Assessing activity of IoT smart stuff using inertial sensors

Juan Tarrat

Follow this and additional works at: https://louis.uah.edu/uah-theses

Recommended Citation

This Thesis is brought to you for free and open access by the UAH Electronic Theses and Dissertations at LOUIS. It has been accepted for inclusion in Theses by an authorized administrator of LOUIS.
ASSESSING ACTIVITY OF IOT SMART STUFF USING INERTIAL SENSORS

Juan Tarrat

A THESIS

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Engineering in Computer Engineering to The Graduate School of The University of Alabama in Huntsville

December 2023

Approved by:
Dr. Emil Jovanov, Research Advisor/Committee Chair
Dr. David Coe, Committee Member
Dr. Dinh Nguyen, Committee Member
Dr. Aleksandar Milenkovic, Department Chair
Dr. Shankar Mahalingam, College Dean
Dr. Jon Hakkila, Graduate Dean
Abstract

ASSESSING ACTIVITY OF IOT SMART STUFF USING INERTIAL SENSORS

Juan Tarrat

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Engineering

Computer Engineering
The University of Alabama in Huntsville
December 2023

Inertial sensors, including accelerometers and gyroscopes, are commonly used for capturing and quantifying motion and orientation data. IoT devices use inertial sensors to detect user activity. In this thesis we present the use of inertial sensors for determining user interactions with smart objects of everyday use, which we refer to as Smart Stuff. We present a novel use of inertial sensor to assess the use and state of the smart water bottle configured as IoT appliance. Staying hydrated is crucial for health and wellbeing; precise monitoring of user hydration is very important for the elderly and patients with chronic conditions. We developed a method of monitoring liquid intake using a smart cup equipped with inertial sensors. To do this, we developed a theoretical model, implemented algorithms on embedded controller in the smart cup, and experimentally validated the accuracy of proposed algorithms. The smart cup is integrated in IoT environment and sends data wirelessly to a server that can be integrated into personal medical records to allow users, physicians, nurses, and caregivers to track a user’s hydration remotely.
Acknowledgements

I would like to thank all the members of the thesis committee, in particular to Dr. Emil Jovanov. Your guidance, expertise, and passion are the main reason I decided to pursue this project, and without you this would have not been possible.

I would like to mention my gratitude to this academic institution, with special focus on the College of Engineering, for hosting me for such a long time and giving me the opportunity to learn and develop as a professional and as a person.

I would also like to thank my family for their unconditional support throughout my college career. Their effort and sacrifice have been a determining factor in the success of my career. My friends, international and from home, for always being there for me in my ups and downs.

This is to all of you,
Thank you.
Table of Contents

Abstract ................................................................. ii

Acknowledgements ......................................................... iv

Table of Contents ........................................................... vii

List of Figures .............................................................. viii

List of Tables ............................................................... x

Chapter 1. Introduction ..................................................... 1

Chapter 2. IoT Survey ....................................................... 4
  2.1 IoT Trends .......................................................... 4
  2.2 The IoT Framework ................................................ 5
  2.3 IoT Development Boards ......................................... 6
  2.4 IoT Wireless Protocols .......................................... 10
    2.4.1 Data Link Protocols ...................................... 11
    2.4.2 Network Layer Protocols ................................. 12
    2.4.3 Session Layer Protocols ................................. 12
4.3.3 Validation of Results ................................. 42

Chapter 5. Smart Cup as an IoT Appliance ................. 45

5.1 Existing Literature .......................................... 45
5.2 System Architecture ......................................... 48
5.3 System Organization .......................................... 49
  5.3.1 euHy Controller .......................................... 49
  5.3.2 ESP32 Microcontroller ................................. 52
  5.3.3 Inertial Sensors .......................................... 52
  5.3.4 The Board ................................................. 53
  5.3.5 Server .................................................. 54
  5.3.6 Software Model ......................................... 57
5.4 Assessment of Tremor of Parkinson Patients ........... 62

Chapter 6. Conclusion ........................................... 64

References .......................................................... 66

Appendix A. Software Diagrams ............................... 70
List of Figures

2.1 IoT framework. .................................................. 6
2.2 ESP32C6 module. ............................................. 8
2.3 ESP32-C6 functional block diagram. ....................... 9
2.4 DuckDNS web portal. .......................................... 17
2.5 Home assistant hardware platforms. ......................... 18
2.6 Tinker board by ASUS. ........................................ 19
2.7 Odroid board by HardKernel. ............................... 19
2.8 Raspberry Pi 5. .................................................. 20
2.9 MQTT architecture. ............................................ 21
3.1 Calibration experiment .......................................... 25
3.2 MMA8653. ...................................................... 26
3.3 Adafruit 9-DOF sensor. ....................................... 28
4.1 Cylindrical segment ............................................ 31
4.2 Cylindrical wedge. ............................................. 33
4.3 Water in the cup. .............................................. 34
4.4 Model ............................................................. 36
4.5 Acceleration and tilt angle during the drinking events. 38
4.6 Sensor data vs Model. ......................................... 39
4.7 Piece-wise fitting of experimental data for off-board sensor. 40
4.8 Fitted line for the off-board sensor. ........................ 41
4.9 Piece-wise fitting of experimental data for on-board sensor. 42
4.10 Fitted line on-board sensor. ................................. 43
List of Tables

2.1 ESP32-C6 power consumption in low-power modes. . . . . . . . . . 10
2.2 Summary of options for MQTT broker. . . . . . . . . . . . . . . . 23
4.1 Error comparison vs theoretical model. . . . . . . . . . . . . . . . 38
4.2 Error comparison vs correspondent fitted line. . . . . . . . . . . . 43
5.1 Comparison between the two candidate controller boards. . . . . 53
5.2 Embedded controller states. . . . . . . . . . . . . . . . . . . . . . 57
5.3 Server automations. . . . . . . . . . . . . . . . . . . . . . . . . . 61
Chapter 1. Introduction

Internet of Things (IoT) systems have revolutionized industries and our daily lives by enabling the seamless communication and integration of devices and sensors in our environment and objects of everyday use [1]. This connectivity facilitates gathering and analysis of data, and control of devices and environments. Integration of wearable monitoring with IoT sensors opens new opportunities for synergy of information from physiological and environmental sensors [2].

Inertial sensors are very important in IoT systems, as they facilitate motion sensing, context awareness, energy efficiency, and data fusion. They improve robustness of measurements, context of other measurements, and assessment of activity and our interactions with objects of everyday use.

Hydration is one of the key aspects of human health and well-being, and maintaining it at optimal level is of great importance. Depending on the age, a human body can be made up of up to 78% of water in babies and 60% in adults [3]. As people become older, the amount of water they consume daily decreases, in fact, research shows that older adults are more likely to suffer from dehydration, with the elderly population being up to 30% more likely to develop this pathology due to decreased mobility, impaired thirst mechanisms and other organ failures.
This is particularly important for subjects in assisted living facilities, such as nursing homes.

Studies show that correct hydration in general population plays an important role in skin health, neurological function, renal function and body weight and composition [5]. In addition to this, water intake may also play an important role in the early detection of pathologies like diabetes. When some patient is suffering from diabetes, there is an increase of sugar in the blood, which then leads to an increased renal activity and thus making people thirstier [6].

