechanical vibration as a viable source of energy when a sufficient mechanical excitation is available [40], [41], [45]. Generally, this kind of energy can be harvested using three main approaches, namely piezoelectric, electromagnetic, and electrostatic generators — each characterized by advantages and disadvantages [45]. Given the target environment of our sensor (e.g., vehicles with more than 2 axles pass on highways), the vibration generated by the vehicles could be a significant amount of harvested energy by a sensor node. Likewise, using electromagnetic flux for vehicle detection can be utilized for scavenging power, as explained in [45].

Another widely used type of energy is RF, especially given the current ubiquitous deployments of wireless APs. Although results hold true in environments characterized by urban streets, residential and shop buildings (e.g., dense in some cases) are increasing. Thus, it might end up that results do not hold true on highways, on which major dependence would be placed on sub-GHz frequencies. In [46], power measurements in an urban scenario revealed that maximum power levels were observed in the GSM-800 band, while much lower power levels were recorded in the GSM-1800 band due to propagation loss. Hence, the proposed technologies should be studied and evaluated in light of the operating environment intended for *i*VCCS.

References

[1] W. Vereecken *et al.*, "Power consumption in telecommunication networks: overview and reduction strategies," *IEEE Communications Magazine*, vol. 49, no. 6, pp. 62–69, Jun. 2011.

[2] ITS (Intelligent Transportation Society of America), "Annual Report 2010-2011," 2011.

[3] ITS (Intelligent Transportation Society of America), "Rise of the Real-Time Traveler," 2015.

[4] Cisco, "Cisco Connected Roadways – IoT in Roads, Traffic, Transit - Cisco." [Online]. Available: https://www.cisco.com/c/en/us/solutions/industries/transportation/connected-roadways.html. [Accessed: 14-Apr-2018].

[5] L. E. Y. Mimbela and L. a Klein, "A Summary of Vehicle Detection and Surveillance Technologies used in Intelligent Transportation Systems."

[6] M. T. Alamiri, "IoT Systems for Travel Time Estimation," University of Oklahoma, 2017.

[7] W. Balid, "Fully Autonomous Self-Powered Intelligent Wireless Sensor for Real-Time Traffic Surveillance in Smart Cities," University of Oklahoma, 2016.

[8] W. Balid, H. Tafish, and H. H. Refai, "Intelligent Vehicle Counting and Classification Sensor for Real-Time Traffic Surveillance," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–11, 2017.

[9] W. Balid and H. H. Refai, "Real-Time Magnetic Length-Based Vehicle Classification: Case Study for Inductive Loops and Wireless Magnetometer

Sensors in Oklahoma State," 2018.

[10] Y. Guyodo and J.-P. Valet, "Global changes in intensity of the Earth's magnetic field during the past 800[thinsp]kyr," *Nature*, vol. 399, no. 6733, pp. 249–252, 1999.

[11] NASA, "2012: Magnetic Pole Reversal Happens All The (Geologic) Time," 2015. [Online]. Available: https://www.nasa.gov/topics/earth/features/2012-poleReversal.html. [Accessed: 15-Apr-2018].

[12] N. Wahlstrom and F. Gustafsson, "Magnetometer Modeling and Validation for Tracking Metallic Targets," *IEEE Transactions on Signal Processing*, vol. 62, no. 3, pp. 545–556, Feb. 2014.

[13] C. Park, P. H. Chou, Y. Bai, R. Matthews, and A. Hibbs, "An ultra-wearable, wireless, low power ECG monitoring system," in *IEEE 2006 Biomedical Circuits and Systems Conference Healthcare Technology, BioCAS 2006*, 2006, pp. 241–244.

[14] M. T. Lazarescu, "Design of a WSN platform for long-term environmental monitoring for IoT applications," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 3, no. 1, pp. 45–54, 2013.

[15] O. Olorode and M. Nourani, "Reducing leakage power in wearable medical devices using memory nap controller," in *2014 IEEE Dallas Circuits and Systems Conference: Enabling an Internet of Things - From Sensors to Servers, DCAS 2014*, 2014, pp. 1–4.

