

# TRIWAVE SYSTEMS

July 7, 2019

Dr. Craig Scratchley  
Dr. Andrew Rawicz  
School of Engineering Science  
Simon Fraser University  
Burnaby, British Columbia  
V5A 1S6

RE: ENSC 405W/440 Design Specifications for Arkriveia Beacon

Dear Dr. Scratchley and Dr. Rawicz,

The following document contains the Design Specification for Arkriveia Beacon - The Indoor Location Rescue System created by TRIWAVE SYSTEMS. The Arkriveia Beacon focuses on locating personnel trapped in buildings during small scale disasters such as fires and low magnitude earthquakes. This is achieved by incorporating combinations of advanced Ultra-wide-band radio modules and microcontrollers to create a dependable indoor positioning system using trilateration. We believe our system allows search and rescue operator to safely and reliably locate victims during an emergency or disaster.

The purpose of this document is to provide low and high-level design specifications regarding the overall system architecture, functionality and implementation of the Arkriveia Beacon system. The system will be presented according to three different development stages: proof-of-concept, prototype, and final product. This document consists of system overview, system design, hardware design, electrical design, software design and as well as a detailed test plans for the Alpha and Beta products.

TRIWAVE SYSTEMS is composed of five dedicated and talented senior engineering students. The members include Keith Leung, Jeffrey Yeung, Scott Checko, Ryne Watterson, and Jerry Liu. Each member came from various engineering concentrations and with a diverse set of skills and experiences, we believe that our product will truly provide a layer of safety and reliability to emergency search and rescue operations.

Thank you for taking the time to review our design specifications document. If there are any further questions or comments, please direct them to our Chief Communications Officer Jeffrey Yeung at [zjyeung@sfu.ca](mailto:zjyeung@sfu.ca)

Sincerely,  
Jerry Liu  
Chief Executive Officer



Enclosed: Design Specification for Arkriveia Beacon

# TRIWA<sup>A</sup>VE SYSTEMS

## ENSC 405W

---

AKRIVEIA BEACON

Design Specification

Team 5

07/07/2019

Project Team   Jeffery Yeung CCO  
                     Keith Leung CTO  
                     Scott Checko COO  
                     Ryne Waterson CIO  
                     Jerry Liu CEO

Contact   Jeffery Yeung  
                 zjyeung@sfu.ca

Submitted to   Dr. Craig Scratchley  
                     Dr. Andrew Rawicz  
                     School of Engineering Science  
                     Simon Fraser University

Issue Date   July 07, 2019

## Abstract

The Akriveia Beacon by TRIWAVE SYSTEMS focuses on improving the locating and rescue process of personnel trapped in buildings during and after small scale disasters such as fires and low magnitude earthquakes. The Akriveia Beacon system allows search and rescue operations to safely and reliably locate trapped victims in a disaster situation confined within complex urban environments. By pinpointing the exact location of any victim wearing an ID tag, the search and rescue time for first responders is minimized; which is crucial in any disaster rescue operations.

The Akriveia Beacon consists of various hardware, electrical and software components in order to create a product that can accurately and reliably to locate and identify ID tags within commercial building structure in near real time. This is achieved by incorporating a combination of advanced Ultra-wideband radio modules, microcontroller units, data processing units, and reliable trilateration techniques to create a dependable indoor location positioning system.

The design specifications outlined in this document details the specifications of design elements such as the high-level system architecture, functionality, and implementation for each critical section of the Akriveia Beacon. Sections include design for the System Components, Hardware, Electrical, and Software components of the product. The specifications will also cover the three phases of product development: Proof-of-concept (PoC) phase, Prototype phase, and Final Product phase. Additional appendices for the user interface design and test plans are included at the end of this document. These appendices outlines the intended development time line and task assignment breakdown for the development of the Akriveia Beacon.

TRIWAVE SYSTEMS is dedicated to creating a reliable and robust system design to improve disaster search and rescue operations with human safety as the pivotal focus.