Although the most common problem is dehydration, overhydration is also a critical problem, particularly for cardiac and renal patients. Therefore, it is only logical to think that having a way of precisely monitoring the volume of fluid taken by a person is essential, at least for some categories of users. With smart health this has become a reality. With all the advanced technology in Internet of Things (IoT), and with microcontrollers and sensors becoming increasingly more powerful and smaller, the opportunity for implementing seamless monitoring systems to track daily activities such as hydration is closer than ever [7]. Human activity recognition (HAR) is the process of classification of activities based on discrete measurements, such as position, acceleration, or angular velocity, from personal digital devices. The number of articles on HAR has been exponentially increasing since 2008 [8]. Starting from iPhone 5S, motion processors became a standard feature of mobile phones, allowing monitoring of activity, number of steps, number of flights climbed, or assessment on the quality of our steps when running, as a standard support of smartphones, smartwatches, and wearable
devices. New smartwatches, like Samsung Watch 4, integrate bio impedance, and most smartwatches feature monitoring of heart rate.

The main objective of this research is to evaluate a method for assessment of the use of smart objects and systems for hydration monitoring using embedded inertial sensors. We use a smart cup and water bottle developed at UAH with integrated inertial sensors to monitor the volume of liquids consumed throughout the day. To determine the amount of water in the bottle after each use, we developed a mathematical model to assess the volume of water left in a cup as a function of the final angle of inclination. Calculated volume is then processed and wirelessly transferred to a home server where the user can track his or her liquid consumption during the day.

The organization of this document is as follows. Chapter 2 presents a survey of the state of the art of IoT, as well as the architecture and framework of the IoT systems, a review of the IoT device platforms and gateways and wireless protocols, and particularly MQTT protocol and MQTT broker alternatives. Chapter 3 presents inertial measurement units. Specifically, we present the two inertial sensors used in the system implementation, and rationale for the choice of communication protocols. Chapter 4 presents derivation of the mathematical model developed to estimate the volume of liquids in a cup after tilting. In addition, a discussion of results and their validity is also present in this section. Chapter 5 presents system’s architecture and organization, the software model, and a survey of existing literature on smart cup implementations. Chapter 6 concludes the thesis and outlines how this work could be expanded and improved.
Chapter 2. IoT Survey

2.1 IoT Trends

We live in an era called the third wave of the internet revolution. IoT is increasingly present in our daily lives. Ubiquitous computing is the future and the present of technology as we know it. Thanks to enhanced connectivity and the evolution of processors, there is a universe of opportunity for development of IoT devices that can make our lives better. According to Statista, in 2022 the number of connected devices was 9.76 billion, with a forecast of 29.42 billion in 2030 [9].

One of the areas of increasing development and innovation is smart health [10]. Cheaper and smaller sensors are enabling the capabilities of seamless detection of activities. Models like the one described in [10] bring together all the available technologies in a solid infrastructure that uses sensors that monitor different health parameters (body temperature, ECG, blood pressure, etc.). Data from the sensors goes through the gateway level, like a phone, or a home server and from there it becomes available for storage in the cloud or edge devices that act on the data [11].

Among the activities that can be monitored on a daily basis, one that is undoubtedly of crucial importance for every human being is hydration. Having a
way of seamlessly and precisely estimate the volume of liquid intake from a user could be of great help not only for people that need constant monitoring (like elderly in nursing homes and hospitals), but for every other person that wants to live a healthier life. Real examples have already been put into practice like in [12], where they developed a system in a nursing home in France. One of the main challenges of implementing a hydration monitoring system is that it is hard to pair a smart cup with a user, especially when cups must be regularly washed. In [12] they designed the system that allows users to use a tag to assign a cup in the system, making it very flexible.

2.2 The IoT Framework

IoT aims to connect millions of devices to make life of people easier by having a network of interconnected devices that together form an environment where not only users can monitor daily aspects of their life, but also where automation is a key advantage.

The IoT framework is comprised of four layers, including the device layer, consisting of all the edge components like sensors that monitor parameters, such as room temperature, or inertial sensors that monitor the use of a certain object. Next is the aggregation layer, where the data from the device layer is put together (normally using some wireless protocol like BLE or WiFi, and messaging protocol such as MQTT) before sending it to the cloud for processing; then comes the processing layer, where the data gathered from the sensors and all the calculations
are displayed. Lastly comes the application layer, which is the result of all the previous processing in the form of a front-end display or an event taking place\cite{13}.

An example of this would be a home automation system comprised of several light sensors that will turn on the front porch lights when the sun goes down. Here, the device layer is comprised of the sensors, that will put together light data which will then be processed to determine whether or not to turn on the light. Turning the lights on or off is the result of the processing layer, which determines whether or not the event should take place.

### 2.3 IoT Development Boards

The first layer in the IoT framework involves the IoT device itself. Most of the time, these devices are built around a low power embedded micro controllers.
More complex applications may have their own architecture, but the vast majority of IoT projects can be developed using an Arduino-enabled development board. Manufacturers of these micro controllers include Arduino, Teensy or Espressif \[14\][15][16]. There are two approaches to the development an IoT application using an embedded micro controller. The first one is by using an external wireless module that connects to the board, thus expanding it’s original functionality. These modules expand the original functionality of the processor so there is no need to use an extra controller with wireless communications capabilities. This is the case for most boards that don’t support wireless connectivity. The most frequently used wireless interfaces are WiFi, and BLE for extremely low power applications.

The second option is to use an embedded micro controller development board that does have built-in wireless functionality. The most popular choices for these are the ESP32 and ESP8266. ESP32 was created by Espressif Systems \[16\], and is a low cost and low power consumption system on chip with WiFi and Bluetooth capabilities. The ESP8266 is the precursor of the ESP32 family and does not offer BLE capabilities. On the other hand, ESP32 boards offer the users all the advantages of a lightweight Arduino with the additional advantage of WiFi and BLE. They are very easy to program thanks to the Arduino compatibility, and in addition to this, if users are not proficient in C++ development used in Arduino environment, ESP32 supports Micropython, an implementation of Python made to run on microcontrollers. The latest product they have released is the ESP32-C6 system on chip, shown in Fig. 2.2 and Fig 2.3.
Among the most interesting capabilities of the ESP32-C6 module, we find the following [17]:

- **WiFi**: 4 virtual WiFi interfaces (STA, AP, Sniffer, and reserved), 802.11b/g/n, up to 150 Mbps, antenna diversity, simultaneous support for Infrastructure Station, SoftAP, and Promiscuous modes.

- **Bluetooth**: Bluetooth LE 5.3 certified, high power mode (20 dBm), speed up to 2 Mbps, multiple advertisement sets, coexistence mechanism with WiFi to share the same antenna.

- **Processor**: HP RISC-V processor with a speed up to 160 MHz, four stage pipeline; LP RISC-V with a speed up to 20 MHz, two stage pipeline.

- **Memory**: 320 KB ROM, 512 KB HP SRAM, 16 KB LP SRAM.
However, what makes this family of processors so suitable for IoT applications, in addition to the wireless support, is the support for low power modes of operation. As an example, the ESP32C6 supports four power modes [17]:

- **Active mode**: Chip and radio are powered. The chip can transmit, receive, and listen. Full capabilities of the chip are available.