[16] B. Martinez, M. Monton, I. Vilajosana, and J. D. Prades, "The Power of Models: Modeling Power Consumption for IoT Devices," *IEEE Sensors Journal*, vol. 15,

no. 10, pp. 5777–5789, Oct. 2015.

[17] I. Joe and M. Shin, “Energy management algorithm for solar-powered energy harvesting wireless sensor node for Internet of Things,” *IET Communications*, vol. 10, no. 12, pp. 1508–1521, Aug. 2016.

[18] P. Kamalinejad, C. Mahapatra, Z. Sheng, S. Mirabbasi, V. C. M. Leung, and Y. L. Guan, “Wireless energy harvesting for the Internet of Things,” *IEEE Communications Magazine*, vol. 53, no. 6, pp. 102–108, 2015.

[19] K.-Y. Chan, H.-J. Phoon, C.-P. Ooi, W.-L. Pang, and S.-K. Wong, “Power management of a wireless sensor node with solar energy harvesting technology,” *Microelectronics International*, vol. 29, no. 2, pp. 76–82, May 2012.

[20] M. Hassanalieragh *et al.*, “Health Monitoring and Management Using Internet-of-Things (IoT) Sensing with Cloud-Based Processing: Opportunities and Challenges,” *Proceedings - 2015 IEEE International Conference on Services Computing, SCC 2015*, pp. 285–292, 2015.

[21] L. B. and G. D. M. E.-Y. Chung, “Dynamic power management using adaptive learning tree,” *ICCAD ’99 Proceedings of the 1999 IEEE/ACM international conference on Computer-aided design*, pp. 274–279, 1999.

[22] V. L. Prabha and E. C. Monie, “Hardware Architecture of Reinforcement Learning Scheme for Dynamic Power Management in Embedded Systems,” *EURASIP Journal on Embedded Systems*, vol. 2007, pp. 1–6, 2007.

[23] A. Iranfar, M. Zapater, and D. Atienza, “A machine learning-based approach for power and thermal management of next-generation video coding on MPSoCs,” *Proceedings of the Twelfth IEEE/ACM/IFIP International Conference on*

Hardware/Software Codesign and System Synthesis Companion - CODES '17, pp. 1–2, 2017.

[24] K. Bhatti, C. Belleudy, and M. Auguin, "Power management in real time embedded systems through online and adaptive interplay of DPM and DVFS policies," *Proceedings - IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, EUC 2010*, pp. 184–191, 2010.

[25] STMicroelectronics, "STM32L071KB - Ultra-low-power ARM Cortex-M0+ MCU with 128-Kbytes Flash, 32 MHz CPU - STMicroelectronics." [Online]. Available: http://www.st.com/en/microcontrollers/stm32l071kb.html. [Accessed: 08-Apr-2018].

[26] M. C. U. Arm, "STM32L071x8 STM32L071xB," no. March. 2016.

[27] Kionix, "Tri-Axis Magnetometer/Tri-Axis Accelerometer - KMX62." [Online]. Available: https://www.kionix.com/product/KMX62-1031.

[28] ZLG, "Nano Type ZigBee Wireless Module." [Online]. Available: http://www.zlg.com/wireless/wireless/product/id/136.html. [Accessed: 09-Apr-2018].

[29] Quectel, "GNSS L76 - Quectel Wireless Solutions." .

[30] SiTime, "SiT1552: 1.2mm2, NanoPower, ±10 ppm 32 kHz TCXO | SiTime." .

[31] Macronix, "Macronix MX25R6435F Datasheet." pp. 1–86, 2017.

[32] Texas Instrument, "BQ27621-G1 System-Side Fuel Gauge with Dynamic Voltage Correlation, Battery Gas Gauge | TI.com." [Online]. Available: http://www.ti.com/product/BQ27621-G1/description. [Accessed: 09-Apr-2018].

[33] Analog-Devices, "Adp5091-2." .

[34] SanDisk Corporation, “SanDisk SD Card Product Manual,” vol. 95035, no. 80, p. 123, 2004.

[35] Fluke, “True-RMS AC Voltage and Current Logging Multimeter | Fluke 289.” [Online]. Available: http://en-us.fluke.com/products/digital-multimeters/fluke-289-digital-multimeter.html. [Accessed: 12-Apr-2018].