## Glossary

- Actix** An implementation of the Actor Model in Rust. 38
- Actix Web** A webserver library built on top of Actix. 38
- Actor** A class that contains message handling callbacks. 38
- Actor Model** A concurrency management paradigm that uses message passing between objects rather than locks or atomics. 38
- Aptitude** A debian package manager that automatically manages updates and installation of software and dependencies. 37
- Arbitor** A thread pool, where each thread is an event loop. 40
- ARM64** Advanced RISC Machine, A family of RISC based processors. 37
- Babel** A scripting language similar to Javascript with the ability to directly construct HTML snippets. 48
- Borrow Checker** A component of the Rust compiler to prevent data races by enforcing data ownership rules. 38
- CSA** Canadian Standards Association is a standards development organization. 70
- DC** Direct current voltage. 24
- Debian** A Linux based operating system. 37
- DHCP** Dynamic Host Configuration Protocol. 27
- ETSI** European Telecommunications Standards Institute. 23
- FCC** Federal Communications Commission. 23
- FM** Frequency modulation. 15
- Garbage Collector** A runtime component of some programming languages that detects and cleans up unused memory.. 38
- GPIO** General Purpose Input Output. 22
- Hostapd** Host access point daemon. 27
- HTTP** HyperText Transfer Protocol. 39
- ID** Identification. 8
- IEC** International Electrotechnical Commission. 70
- IEEE** Institute of Electrical and Electronics Engineers. 70
- ISO** the International Organization for Standardization.. 70
- JQuery** A Javascript library designed to manipulate HTML. 49

**MAC** Media Access Control. 28

**MPSC** Multiple Producer Single Consumer Queue. 42

**MVC** Model View Controller, a method of organization for GUI implementations. 42

**OS** operating system. 37

**OSI Model** Open Systems Interconnection model. 26

**PCB** Printed circuit board. 14

**PLA** Polylactic acid or polylactide. 29, 31

**PoC** Proof of concept is the sample product assembled to explore project feasibility. 12

**React** A Javascript framework to dynamically generate HTML. 48

**RF** Radio Frequency. 15

**Rolling Release** Frequent updates of software, without versions. 37

**RSSI** Received Signal Strength Indicator. 8, 12

**Rust** A systems language that focuses on reliability and performance. 38

**Server Side Rendering** Creating HTML files on the server, which can then be directly consumed by the browser without modification to render a GUI. 44

**ToF** Time-of-Flight is a method for measuring the distance between a sensor and an object. 11

**UDP** User Datagram Protocol. 11

**UWB** Ultra wide band. 8

**x86-64** Intel designed CISC family of processors. 37

## Contents

|  |           |
|--|-----------|
| <b>Contents</b>                                  | <b>6</b>  |
| <b>List of Figures</b>                           | <b>8</b>  |
| <b>List of Tables</b>                            | <b>9</b>  |
| <b>1 Introduction</b>                            | <b>10</b> |
| 1.1 Background . . . . .                         | 10        |
| 1.2 Scope . . . . .                              | 11        |
| 1.3 Intended Audience . . . . .                  | 11        |
| 1.4 Design Classification . . . . .              | 12        |
| <b>2 System Overview</b>                         | <b>13</b> |
| 2.1 Proof of Concept . . . . .                   | 14        |
| 2.2 Prototype . . . . .                          | 15        |
| 2.3 Final Product . . . . .                      | 16        |
| 2.4 RF Fundamentals . . . . .                    | 17        |
| 2.5 Received Signal Strength Indicator . . . . . | 19        |
| 2.6 Ultra-Wideband and Time-of-Flight . . . . .  | 19        |
| 2.7 Trilateration Methods . . . . .              | 20        |
| 2.8 System Design Specification . . . . .        | 22        |
| <b>3 System Components</b>                       | <b>23</b> |
| 3.1 MCU - ESP32 . . . . .                        | 23        |
| 3.2 Transceiver - DWM1000 . . . . .              | 25        |
| 3.3 DPU - Raspberry Pi . . . . .                 | 26        |
| <b>4 Hardware Design</b>                         | <b>27</b> |
| 4.1 System Operation Modes . . . . .             | 27        |
| 4.2 Communication Protocol . . . . .             | 28        |
| 4.2.1 Beacon to ID Tag Communication . . . . .   | 28        |
| 4.2.2 Beacon to DPU Communication . . . . .      | 29        |
| 4.3 Beacon Design . . . . .                      | 30        |
| 4.4 ID Tag Design . . . . .                      | 32        |
| 4.5 Hardware Design Specification . . . . .      | 34        |
| <b>5 Electrical Design</b>                       | <b>35</b> |
| 5.1 Power Management . . . . .                   | 35        |
| 5.2 RF Harvester . . . . .                       | 37        |
| 5.3 Electrical Design Specification . . . . .    | 38        |
| <b>6 Software Design</b>                         | <b>39</b> |
| 6.1 Software Overview . . . . .                  | 39        |
| 6.2 Software Stack . . . . .                     | 39        |
| 6.2.1 Software Environments . . . . .            | 39        |
| 6.2.2 Software Languages . . . . .               | 40        |
| 6.2.3 Software Standards . . . . .               | 40        |
| 6.2.4 Frameworks . . . . .                       | 40        |
| 6.2.5 Libraries . . . . .                        | 41        |