- **Modem-sleep mode**: The CPU is operational. WiFi base and radio are disabled, but WiFi connection can remain active.
• **Light-sleep mode:** The CPU is paused, any wake-up events will turn on the chip again. WiFi connection can remain active.

• **Deep-sleep mode:** Only the LP CPU is operational. WiFi connection data is stored in LP memory.

Modem-sleep is still considered an active mode, with peaks of up to 38 mA of current consumption. The other three modes provide a much lower power consumption.

**Table 2.1:** ESP32-C6 power consumption in low-power modes.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
<th>Typ (µA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Light-sleep</td>
<td>CPU and wireless communication modules are powered down, peripheral clocks are disabled, and all GPIOs are high-impedance</td>
<td>180</td>
</tr>
<tr>
<td></td>
<td>CPU, wireless communication modules and peripherals are powered down, and all GPIOs are high-impedance</td>
<td>35</td>
</tr>
<tr>
<td>Deep-sleep</td>
<td>RTC timer and LP memory are powered on</td>
<td>7</td>
</tr>
<tr>
<td>Power off</td>
<td>CHIP PU is set to low level, the chip is powered off</td>
<td>1</td>
</tr>
</tbody>
</table>

2.4 IoT Wireless Protocols

Following the IoT framework, the devices on the edge must connect to the gateway using some sort of wireless protocol. There is a wide variety of reliable wireless protocols to choose from and the work in [18] identifies multiple layers, in what they call the *interconnection layer*, in the IoT ecosystem. These layers are the data link layer, the network layer, and the transport/session layer.
2.4.1 Data Link Protocols

The data link layer connects IoT sensors to the Internet. Sometimes systems must put together data from different sensors before sending it to the aggregate layer. Commonly used protocols include:

- IEEE 802.15.4e
- IEEE 802.11ah
- WirelessHART
- Z-Wave
- Bluetooth Low Energy (BLE)
- ZigBee Smart Energy
- DASH7
- HomePlug
- G.9959
- LTE-A
- LoRaWAN
- Weightless
- EnOcean
- DECT/ULE
2.4.2 Network Layer Protocols

This contains the set of specialized protocols for communications between IoT devices. Authors identify the following set of protocols in this layer:

- RPL
- CORPL
- CARP and E-CARP
- 6LoWPAN
- 6TiSCH
- 6Lo
- IPv6 over G.9959
- IPv6 over Bluetooth Low Energy

2.4.3 Session Layer Protocols

This section contains the set of protocols developed for message passing; most of these protocols make use of the transport layer, thus involving TCP/UDP protocols for wireless communication:

- MQTT
- SMQTT
• AMQP
• CoAP
• XMPP
• DDS

2.5 IoT Servers

As previously mentioned, two of the key layers of the IoT framework are the aggregation and gateway layers, where data from all the devices is put together and processed (locally or in the cloud). The choice of the correct software/platform for this layer is crucial. Systems must choose a gateway that is compatible with all the sensors and supports functionality they require.

As IoT implementations increase in popularity, the available options for server implementations vary according to the requirements and preferences of the users. Among the candidates for a server implementation that are open-source, we identify three: OpenHAB, Home Assistant, and IoT Stack. What they all have in common is that they can implement the server in small single board computer like Raspberry Pi, which is ideal because of the low price rich set of features.

2.5.1 OpenHAB

OpenHAB is a home automation platform developed in Java that started in 2010. It operates on a Java virtual machine and is available for different platforms like Windows, Mac, or Raspberry Pi. Users are required to be familiar with
programming concepts to edit the configuration. Although all communication is
done locally (meaning that most of the time the server will communicate with
the different smart devices in the house through the local network), OpenHAB
offers a free remote management service that tunnels the traffic through their
own domain and allows the user to access the server via OpenHAB’s service [19].
Users can also access their home server through a dedicated smartphone app.

2.5.2 IoT Stack

One of the most flexible solutions for server implementation is IoT Stack.
It is an open source GitHub project that can be installed in a Raspberry Pi
and offers the users complete freedom to install whatever IoT technology they
want. The way it does this is with containers. Not only can users install both
OpenHAB and Home Assistant separately and run them, but they can also decide
among several MQTT brokers, such as Mosquitto or Zigbee MQTT. It also offers
different database options like InfluxDB or MariaDB for users to store their data
and many other containers for different functionality like remote access.

Users decide what to install; the main issue is that all these containers
have to be managed via SSH connection to the Raspberry Pi, meaning that the
users must have an intermediate to advanced knowledge of computing to success-
fully implement and configure the system. The main benefit of this is that the
containers are ran separately and thus function independent from one another.
This offers a big advantage against a limitation of other more constraint platforms
like Home Assistant. Although Home Assistant is expanding its range of add-ons, some of these still encounter limitations.

### 2.5.3 Home Assistant

Home Assistant is one of the most popular choices for home automation software in the market. It is an open source project that can be deployed in numerous platforms:

- Home Assistant Green (Hardware platform provided by Home Assistant)
- Home Assistant Yellow (Hardware Platform provided by Home Assistant)
- Raspberry Pi
- Odroid
- ASUS Tinker Board
- Generic x86-64
- Windows
- macOS
- Linux

In addition, on each platform users can decide whether to install the Home Assistant operating system, the Home Assistant container, or the Home Assistant core.
According to Schoutsen [20], there are currently 259,078 active installations of Home Assistant, and almost 80% of them run the software through the operating system. Additionally, more than half the installations are running on a Raspberry Pi.

One of the limiting factors of this software is the remote access. As with OpenHAB, users can access their home server from their smartphone using the official app. However, advanced configuration is necessary if users want to send data from sensors outside the local network. Home Assistant requires forwarding of port 8123 for users to see their dashboard, and port 1883 if users want to publish MQTT messages as well as using a DNS service. A DNS service associates an IP of the server to a unique subdomain, giving it a name. This is useful for remote access to a Home Assistant implementation on a local network, since most routers have their public IP set by the internet provider that may change after reboots or reconnections. Home Assistant recommends the use of DuckDNS [21] as a reliable free service with its own add-on. Subdomains provided by this service have the form of <subdomain_name>.duckdns.org. An example of the DuckDNS configuration screen is shown in Fig. 2.4.

A factor that makes Home Assistant so popular among users is the easy configuration. With the most recent versions, users can configure their sensors, automations, connections, and configurations from the web portal’s UI. If one wants a more programming-like configuration, Home Assistant works with YAML configuration files, which are very easy to use and allow users to specify sensors, entities, databases and other configuration options.
2.6 Hardware Platforms

There is a variety of hardware platforms for home servers. The most common implementations are very evenly divided between a separate physical server connected to the home network and a virtual machine, as shown in Figure 2.5. When using a virtual machine, the user is lending his or her personal computer (meaning sharing resources) to the Home Automation platform. This can limit the performance of their personal computer, and that is why the other option involves using an Single Board Computer (SBC) to solely run the IoT automation platform. This section will discuss the three most popular SBC choices: ASUS Tinkerboard, Odroid, and RaspberryPi. The first platform, and one of the least popular among Home Assistant users is ASUS TinkerBoard (Fig. 2.6). It is a powerful ARM-based SBC aimed at more powerful applications. The platform is also more expensive than competitors.