[36] R. S. Sutton and A. G. Barto, *Reinforcement Learning : An Introduction*. 2015.

[37] D. L. Poole and A. K. Mackworth, *Artificial intelligence: Foundations of computational agents*, vol. 9780521519. 2010.

[38] J. Skach, I. Puncochar, and F. L. Lewis, “Optimal active fault diagnosis by temporal-difference learning,” in *2016 IEEE 55th Conference on Decision and Control (CDC)*, 2016, no. Cdc, pp. 2146–2151.

[39] M. A. M. Vieira, C. N. Coelho, D. C. da Silva, and J. M. da Mata, “Survey on wireless sensor network devices,” *EFTA 2003. 2003 IEEE Conference on Emerging Technologies and Factory Automation. Proceedings (Cat. No.03TH8696)*, vol. 1, pp. 537–544, 2003.

[40] S. Chalasani and J. M. Conrad, “A survey of energy harvesting sources for embedded systems,” *Conference Proceedings - IEEE SOUTHEASTCON*, pp. 442–447, 2008.

[41] M. K. Stojčev, M. R. Kosanovic, and L. R. Golubovic, “Power management and energy harvesting techniques for wireless sensor nodes,” *Proceedings of the 9th International Conference on Telecommunication in Modern Satellite, Cable, and Broadcasting Services 2009 (TELSIKS ’09)*, pp. 65–72, 2009.

[42] V. Raghunathan, S. Generiwal, M. Srivastava, S. Ganeriwal, and M. Srivastava,

"Emerging techniques for long lived wireless sensor networks," *IEEE Communications Magazine*, vol. 44, no. 4, pp. 108–114, Apr. 2006.

[43] Y. Ramadass, "Powering the internet of things," in *2014 IEEE Hot Chips 26 Symposium (HCS)*, 2014, pp. 1–50.

[44] H. G. Lee and N. Chang, "Powering the IoT: Storage-less and converter-less energy harvesting," *20th Asia and South Pacific Design Automation Conference, ASP-DAC 2015*, pp. 124–129, 2015.

[45] S. P. Beeby, M. J. Tudor, and N. M. White, "Energy harvesting vibration sources for microsystems applications," *Measurement Science and Technology*, vol. 17, no. 12, pp. R175–R195, 2006.

[46] F. Giuppi, K. Niotaki, A. Collado, and A. Georgiadis, "Challenges in energy harvesting techniques for autonomous self-powered wireless sensors," *2013 43rd European Microwave Conference, EuMC 2013 - Held as Part of the 16th European Microwave Week, EuMW 2013*, pp. 854–857, 2013.