|           |  |           |
|-----------|--|-----------|
| 6.3       | Model-View-Controller . . . . .                  | 41        |
| 6.4       | Threading Model . . . . .                        | 42        |
| 6.5       | Data Processor Software Architecture . . . . .   | 44        |
| 6.5.1     | Webserver Subsystem . . . . .                    | 46        |
| 6.5.2     | Beacon Manager Subsystem . . . . .               | 46        |
| 6.5.3     | Serial Beacon Communication Subsystem . . . . .  | 46        |
| 6.5.4     | Trilateration Processing Subsystem . . . . .     | 47        |
| 6.5.5     | UDP Beacon Communication Subsystem . . . . .     | 47        |
| 6.6       | Database . . . . .                               | 47        |
| 6.7       | Models . . . . .                                 | 47        |
| 6.8       | Controllers . . . . .                            | 49        |
| 6.9       | View . . . . .                                   | 49        |
| 6.10      | Security . . . . .                               | 50        |
| 6.11      | Software Design Requirements . . . . .           | 51        |
| <b>7</b>  | <b>Conclusion</b>                                | <b>52</b> |
| <b>8</b>  | <b>References</b>                                | <b>53</b> |
| <b>9</b>  | <b>Appendix A: Supporting Test Plans</b>         | <b>55</b> |
| 9.1       | PoC Test Plan . . . . .                          | 56        |
| 9.2       | Prototype Test Plan . . . . .                    | 58        |
| 9.3       | Final Product Test Plan . . . . .                | 60        |
| 9.4       | Usability Test Plan . . . . .                    | 62        |
| <b>10</b> | <b>Appendix B: User Interface and Appearance</b> | <b>64</b> |
| 10.1      | Introduction . . . . .                           | 64        |
| 10.1.1    | Purpose . . . . .                                | 64        |
| 10.1.2    | Scope . . . . .                                  | 64        |
| 10.2      | User Analysis . . . . .                          | 65        |
| 10.3      | Technical Analysis . . . . .                     | 66        |
| 10.3.1    | Discoverability . . . . .                        | 66        |
| 10.3.2    | Feedback . . . . .                               | 66        |
| 10.3.3    | Conceptual models . . . . .                      | 67        |
| 10.3.4    | Affordances . . . . .                            | 67        |
| 10.3.5    | Signifiers . . . . .                             | 67        |
| 10.3.6    | Mappings . . . . .                               | 67        |
| 10.3.7    | Constraint . . . . .                             | 67        |
| 10.4      | Graphical Representation . . . . .               | 68        |
| 10.4.1    | UI State Diagrams . . . . .                      | 68        |
| 10.4.2    | UI Mock-Ups . . . . .                            | 69        |
| 10.5      | Engineering Standards . . . . .                  | 73        |
| 10.6      | Analytical Usability Testing . . . . .           | 74        |
| 10.7      | Empirical Usability Testing . . . . .            | 75        |
| 10.8      | Conclusion . . . . .                             | 76        |
| <b>11</b> | <b>Appendix References</b>                       | <b>77</b> |