Next in popularity is the ODROID-N2 (Fig 2.7). Odroids are a family of SBCs created by the South Korean company Hardkernel Co. Although more
recent models support more Linux distributions, originally, they were meant to run Android OS. In addition to IoT applications, Odroids are also a popular option for gaming products, and have a higher price.

The most popular SBC option is Raspberry Pi, shown in Figure 2.8. This family of SBCs has been around since 2012. They run their own Linux distribution called Raspberry Pi OS and other operating systems. Models keep evolving and new generations of these single board computers arrive to the market regularly. Recently, Raspberry Pi has announced their most recent board: Raspberry Pi 5. The Raspberry Pi family places itself in the more affordable end of the spectrum of choices for SBCs, with prices between $40 and $80.
2.7 MQTT Protocol

As explained in the previous section, there are different protocols that can be used for wireless data transmission and device communication. The most frequently used protocols in IoT systems include BLE (Bluetooth Low Energy),
ZigBee, HTTP, or MQTT. In this project we used MQTT for communication with the server. Another option was to use BLE as very power efficient method for lightweight communication, but we wanted to support sensor communication with the home server even from outside the network just by connecting to a WiFi network. Therefore, we decided to use a standard WiFi network with MQTT protocol.

Introduced in 2013 by OASIS, MQTT (Message Queuing Telemetry Transport Control) is a client-server, publish/subscribe messaging transport protocol. It runs over TCP/IP layer and is the leading protocol in IoT implementations [22].

Normally, architectures that use MQTT use a common broker. The broker acts as a server where messages are stored and exchanged [23]. Messages come in a variety of formats; one of the most common ones and the one we used for this project is JSON. Messages are published under topics that each client chooses. If we take this system as an example, the broker is hosted on the home server and
the IoT entity (a smart cup) and the IoT platform (Home Assistant) are clients (Fig. 2.9).

Message are published under topics. A topic can be thought of as a key to get access to messages and to publish messages. The broker acts as the middleman between clients, so all the information flows through it. MQTT supports three qualities of service (QoS) delivery [23]:

- QoS 0: Messages are delivered with no assurance of delivery, which means that there could be message loss. An example of an entity that would use this QoS is an environmental sensor that publishes information once per minute. If one of the messages does not arrive, it will not compromise the integrity of the system, since a new one will be published shortly after.

![Figure 2.9: MQTT architecture.](image)
- QoS 1: Messages are assured to arrive, meaning that there is a chance a duplicate might happen.

- QoS 2: Messages are assured to arrive just once. This QoS is useful when duplicate information can be very damaging to any of the clients.

There are several options for MQTT broker implementation, as presented in Table 2.2 [22]. We selected Mosquitto broker. The reasons are that this broker is open source and very popular among end-users for home automation applications. It has its own integration in Home Assistant and is very easy to manage since the integration hides a lot of the complexity from the user, which allows users to change configurations without extensive knowledge of programming using the YAML configuration files.

What makes MQTT so popular among IoT is its ease of implementation and how useful it is for resource-constrained devices, as shown in the work done in [24], where authors explored the use of this protocol using low-cost hardware like Raspberry Pi for the broker and Arduino Uno as the client. The article shows how even in a LAN (local area network) intended for students, this MQTT ecosystem was able to publish messages every 50 ms, which is a very acceptable throughput.
Table 2.2: Summary of options for MQTT broker.

<table>
<thead>
<tr>
<th>Brokers</th>
<th>Open/Close</th>
<th>Written In</th>
<th>Supported OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mosquitto</td>
<td>Open</td>
<td>C</td>
<td>Linux, Mac, Windows</td>
</tr>
<tr>
<td>Bevywise MQTT Route</td>
<td>Close</td>
<td>C, Python</td>
<td>Linux, Unix, Mac, Windows, BSD</td>
</tr>
<tr>
<td>EMQ X</td>
<td>Open</td>
<td>Erlang</td>
<td>Linux, Mac, Windows, BSD</td>
</tr>
<tr>
<td>HiveMQ CE</td>
<td>Open</td>
<td>Java</td>
<td>Linux, Mac, Windows</td>
</tr>
<tr>
<td>HiveMQ</td>
<td>Close</td>
<td>Java</td>
<td>Linux, Mac, Windows</td>
</tr>
<tr>
<td>IBM WIoTP Message Gateway</td>
<td>Close</td>
<td>C</td>
<td>Linux</td>
</tr>
<tr>
<td>JoramMQ</td>
<td>Close</td>
<td>Java</td>
<td>Linux, Unix, Mac, Windows, Raspbian</td>
</tr>
<tr>
<td>flespi</td>
<td>Close</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>PubSub+</td>
<td>Close</td>
<td>C, C++</td>
<td>Linux, Mac</td>
</tr>
<tr>
<td>Thingstream</td>
<td>Close</td>
<td>C, C++, Java, JavaScript, Python, Go</td>
<td>-</td>
</tr>
<tr>
<td>VerneMQ</td>
<td>Open</td>
<td>Erlang</td>
<td>Linux, Mac</td>
</tr>
<tr>
<td>RabbitMQ</td>
<td>Open</td>
<td>Erlang</td>
<td>Windows, Unix, Mac, Linux, BSD</td>
</tr>
<tr>
<td>ActiveMQ</td>
<td>Open</td>
<td>Java</td>
<td>Windows, Unix, Linux, Cygwin</td>
</tr>
<tr>
<td>ActiveMQ Artemis</td>
<td>Open</td>
<td>Java</td>
<td>Windows, Unix, Linux, Cygwin</td>
</tr>
</tbody>
</table>
Chapter 3. IMU Survey

Massive use of inertial sensors in embedded systems was enabled by the development of the Micro-Electro-Mechanical Systems (MEMS). Researchers explored the miniaturization of mechanical systems, including accelerometers and gyroscopes, using semiconductor fabrication techniques. Inertial sensors quickly found the place in auto and computer industry, that created huge need and increased precision and sophistication of MEMS inertial sensors in use today. Initial implementations were one axis accelerometers, followed by 2 and 3-axis accelerometers, gyroscopes, and magnetometers for the fully integrated 9 degrees of freedom sensors (9DOF).

3.1 3-axis Accelerometer

The embedded microcontroller board we used in the water bottle uses NXP 3-axis accelerometer MMA8653 (See Figure 3.2). The main features of the accelerometer include:

- ±2g, ±4g, ±8g selectable ranges
- 10-bit digital output
- I²C interface
- 1.95V to 3.6V voltage supply

This sensor is placed on the main board located at the base of the cup. For our system, we use the ±2g sensitivity, since the acceleration of movements when drinking does not usually exceed 2g. Since this is an inexpensive accelerometer, there was an initial concern that the sensor may not be calibrated correctly, that would be critical for estimation of bottle angles. In order to check that, we performed an experiment to validate the need for calibration. The experiment consisted in reading values for 1g and -1g on each of the three axis; then, the points were plotted and a line of best fit was drawn in the graph as seen in Figure 3.1. As we can observe, the line almost perfectly fits a line with function $y = x$, which is quite acceptable and should not introduce a lot of error for real measurements.