Appendix A: Ported Algorithms

Detection and counting algorithm.

```
/* =============== Detection Algorithm ================ */
if (KMX62_DRDY_INT_Flag == 1)  // Whenever data ready interrupt is triggered
{
	KMX62_DRDY_INT_Flag = 0;	// Clear the flag
	KMX62_GetData();                                        // Get the data reading

	// Wait for the OTH interrupt to come from the magnetometer
	if (KMX62_OTH_INT_Flag == 1)
	{
		// Check if time of arrival should be logged
		if (TA_Flag == 1)
		{
			TA_Flag = 0;
			// Display time of arrival
			RTC_GetDateTime(vehTime, vehDate);
			sprintf((char*)uart_tx_buffer,"%s\n%s %s %s\n",
				"Vehicle Detected!", "Arrival Time:", vehDate, vehTime);
			HAL_UART_Transmit(&huart2, (uint8_t *)uart_tx_buffer,
				strlen((const char*)uart_tx_buffer), 5000);
			DP_counter = 0;
			HAL_TIM_Base_Start_IT(&htim7);
		}
		if (MEMS_Magnetometer < MAG_HTH_THRESHOLD)
		{
			if (!(htim6.Instance->CR1 & TIM_CR1_CEN_Msk))
			{

			// In other words, check if TIM6 is disabled
				// Take a reference point of time of departure
				RTC_GetDateTime(vehTime, vehDate);
				// Start the debounce timer
				HAL_TIM_Base_Start_IT(&htim6);
			}

			// If HTH debounce flag is triggered, then vehicle departure is
confirmed
			if (HTH_Debounce_Flag == 1)
			{
				// Stop the HTH debounce timer
				HAL_TIM_Base_Stop_IT(&htim6);
				// Clear the flag
				HTH_Debounce_Flag = 0;
				// Stop the PDT debounce timer
				HAL_TIM_Base_Stop_IT(&htim7);
				// Clear the DP counter
				DP_counter = 0;
				// Save the reference departure time as the actual one
				sprintf((char*)uart_tx_buffer,"%s %s %s\n",
					"Departure Time:", vehDate, vehTime);
				HAL_UART_Transmit(&huart2, (uint8_t *)uart_tx_buffer,
					strlen((const char*)uart_tx_buffer), 5000);
				// Increase the vehicle counter
				vehiclesCounter++;
				sprintf((char*)uart_tx_buffer,"%s %d\n",
					"Vehicles Counter:", vehiclesCounter);
				HAL_UART_Transmit(&huart2, (uint8_t *)uart_tx_buffer,
					strlen((const char*)uart_tx_buffer), 5000);
				// Clear the OTH flag indicating the end of the detection cycle
				KMX62_OTH_INT_Flag = 0;
```

```
                                // Read the Interrupt Latch Release Register to clear the
interrupt
                                KMX62_I2C_BufferRead(KMX62_DeviceAddress, &KMX62_char,
                                        KMX62_INL, 1);
                                // Calibrate the magnitude references
                                Ref_Mag_Calb();
                        }
                }
                else
                {
                        // Stop the HTH debounce timer
                        HAL_TIM_Base_Stop_IT(&htim6);
                        // Clear the flag
                        HTH_Debounce_Flag = 0;
                }

                // Check if the detection period exceeds the specified delay (Whether a vehicle
is parked on top of the sensor
                if (DP_counter >= DP_Delay)
                {
                        UART_send("Stuck vehicle... Re-calibrating... ");
                        // Stop the DP timer
                        HAL_TIM_Base_Stop_IT(&htim7);
                        // Clear the DP counter
                        DP_counter = 0;
                        /// TODO: Execute function to check if vehicle is on top of the sensor
                        // Execute magnetometer threshold recalibration function
                        Ref_Mag_Calb();
                }
        }
}
```

Adaptive compensation of baseline drift.

```
void Ref_Mag_Calb ()
{
	// Raise the magnitude reference flag to prevent nested calls to Ref_Mag_Calb
	Ref_Mag_Calb_Flag = 1;

	// Stop all timers in case they are on
	HAL_TIM_Base_Stop_IT(&htim6);
	HAL_TIM_Base_Stop_IT(&htim7);

	// Clear their flags
	HTH_Debounce_Flag = 0;
	DP_counter = 0;

	// Loop for the number of the moving average filter taps
	for (int i = 0; i < MAF_Taps; i++)
	{
		// Wait for the data ready interrupt
		while (!KMX62_DRDY_INT_Flag) {};

		// Clear the flag once we detect that it is raised
		KMX62_DRDY_INT_Flag = 0;

		// Read the magnitude
		KMX62_GetData();

		// Accumulate readings for the three axises separately and the
magnitude
		AVG_Mag_Conv_X_Axis = AVG_Mag_Conv_X_Axis + MEMS_Mag_Xout_Conv;
		AVG_Mag_Conv_Y_Axis = AVG_Mag_Conv_Y_Axis + MEMS_Mag_Yout_Conv;
		AVG_Mag_Conv_Z_Axis = AVG_Mag_Conv_Z_Axis + MEMS_Mag_Zout_Conv;
	}

	// Average the 3 axises readings and assign the result to the reference
variables
	MEMS_Mag_Conv_Xout_ref = AVG_Mag_Conv_X_Axis / MAF_Taps;
	MEMS_Mag_Conv_Yout_ref = AVG_Mag_Conv_Y_Axis / MAF_Taps;
	MEMS_Mag_Conv_Zout_ref = AVG_Mag_Conv_Z_Axis / MAF_Taps;

	// Clear variables
	AVG_Mag_Conv_X_Axis = 0;
	AVG_Mag_Conv_Y_Axis = 0;
	AVG_Mag_Conv_Z_Axis = 0;
	// Clear the OTH interrupt flag to eliminate the missdetection that happens
after threshold recalibration
	KMX62_OTH_INT_Flag = 0;
	// Clear the reference magnitude calibration flag
	Ref_Mag_Calb_Flag = 0;
	UART_send("MGM re-calibration done!\n");
}
```