![Figure 3.1: Three graphs showing calibration for each axis in the accelerometer.](image)
Since the sensor is placed at the base of the cup, there is the risk of losing sensitivity due to heat. The MMA8653 specification indicates that the operating temperature of the sensor is between -40°C and 85°C. The zero-g level change vs temperature is of $\pm 0.27$ mg (mili g) per degree Celsius. In addition, the sensitivity change is of $\pm 0.0074\%$ per degree Celsius.

### 3.2 9-axis IMU

As an external sensor, a more robust 9-axis accelerometer is used placed outside of the case. We used a high performance 9DOF sensor Adafruit BNO08x with internal signal processing powered by MotionEngine™ processor to provide calibrated outputs and software implemented virtual sensors, like quaternions. Among it’s characteristics, we find these:

- Voltage regulator for input voltage
• I2C interface

• SPI interface

• UART interface

• UART-RVC interface

Ideally, our desire was to connect the sensor via I2C to the Teensy board. However, this option was discarded due to the lack of dedicated I2C connector on the controller board. We decided to go with the UART-RVC option instead, which resulted in limited functions that could be used from such a powerful sensor. Another option we considered was to connect the sensor to the secondary board that is used for wireless data transmission. However, Adafruit specifies in their documentation that the sensor does not perform well with ESP32 based boards, and we verified that with unsuccessful attempts of connecting the sensor via I2C. Additionally, simple UART needs a buffer size of at least 300 bytes, which is very big for the controllers being used so the best solution was to go with the UART-RVC connection. Future implementations should use a dedicated I2C connector or use of the IMU controller as a daughter card on the main controller. That would allow controller access to all the features, and users with a more sophisticated monitoring of activity.

The RVC in the interface’s name stands for Robot Vacuum Cleaners, since it is a simplified UART interface to use on unmanned moving robots. The main advantage is that the wiring is simpler than over normal UART. Here, only the
UART TX pin is used, and the sensor transmits acceleration and heading information (yaw, pitch, and roll) at a data rate of 100 Hz.

Figure 3.3: Adafruit 9-DOF sensor.
Chapter 4. Assessment of the Amount of Liquid in a Container Using Inertial Sensors

4.1 General Idea

The Smart water bottle contains an accelerometer that is used to detect when the bottle is used. We want to use the existing sensor to assess the amount of liquid in the bottle at the end of drinking based on the final angle of the bottle. Robust measurement can augment measurements from the existing capacitive sensor or serve as the only sensing mechanism in another model of the bottle. This chapter discusses the procedures followed to estimate the volume of water inside the glass. We tested the method using two different sensors: an inexpensive on-board accelerometer on controller placed at the base of the bottle and a high performance inertial sensor placed on the handle of the cup.

In order to develop an algorithm to use in the embedded controller, we first developed a theoretical model to estimate the volume of water left in a cylinder depending on the tilting angle. The model assumes a perfect cylinder, which as will be shown in later sections, introduces some error when use with off-the-shelf glass cup.
4.2 Theoretical model

The objective is to develop a mathematical function that models the volume of water present in a cylindrical cup after being tilted at a given angle. We will assume first a perfect cylinder for volume calculations. The function would generate the volume for a given tilt angle. This would allow to find the remaining volume of water in the bottle whenever the user takes a sip from the cup, based on the minimum angle the user had to tilt the cup.

We identify three different cases for the model: 1) cup is full (no tilting, so volume is maximum), 2) cup is tilted but water covers the bottom of the cup, and 3) cup is tilted and water covers the bottom only partially.

4.2.1 Case 1: Perfect Cylinder

This case is the most trivial. Since we want to find the volume of water in the cup and the cup is full (i.e. not tilted), the volume of water corresponds to a perfect cylinder as follows:

\[ V = \pi r^2 H, \]  

where \( H \) is the length of the cup’s wall and \( r \) is the radius of the base and top circumferences of the cylinder. However, this remains true only for one point in the model, the full bottle.
4.2.2 Case 2: Cylindrical Segment

While the water completely covers the bottom of the cup, remaining volume can be represented as a cylindrical segment, as shown in Figure 4.1. The shape of the water and the cylinder as seen from the side looks like figure, where $\theta$ is the angle displaced from the vertical position and $\alpha$ is the theoretical angle the inertial sensor reads. Note that due to the inclination of the cup the water’s surface aligns with the $x$ axis. The volume of the blue section can be found as follows:

$$V = \pi r^2 \left( H - \frac{h_2}{2} \right).$$

(4.2)
We already know $H$, since it is the length of the cylinder (depth of the cup), so the next step is to find $h_1$. We do so first defining that $h_1 = H - h_2$. Next, we need to determine $h_2$ based on the tilt of the cup. We observe that the segment that is not filled with water resembles a triangle, so since we know $\alpha$ and $2r$ (diameter of the base of the cylinder), we get $h_2$ from the formulas 4.3 and 4.4:

$$\tan \alpha = \frac{2r}{h_2} \quad (4.3)$$

and so

$$h_2 = \frac{2r}{\tan \alpha}. \quad (4.4)$$

This will remain true and valid while $h_2 \leq H$ since from that point the figure the water makes changes completely.

### 4.2.3 Case 3: Cylindrical Wedge

The moment the water stops covering the base of the cylinder, the shape transforms from a cylindrical segment to a cylindrical wedge, as shown in Figure 4.2. This is a shape created by a plane intersecting a cylinder through its base [25].

In order to derive the volume as a function of $\alpha$, we need to define some variables. We will denote the length of the wedge as $r_1$. The wedge is the segment that goes from the wall of the cylinder to the center of the plane intersecting the base (or in our case the water), see Figure 4.3. In addition to this, we will denote $\phi$ the angle between the radius and the wedge. While $\phi \geq 90^\circ$ we will denote it's
complementary as $\beta$ ($\beta = 180 - \phi$). From the geometry of the tilted cup as seen in Figure 4.3, we get that

$$\tan \alpha = \frac{r_1}{H}$$

(4.5)

and

$$r_1 = \tan \alpha \cdot H.$$  

(4.6)

For both cases, we will need an extra segment which we will denote as $r_3$. This is used to represent the segment between the center of the cylinder’s base

**Figure 4.2:** Cylindrical wedge.
Figure 4.3: Water in the cup.

and the plane. It is needed to estimate angle $\phi$ using trigonometric identities:

$$r_3 = r_1 - r.$$  \hfill (4.7)

Then using trigonometry we find that

$$\cos \beta = \frac{r_3}{r}.$$  \hfill (4.8)

and thus

$$\beta = \arccos \left( \frac{r_3}{r} \right).$$  \hfill (4.9)
After these computations we get $\phi$ by the operation $\phi = 180 - \beta$ and then we can compute the volume of the wedge using the following formula [25]:

$$V = \frac{Hr^2}{3} \left( 3 \sin \phi - 3\phi \left( \frac{\pi}{180} \right) \cos \phi - \sin^3 \phi \right).$$  \tag{4.10}

Note that since we are using $\phi$ in degrees, we need to transform it into radians when using it in the volume formula to get correct results. The volume formula will not change till the cup is empty. What changes is how we get the measurements of the different components of it. Once the plane has past the center of the base ($\phi < 90^\circ$), the wedge ($r_1$) becomes smaller than the radius, meaning that now

$$r_3 = r - r_1$$  \tag{4.11}

(see Figure 4.2). And since now we don’t need angle $\beta$

$$\phi = \arccos \left( \frac{r_3}{r} \right).$$  \tag{4.12}

We then apply equation 4.10. This will be valid till the inclination of the cup is tilted $90^\circ$, which means $\alpha = 0^\circ$ and thus $\theta = 90^\circ$ and cup is empty.

As an illustration, the theoretical model of the water volume as a function of the tilt angle for a cup of radius $r = 3.1$ cm and height $H = 15$ cm (shown in Fig. 4.2a) is illustrated in Figure 4.4.
4.3 Experimental Evaluation and Validation

To validate theoretical model for water volume, we recorded a total of 45 sips, by measuring tilt angle using inertial sensors and remaining volume using precision weight scale. We used the UAH Serial App[26] to get continuous stream of data from inertial sensors: $x, y, z$ acceleration of both on-board and off-board sensors and heading from the off-board sensor. After each sip, the final volume in the cup was measured and saved. Since what we are interested in is the angle, for validation only pitch (heading on the $y$-axis) was used from the off-board sensor and angle of the $z$-axis respective to the $x−y$ plane. Angle $\alpha$ is calculated in
degrees following 4.13 for the on board accelerometer since it only has 3 axis:

\[ \alpha = \arctan \left( \frac{a_z}{\sqrt{a_x^2 + a_y^2}} \right). \]  

(4.13)

Local minima of sensor angles were recorded for each sipping event, as the biggest difference of the angle from the initial position for each sip. Since the on-board accelerometer has no internal signal processing, a 20th order low-pass filter was implemented in Matlab for more clear data. Results can be seen in Figure 4.5. After applying the filter, the signal from the inexpensive on-board sensor is even cleaner than that of the off-board and more sophisticated one. The results of these experiments are shown in Figure 4.6.

As it can be observed, the points seem to fit the curve fairly well. The next step taken was to fit a curve for each of the sensors to get the most suitable function for the volume estimation taking into consideration that the cup is not an ideal cylinder. In order to see how each sensor performed compared to the theoretical model, the error of each measurement with respect to the theoretical model was calculated using equation (4.14). Here \( v_{th}(\theta) \) is the theoretical volume at angle \( \theta \) and \( v_c(\theta) \) is the volume measured at the same angle \( \theta \):

\[ E = v_{th}(\theta) - v_c(\theta). \]  

(4.14)

The results indicate that the on-board accelerometer has much smaller mean estimation error and standard deviation. Although the result is not expected, it is possible that the higher error of the off-board but higher precision sensor is
Figure 4.5: Acceleration and tilt angle during the drinking events.

caused by a different method of attachment of sensor. The sensor should have been glued to eliminate any suspicion about the nature of the result.

Table 4.1: Error comparison vs theoretical model.

<table>
<thead>
<tr>
<th>Statistic</th>
<th>On-Board</th>
<th>Off-Board</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max</td>
<td>22.8919</td>
<td>10.6039</td>
</tr>
<tr>
<td>Min</td>
<td>−25.6770</td>
<td>−69.3859</td>
</tr>
<tr>
<td>Mean</td>
<td>1.27</td>
<td>−14.0640</td>
</tr>
<tr>
<td>Std Dev</td>
<td>12.3275</td>
<td>20.1218</td>
</tr>
</tbody>
</table>
4.3.1 Off-board Sensor Fitting

Since the theoretical model did not provide sufficient accuracy for a real cup, we decide to create cup-specific formula by fitting the experimental results. As indicated in the previous section, theoretical volume calculation uses two different formulas for two cases: cylindrical segment and cylindrical wedge. Therefore, sensor data was divided into two subsets. To make the fitting as accurate as possible we made use of overlapping on the sets as seen in Figure 4.7. Then, using Matlab’s fitting tool the equation for the piece-wise function was obtained,

Figure 4.6: Sensor data vs Model.
Figure 4.7: Piece-wise fitting of experimental data for off-board sensor.

dealing

\[
f(x) = \begin{cases} 
-0.02626x^2 - 1.039x + 466.1 & \text{for } x \leq 62.6072 \\
0.005094x^3 - 1.205x^2 + 82.79x - 1412 & \text{for } x \geq 62.6072 
\end{cases} \tag{4.15}
\]

Since the model is divided into two main portions, the degree of the fitting polynomial is also different. The first part is a second degree polynomial whereas the second one is more complex and requires a third degree polynomial. The final combined fitted curve is shown in Figure 4.8.
4.3.2 On-board Sensor Fitting

Following a similar process to that of the off-board sensor, the data was divided into two overlapping subsets and the two portions were fitted with the fitting tool as seen in 4.9. Fitted equation is provided in equation (4.16)

\[
f(x) = \begin{cases} 
-0.03106x^2 - 1.018x + 463 & \text{for } x \leq 61.0799 \\
0.004996x^3 - 1.13x^2 + 74.3x - 1176 & \text{for } x \geq 61.0799 
\end{cases}
\] (4.16)

and combined fitting as shown in Figure 4.10.
4.3.3 Validation of Results

For validation of the fitted lines, a similar procedure to the comparison with the theoretical model was followed. In this case, statistics were gathered for each sensor and its fitted line. As it can be observed in Table 4.2, the on-board sensor achieves a better fitting and smaller error than theoretical model that assumes a perfect cylinder.

A comparison of the two fitted lines as compared to the theoretical model can be observed in Figure 4.11. For the small tilt angles, the lines fit very well. However, towards the end of the graph the lines start to deviate from the ideal theoretical model. A possible solution to this issue would be to gather more data.

Figure 4.9: Piece-wise fitting of experimental data for on-board sensor.
Figure 4.10: Fitted line on-board sensor.

points, since only 45 experimental measurements may not be enough, particularly for acquisition of angles from inertial sensors.

Table 4.2: Error comparison vs correspondent fitted line.

<table>
<thead>
<tr>
<th>Statistic</th>
<th>On-Board</th>
<th>Off-Board</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max</td>
<td>15.0811</td>
<td>37.4480</td>
</tr>
<tr>
<td>Min</td>
<td>-29.3847</td>
<td>-437574</td>
</tr>
<tr>
<td>Mean</td>
<td>-0.4499</td>
<td>-0.6585</td>
</tr>
<tr>
<td>Std Dev</td>
<td>9.8097</td>
<td>15.6927</td>
</tr>
</tbody>
</table>
Figure 4.11: Comparison between theoretical model and fitted data points.
Chapter 5. Smart Cup as an IoT Appliance

5.1 Existing Literature

There has been numerous implementations of smart cups and/or smart bottles with the objective of tracking water-intake. A lot of different methods have been implemented ranging from capacitance sensor in the cup [7][27], RGB sensors that analyze not only how much water is left in the cup but also what type of liquid is present[28], and also inertial sensors[29][30][31]. Some other methods include not tracking the glass itself but use wearable sensors to estimate volume of water intake[32].