Adaptive compensation of RTC frequency drift.

```
/* =============== RTC Drift Compensation =============== */
if (Calibrate_RTC_Drift)
{
	// Disable interrupts on LINE 4 and 5, which corresponds to
interrupts on GPIO 4 and 5, i.e. MGM data ready and motion interrupt
	EXTI->IMR &= ~(EXTI_IMR_IM5 | EXTI_IMR_IM4);
	// Check whether it is RTC 1 Hz signal or PPS that should be
measured now
	if (RTC_PPS_Meas_Round == 0)		// RTC Round = 0, PPS
Round = 1
	{
		if (meas_count < RTC_MEAS_PERIOD)   // ...and we still
have measurments to take...
		{
			// Set the frequency measuring flag
			measure_freq = 1;
			// Wait until the measurment is done
			while(measure_freq);
			// Accumulate the counter value
			counter_RTC += (htim22.Instance->CNT << 16) |
(htim21.Instance->CNT);
			// Increase the number of measurments taken
			meas_count++;
		}
		else // If we already collected RTC_MEAS_PERIOD
measurments...
		{
			// ...calculate the average
			counter_RTC = counter_RTC / RTC_MEAS_PERIOD;
			/******** Debug Purpose **************/
			sprintf((char*)uart_tx_buffer,"RTC Counter: %d\n",
counter_RTC);
			HAL_UART_Transmit(&huart2, (uint8_t
*)uart_tx_buffer, strlen((const char*)uart_tx_buffer), 5000);
			/************************************/
			// Set the round now for PPS
			RTC_PPS_Meas_Round = 1;
			// Clear the meas_count
			meas_count = 0;
		}
	}

	else		// Then it is the PPS turn
	{
		if (meas_count < RTC_MEAS_PERIOD)   // ...and we still
have measurments to take...
		{
			// Set the frequency measuring flag
			measure_freq = 1;
			// Wait until the measurment is done
			while(measure_freq);
```

```
                    // Accumulate the counter value
                    counter_PPS += (htim22.Instance->CNT << 16) |
(htim21.Instance->CNT);
                    // Increase the number of measurments taken
                    meas_count++;
              }

              else // If we already collected RTC_MEAS_PERIOD

measurments...

              {
                    // ...calculate the average
                    counter_PPS = counter_PPS / RTC_MEAS_PERIOD;

                    /******** Debug Purpose **************/
                    sprintf((char*)uart_tx_buffer,"PPS Counter: %d\n",
counter_PPS);
                    HAL_UART_Transmit(&huart2, (uint8_t
*)uart_tx_buffer, strlen((const char*)uart_tx_buffer), 5000);
                    /***********************************/

                    // Calculate the corresponding time difference to
be written on RTC Shift registers
                    RTC_Drift = (((float)counter_PPS -
(float)counter_RTC)/8000000.0f);
                    // Update the RTC shift register
                    // Read 22.4.10 RTC synchronization in Reference
Manual to understand why is the following calculations
                    if (RTC_Drift < 0)
                          HAL_RTCEx_SetSynchroShift(&hrtc,
RTC_SHIFTADD1S_RESET, (uint32_t)(fabsf(RTC_Drift)*256.0f));
                    else
                          HAL_RTCEx_SetSynchroShift(&hrtc,
RTC_SHIFTADD1S_SET, (uint32_t)((1-RTC_Drift)*256.0f));

                    sprintf((char*)uart_tx_buffer,"RTC Drift in
seconds: %f\n", RTC_Drift);
                    HAL_UART_Transmit(&huart2, (uint8_t
*)uart_tx_buffer, strlen((const char*)uart_tx_buffer), 5000);
                    UART_send("RTC Calibrated!\n");
                    // Clear the meas_count
                    meas_count = 0;
                    // Clear freq counters
                    counter_PPS = 0;
                    counter_RTC = 0;
                    // Set the measurment round for RTC
                    RTC_PPS_Meas_Round = 0;
                    // Clear the calibration flag
                    Calibrate_RTC_Drift = 0;
                    // Enable interrupts for MGM
                    EXTI->IMR |= (EXTI_IMR_IM5 | EXTI_IMR_IM4);
              }
        }
}
```