In [29] the authors used acceleration data (as opposed to us using tilting angle) to estimate the water intake of users by placing a sensor attached to a bottle and tried three different ML (Machine Learning) approaches: SVM (Support Vector Machine), artificial neural network, and naïve bayes. Their line of thought was that humans drink more water when the event lasts longer and/or the cup tilts faster. Their work showed that artificial neural network provided the best performance yielding 99% success as shown in Figure 5.1. The system was connected to a smartphone via Bluetooth.

It is commonly known that ML is increasingly becoming more popular, and as embedded micro processors become increasingly more powerful, the op-
The research in [30] [31] explores similar implementations as [29] with artificial neural networks, but implemented using an embedded learning system with a pre-trained neural network that estimated the volume of sipping events based on a single axis acceleration. The platform chosen for this was the Arduino Nano 33 with BLE (Bluetooth Low Energy). This work dives more in depth regarding the use of ML in embedded micro controllers taking into account metrics like processing time, energy consumption to accuracy ratio depending on the number of neurons of the network, etc. The main difference between these approaches and ours is that we base the volume drunk by the user using the angle of inclination of the cup rather than the features of the acceleration vector.

Inertial sensors are not the only explored option to monitor hydration of patients. Other solutions include the use of capacitive sensors [27], [33]. Kreutzer et al placed a sensor on the wall of the cup and measured the accuracy of the measurement of the volume for different liquids at different temperatures, see Figure 5.2. They found out that a big inconvenience of their system is that the
temperature of the fluid can affect the reading of the sensor. However, this only happened for extreme temperatures.

![Capacitance study](image)

**Figure 5.2:** Capacitance study.

Lastly we find LIDS, the system developed in [28] that, in addition to an inertial sensor, uses ultrasonic, temperature, and RGB color sensor. The ultrasonic sensor is used to estimate the volume of liquids left in the bottle. The sensor is placed at the top, and whenever the bottle closes, it measures the volume left. This is a very suitable way of isolating events, meaning that if the bottle is moving, if the ultrasonic sensor does not detect a change in volume, then that event is discarded. The RGB sensor adds the feature of liquid identification. Authors were able to identify nine different liquids with less than 2% of absolute error (see Figure 5.3).
5.2 System Architecture

Our system consists of a smart cup and a server. The cup communicates via WiFi using MQTT protocol, seamlessly monitoring volume intake of the end user. Data is gathered from two inertial sensors, accelerometer placed at the base of the cup, and another higher-end inertial sensor in the handle of the cup. The smart cup case is a 3-D printed case that fits a cylindrical glass cup, as shown in Figure 5.5. The home server is installed on a Raspberry Pi running Home Assistant. Whenever the cup is connected to internet through a local WiFi network, data can be published to the server from anywhere.

Figure 5.3: Liquid identification absolute error.
5.3 System Organization

The smart cup consists of two main embedded platforms, two inertial sensors, and a server used for wireless interfacing. The embedded platforms include a TeensyLC compatible euHy embedded controller used for sensor data acquisition and an ESP32 platform for processing and wireless communication.

5.3.1 euHy Controller

euHy controller is an application specific controller software compatible with Teensy LC (Low Cost) controller [34]. The controller uses the same processor as Teensy LC MKL26Z64VFT4 Cortex-M0+. The processor integrates a wide variety of hardware peripherals and features excellent capacitive sensing that can be used for volume estimation, and very low power consumption. The board is easily programmable with ArduinoIDE and Teensyduino libraries. The controller has the following features:
• ARM Cortex M0+ Processor MKL26Z64VFT4: A 32 bit processor with a clock speed of 48 MHz designed for low-power applications.

• Memory: 62 kBytes of flash memory and 8 kBytes of RAM, and a 2 kBytes EEPROM.

• 13 Analog Input pins

• Timers: 7 timers with at least 16 bits of resolution.

• 11 Touch sensors (capacitive to digital controllers)

• Communication: Teensy LC contains 1 USB interface, 3 Serial interfaces, 2 I2C, 2 SPI interfaces one of which has FIFO.
The MKL26Z64VFT4 controller works as sensor back end in charge of sensor data gathering. Processed data is sent to the ESP32 controller using serial link, which sends the data to the server using MQTT messages. We make use of the three serial interfaces. The USB serial interface is mainly used for programming the board in the IDE and debugging; the second serial interface is used for

Figure 5.6: System organization.
communication with the ESP32; the last serial interface is used to communicate with the 9-axis accelerometer via UART-RVC as explained in Chapter 3.

![Image of euHy controller and PPG/ECG sensor.](image)

**Figure 5.7:** euHy controller and PPG/ECG sensor.

5.3.2 ESP32 Microcontroller

For the data transmission part of the project, there were two boards that have been used and tested: ESP32 D1 Mini and XIAO ESP32 C3. Their characteristics are shown in Table 5.1.

5.3.3 Inertial Sensors

As specified in Chapter three, there are two inertial sensors connected to the controller: a 3-axis accelerometer and one 9 DOF inertial sensor. The 3-axis accelerometer is connected via I2C to the Teensy processor. The accelerometer is
integrated on the same PCB, this is what has been called the on-board accelerometer. The controller uses the UART-RVC protocol for communication with the off-board sensor since there were available pins for serial communication with the Teensy Board. This introduces a limitation on the functionality of the off-board sensor since UART-RVC only allows limited size and types of messages. ESP32 board is tasked with wireless communications only, to achieve robust real-time operation of the system.

5.3.4 The Board

The Teensy controller board is a custom development board developed by UAH. The board supports the following:

- High Precision capacitance measurement using MKL26Z64VFT4 processor.

- Photoplethysmography (PPG)/ Electrocardiography (ECG) sensor connector capability via I2C. A custom PPG/ECG sensor is used to monitor physiological signals of the user (heart rate and pulse arrival time). Custom board features a single chip controller MAX86150.
• External Battery connector with power switch

• Expansion connector with SPI interface for daughter card featuring bio impedance controller [35]

The functionality of the cup can thus be expanded. Since all the wireless communications are handled externally, the Teensy board can be used to gather and process the physiological signals, therefore transforming the cup to a home health station that provides hydration measurement and physiological monitoring.

5.3.5 Server

Server provides storage, processing, and remote access to data from the smart bottle. Standard cloud servers are costly and might have connectivity issues. We decided to use a Raspberry Pi 3 B+ running Home Assistant software, as a common inexpensive home automation platform. The server is used as MQTT broker. Since the server is accessible remotely, users could receive notifications wherever they are when the cup detects an event depending on the use case given to it. In order to set up the server for remote access, we had to change some network configurations parameters. The first one is setting up a dynamic DNS, which we did using a free DNS service called duckDNS. In addition to that, since most routers have a solid firewall, port forwarding needed to be enabled in the network on ports 8123 to enable the UI access and port 1883 to enable MQTT traffic.

The Home Assistant UI is comprised of three different cards. A card is a UI element that tracks the state of an entity. We implemented the following
cards: the number of events registered that day, the volume of liquid consumed for each minute for the last hour, and the total amount of liquid consumed during the day. If the user makes more than one drinking event in one minute, the value shown in the volume per sip card will be the average of all the sips in that minute.

HomeAssistant can be reached on the local network using the URL homeassistant.local:8123, and if the user is outside the local network the DNS domain has to be used.

![Home assistant view of the water consumed.](image)

**Figure 5.8:** Home assistant view of the water consumed.
In addition to accessing the server from the web interface, the users can also access it from the Home Assistant app. An automation is set up to allow push notifications on the phone whenever an event is detected. The picture in Figure 5.9 shows what the user sees when an event is triggered. The app in the background is an MQTT processing app [36] from Apple’s store that allows for easy debugging and monitoring of the MQTT messaging.

![Home Assistant push notification](image)

Figure 5.9: Home assistant push notification.
5.3.6 Software Model

The software is organized in two main models for the micro controllers and one for the server. The main model for the Teensy board is shown in Figure A.1. The Teensy board supports several states that can be changed via the USB Serial interface which are useful for debugging.

Table 5.2: Embedded controller states.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDLE</td>
<td>Wait for Serial input for new state</td>
</tr>
<tr>
<td>IMU SERIAL APP</td>
<td>Send data to serial app for debugging</td>
</tr>
<tr>
<td>RUN</td>
<td>Main running mode</td>
</tr>
</tbody>
</table>

The IMU SERIAL APP mode simply takes data from both sensors and transfers them via Serial to UAH Serial App [26], where data can be plotted in real time for debugging and saved in a file for debugging and post processing.

Monitoring of activity and bottle use is also performed using on-board accelerometer. The controller continuously monitors the activity vector of the acceleration, as defined in Equation (5.1). Integral of dynamic acceleration (A) is calculated as follows:

\[
A = |\sqrt{a_x^2 + a_y^2 + a_z^2} - 1|, \tag{5.1}
\]

where the acceleration values \((a_x, a_y, \text{ and } a_z)\) are in Gs. When the activity value exceeds a threshold, an event is flagged and remains true until \(A\) falls below
the threshold. In addition, the calculated tilt angle is always recorded for the maximum tilt that will determine the final volume of liquid in the bottle. Since we don’t have a way to measure the water of the cup after a refill, the implementation assumes that when the cup is empty, there is a refill to almost the full volume. This allows us to track the volume consumed by taking the difference of the remaining volume of the last event and the remaining volume after current event. An illustration of how activity and dynamic acceleration change through sips is shown in Figure 5.10. In addition, a more detailed view of these parameters in one sip is provided in Figure 5.11.

**Figure 5.10:** Activity and acceleration of on board sensor during 10 sips.
Figure 5.11: Activity and angle of both sensors.

Figure A.2 shows the software flowchart of the algorithm in the RUN state. The system defaults to this mode since it is meant to run on an external battery. Since it has been programmed using Arduino IDE environment, the code is divided into two main sections: the setup and the loop. The setup portion is ran only
once, whereas the loop will run constantly until the controller is reset or rebooted. In the setup section, in addition to initialization of variables, a calibration routine can be initiated to get the initial angle of the off-board accelerometer in order to compensate for the inaccuracies of the sensor mounting on the side of the cup. We measured initial error as up to 2 degrees. The setup section also initializes communication with peripherals (I2C and Serial).

The ESP controller connects to WiFi and waits for serial messages. The data transmitted to the server from the cup contains the minimum angle, the time of the last event, and the volume of water taken in the last event. The time and angle values are published to the server in the form of JSON message under the topic \textit{SMARTCUP/values}, whereas the volume is published under the topic \textit{SMARTCUP/volume}. The format the controller expects is a character followed by a float value with no spaces. This is very useful since it is also scalable. In future implementations where capacitance sensing and other sensors are included it will be easy to implement following this packet format:

\texttt{<t><float time [s]><d><float angle [deg]><v><float volume [ml]>}

Figure 5.12 shows the MQTT topics used in the system. The status topic is useful for debugging; every time the controller connects to the client, a message is published.

As shown in Figure 5.8, there are three main widgets that are updated with the smart cup data. The bar graph for the sips taken in the last hour is updated listening to the volumes topic, whereas the other two cards (the event counter and the daily volume) are updated within the server using Home Assistant’s
automations utility. There are four server automations as described in Table 5.3 that make sure the data is correctly updated.

**Table 5.3:** Server automations.

<table>
<thead>
<tr>
<th>Automation</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Increment Counter</td>
<td>Increments the number of events when volume payload is not zero and sends push notification to app</td>
</tr>
<tr>
<td>Reset Daily Values</td>
<td>Resets the event counter and daily volume every day at midnight</td>
</tr>
<tr>
<td>Update Daily Volume</td>
<td>Adds the last volume event to the daily volume entity</td>
</tr>
<tr>
<td>Update Volume Minute</td>
<td>If there has been more than one minute since the last event, publish a 0 in the volume topic to update the bar graph of the sips in an hour.</td>
</tr>
</tbody>
</table>
5.4 Assessment of Tremor of Parkinson Patients

Although the system prototype is meant to track daily hydration habits on end users, inertial sensors can support additional functions. Early detection of Parkinson’s disease and monitoring of tremor throughout the day to assess effects of drugs represent a very important possible use case for the smart water bottle.

As a proof of concept, we conducted an experiment in which the smart cup was held still for a certain amount of time, and then we simulated a tremor typical for Parkinson’s patients. Embedded application could assess fundamental frequency and amplitude of the motion. As seen in Figure 5.13, a simulated tremor is presented from 21 to 31 seconds. A simulated tremor is clearly visible and can be processed to assess the changes in tremors throughout the day. Dominant frequency of the tremor is calculated for windows with activities larger than a threshold using zero crossing method and shown in Figure 5.13.
Figure 5.13: Tremor experiment holding the cup still.
Chapter 6. Conclusion

Seamless health monitoring enabled by the Internet of Things (IoT) presents an opportunity for the continuous and real-time collection of user status, ranging from vital signs to activity levels. By integrating a network of smart devices, such as wearables, ambient sensors, and medical equipment, IoT solutions provide a more holistic view of an individual’s health in their homes, during activities of daily living. Artificial intelligence applications running in the background on the server could facilitate timely interventions, proactive approach, and personalized healthcare.

This study was focused on the use of smart water bottle as IoT appliance for monitoring of user’s hydration and activity. We developed a theoretical model to estimate the amount of liquid left in a cup/water bottle based on the tilt angle at the end of the sip. We integrated a system to track the amount of water consumed and supported remote access by patients, physicians, and caregivers.

Original contributions of the thesis include:

- Development of the theoretical model to estimate the amount of water in a cup based on the final tilt angle.
• Development of embedded program to monitor water consumption and activity of users and data collection on the cloud.

• Validation of algorithms for assessment of water consumption.

Future work includes:

• Multimodal sensing using inertial and of capacitance sensors to estimate the volume of the cup when its refilled, and facilitate a more precise estimation and monitoring when a straw is used to drink water from the cup.

• Adding activity recognition to classify drinking and spilling events.

• Integration of physiological monitoring using PPG, ECG and bio-impedance sensors on euHy controller for monitoring of health status and objective hydration, and use of the cup as a monitoring station in Internet of Medical Things (IoMT).
References


Appendix A. Software Diagrams

Figure A.1: Teensy board main flow.
Figure A.2: Flow chart of the algorithm for detection and upload of drinking events.
Figure A.3: Flow chart of the ESP board main flow.