Optimized ARM DSP arithmetic.

```
// These functions cause a delay of 0.75 mSec
MEMS_Magnetometer = sqrt(pow(X,2)+pow(Y,2)+pow(Z,2));

// The C math functions are replaced with fast, optimized ARM math
functions
// These take 0.13 mSec
arm_power_f32(&X, 1, &X2);
arm_power_f32(&Y, 1, &Y2);
arm_power_f32(&Z, 1, &Z2);
arm_add_f32(&X2, &Y2, &X2Y2_sum, 1);
arm_add_f32(&X2Y2_sum, &Z2,&X2Y2Z2_sum, 1);
arm_sqrt_f32(X2Y2Z2_sum, &MEMS_Magnetometer);
```

Terje Mathisen numeric to ASCII conversion.

```
typedef uint32_t fix4_28;
void itoa(char** int_buf, int16_t value)
{
      uint32_t val;
      if (value < 0) val = -value;
      else val = value;

      *int_buf = calloc(6, sizeof(char));

      fix4_28 const f1_10000 = (1 << 28) / 10000;
      fix4_28 tmplo, tmphi;

      uint32_t lo = val % 100000;
      uint32_t hi = val / 100000;

      tmplo = lo * (f1_10000 + 1) - (lo / 4);
      tmphi = hi * (f1_10000 + 1) - (hi / 4);

      for(size_t i = 0; i < 5; i++)
      {
                  (*int_buf)[i + 0] = '0' + (char)(tmphi >> 28);
                  (*int_buf)[i + 5] = '0' + (char)(tmplo >> 28);
                  tmphi = (tmphi & 0x0fffffff) * 10;
                  tmplo = (tmplo & 0x0fffffff) * 10;
      }
      char* p = *int_buf;
      if (*((uint64_t*) p) == 0x3030303030303030)
                  p += 8;
      if (*((uint32_t*) p) == 0x30303030)
                  p += 4;
      if (*((uint16_t*) p) == 0x3030)
                  p += 2;
      if (*((uint8_t*) p) == 0x30)
                  p += 1;
      free(*int_buf);
```

```
        if (value < 0)
        {
                p -= 1;
                p[0] = '-';
        }
        *int_buf = p;
}
```

Appendix B: Optimized Algorithms

Data buffering technique: RAM-to-Flash.

```
uint8_t Flash_WriteData(uint8_t* data)
{
	// Copy data into the buffer
	memcpy(flashBuf + flashBufSize, data, strlen((char*)data));
	// Change buffer size accordingly
	flashBufSize += strlen((char*)data);
	// If buffer size exceeds 256, move a page into the flash
memory
	if (flashBufSize >= 256)
	{
		// Write only one page (256 bytes from the buffer)
		if (Flash_WritePage(flashBuf, flashAddr, 256))
		{
			// Increase the flash address
			flashAddr += 256;
			// Shift remaining data in the buffer to the
beginning of the buffer
			memmove(flashBuf, flashBuf+256, flashBufSize-256);
			// Adjust the buffer size
			flashBufSize -= 256;
			// Increase number of pages written to flash
			flashPagesNum++;
		}
		else // If flash page write operation failed...
		{
			// Change back the buffer size
			flashBufSize -= strlen((char*)data);
			return 0;
		}
	}
	return 1;
}
```

Data buffering technique: Flash-to-microSD card.

```
uint8_t writeData(char* data)
{
	if (!log_data) return 0;
	// Erase the sector first. If the address points to the beginning of
a sector, erase it
	// Every 16 pages form a sector
	if (flashPagesNum % 16 == 0)
		if (!Flash_SE(flashAddr))
			return 0;
	// Write data to the flash memory
	if (!Flash_WriteData((uint8_t*)data)) return 0;
	// If Number of written pages on flash exceeds a threshold move data
to the SD card
	if (flashPagesNum >= FLASH_PAGES_THRESHOLD) {
```

```
            // Turn on SD card switch
            HAL_GPIO_WritePin(LS_uSD_GPIO_Port, LS_uSD_Pin, GPIO_PIN_SET);
            HAL_Delay(100);
            // Mount SD card
            if(FATFS_LinkDriver(&USER_Driver, SDPath) == 0) {
                  if(f_mount(&SDFatFs, (TCHAR const*)SDPath, 0) != FR_OK)
                        return 0; }
            // Open data file
            if (!newDataFile)
                  openDataFile(dataFile, 0);
            else
                  openDataFile(dataFile, 1);
            // Read data from flash and write to SD card recursively
            uint8_t tempFlashBuf[256] = {0};
            while(flashPagesNum != 0) {
                  // Read one page
                  Flash_ReadData(tempFlashBuf, flashAddr-
(flashPagesNum*256), 256);
                  // Decrease number of occupied pages in flash
                  flashPagesNum--;
                  // Write one page data to SD card
                  if (writeToFile((char*)tempFlashBuf) != FR_OK)
                        return 0;
            }
            flashBuf[flashBufSize] = '\0';
            if (writeToFile((char*)flashBuf) != FR_OK)
                  return 0;
            // Reset the flash buffer
            memset(flashBuf, 0, 256);
            // Reset buufer size
            flashBufSize = 0;
            // Reset flash writing address to the beginning
            flashAddr = 0x00000000;
            // Close data file
            if (closeDataFile(0) != FR_OK) return 0;
            // Unmount the SD card
            f_mount(0, (TCHAR const*)SDPath, 0);
            FATFS_UnLinkDriver(SDPath);
            // Turn off SD card switch
            HAL_GPIO_WritePin(LS_uSD_GPIO_Port, LS_uSD_Pin,
GPIO_PIN_RESET);
      }
      return 1;
}
```

Communication scheme.

```
if (Process_CMD_Flag) {
#ifdef _STATUS_BEACON
    // Resetting the counter, giving the user another 2 minutes to send
commands over ZigBee
    zigbee_on_cntr = 0;
#endif // _STATUS_BEACON
    Process_CMD_Flag = 0;
    zigbee_send("CMD Received: ");
    zigbee_send(cmd_buffer);
    zigbee_send("\n");
    handle_command(cmd_buffer);
}
```

```
/* ================ 1-Minute Reference, Reinforcement Learning Algorithm,
and Status Beacon Messages ================ */
if (Alarm_Min_Flag && !OTH_INT_Flag)
{
#ifdef _STATUS_BEACON
    // Status beacon
    uint8_t minute = floor(((float)(timestamp)/3600.0f -
(float)floor(timestamp/3600))*60.0f);
    if (minute == 0 || minute == 15 || minute == 30 || minute == 45)
    {
        // Enable ZigBee
        HAL_GPIO_WritePin(LS_ZigBee_GPIO_Port, LS_ZigBee_Pin,
GPIO_PIN_SET);
        // Wait a few milliseconds
        HAL_Delay(100);
        // Enable terminal messages
        sendTerminal = 1;
        // Send battery status and vehicles counter
        sprintf(uart_tx_buffer, "BAT: VOLT = %d mV, CAP = %d mAh, SOC = %d
%%\r\nVehicles Counter: %d\r\n", batVolt(), batRemainingCapacity(),
batSOC(), vehiclesCounter);
        zigbee_send(uart_tx_buffer);
        // Wait another minute for command requests from AP/user
        zigbee_wait_enable = 1;
    }
    // If the AP/user sends a command, 'zigbee_on_cntr' will be reset
giving 2 minutes before ZigBee is turned off
    // by the following block
    if (zigbee_wait_enable)
    {
        if (++zigbee_on_cntr == ZIGBEE_ON_PERIOD)
        {
            // Disable the wait flag
            zigbee_wait_enable = 0;
            // Clear the counter
            zigbee_on_cntr = 0;
            // Turn off ZigBee
            HAL_GPIO_WritePin(LS_ZigBee_GPIO_Port, LS_ZigBee_Pin,
GPIO_PIN_RESET);
            // Disable terminal messages
            sendTerminal = 0;
        }
    }
#endif  // _STATUS_BEACON
}
```