## List of Figures

|    |  |    |
|----|--|----|
| 1  | Development Cycle . . . . .  | 11 |
| 2  | High Level System Layout . . . . .                                 | 13 |
| 3  | PoC System Block Diagram . . . . .                                 | 14 |
| 4  | Prototype System Block Diagram . . . . .                           | 15 |
| 5  | Final System Block Diagram . . . . .                               | 16 |
| 6  | Relationship Between Signal Strength and Distance . . . . .        | 17 |
| 7  | Effects of Different Frequencies on Distance Propagation . . . . . | 18 |
| 8  | An Example of Multipath . . . . .                                  | 18 |
| 9  | Trilateration Diagram . . . . .                                    | 20 |
| 10 | ESP32 Architect Block Diagram . . . . .                            | 23 |
| 11 | ESP32 Development Board . . . . .                                  | 23 |
| 12 | Circuit Diagram of ESP32 & DWM1000 . . . . .                       | 24 |
| 13 | ESP32 Pin Layout . . . . .   | 24 |
| 14 | Decawave DWM1000 Modules . . . . .                                 | 25 |
| 15 | DWM1000 Internal Block Diagram . . . . .                           | 25 |
| 16 | Raspberry Pi 3 B+ Model . . . . .                                  | 26 |
| 17 | Akriveia Beacon System State Diagram . . . . .                     | 27 |
| 18 | Common Signal Power spectral Density VS Frequency . . . . .        | 28 |
| 19 | UDP Communication Network . . . . .                                | 29 |
| 20 | Beacon System Flow Diagram . . . . .                               | 30 |
| 21 | CAD representation of Beacon . . . . .                             | 31 |
| 22 | ID Tag System Flow Diagram . . . . .                               | 32 |
| 23 | CAD representation of ID Tag . . . . .                             | 33 |
| 24 | ESP32 Active Mode Power Usage . . . . .                            | 35 |
| 25 | ESP32 Deep Sleep Mode Power Usage . . . . .                        | 36 |
| 26 | RF Harvester Block Diagram . . . . .                               | 37 |
| 27 | RF Harvester Circuit Diagram . . . . .                             | 37 |
| 28 | Actix Thread Model . . . . .                                       | 43 |
| 29 | Actix Message Passing . . . . .                                    | 43 |
| 30 | Proof of Concept Software Architecture . . . . .                   | 45 |
| 31 | Final Software Architecture . . . . .                              | 45 |
| 32 | UI State Diagram - Primary User . . . . .                          | 68 |
| 33 | UI State Diagram - Secondary User . . . . .                        | 68 |
| 34 | PoC Console UI . . . . .   | 69 |
| 35 | Prototype Box Map Layout View . . . . .                            | 69 |
| 36 | Primary User Map View . . . . .                                    | 70 |
| 37 | Primary User System Status View . . . . .                          | 70 |
| 38 | Secondary User System Status View . . . . .                        | 71 |
| 39 | Add Map View . . . . .   | 71 |
| 40 | Add Beacon View . . . . .  | 72 |
| 41 | Add User View . . . . .  | 72 |



## List of Tables

|    |  |    |
|----|--|----|
| 1  | Design Requirement Encoding . . . . .                  | 12 |
| 2  | Design Domain Abbreviation Code . . . . .              | 12 |
| 3  | Development Stage Encoding . . . . .                   | 12 |
| 4  | System Design Specification . . . . .                  | 22 |
| 5  | Hardware Design Specification . . . . .                | 34 |
| 6  | Electrical Design Specification . . . . .              | 38 |
| 7  | Akriveia Beacon Dependencies - Rust . . . . .          | 41 |
| 8  | Akriveia Beacon Dependencies - Arduino . . . . .       | 41 |
| 9  | User Model . . . . .                                   | 48 |
| 10 | Beacon Model . . . . .                                 | 48 |
| 11 | Map Model . . . . .                                    | 48 |
| 12 | Controllers list . . . . .                             | 49 |
| 13 | Software Design Specification . . . . .                | 51 |
| 14 | Test Case Encoding . . . . .                           | 55 |
| 15 | Planning Stage Abbreviation Code . . . . .             | 55 |
| 16 | PoC System (General) Test Plans - Part 1 . . . . .     | 56 |
| 17 | PoC System (General) Test Plans - Part 2 . . . . .     | 57 |
| 18 | Prototype Test Plans - Part 1 . . . . .                | 58 |
| 19 | Prototype Test Plans - Part 2 . . . . .                | 59 |
| 20 | Final Product Test Plans - Part 1 . . . . .            | 60 |
| 21 | Final Product Test Plans - Part 2 . . . . .            | 61 |
| 22 | PoC Software Requirement Test Plans - Part 1 . . . . . | 62 |
| 23 | PoC Software Requirement Test Plans - Part 2 . . . . . | 63 |
| 24 | Engineering Standards . . . . .                        | 73 |
| 25 | Usability Test Results . . . . .                       | 74 |

# 1 Introduction

## 1.1 Background

Over the past couple of decades, urban centers around the world have faced substantial population growth. As a result, the number of large and complex structures in dense urban areas around the world is rapidly increasing. In Canada alone there are approximating 500,000 commercial buildings [1]. A large population combined with massively complex buildings in relatively dense areas leads to higher risk for damage and casualties in the event of a disaster. Due to increased urbanization and complexity of urban structures, search and rescue operations in indoor urban environments face various complications and uncertainties. According to Statistics Canada, an average of 135 fire related deaths occur with commercial structures each year from 2010 to 2014 [2].

In current practices, first responders know little about the severity of the disaster until arriving on scene. Once responders are on scene, emergency management have to quickly evaluate the situation and take appropriate actions [3]. Assessments of the structure are conducted with readily available blueprints of buildings along with limited information of last known location of possible trapped victims, usually derived from witness reports. Situational data are created dynamically during this process and the actual rescue process heavily depends on the situational awareness of the first line of emergency response operators [4].

An important issue that must be considered is how emergency first responders should be dispatched inside the building in the event of a disaster in order to minimize search and rescue time as well as to ensure their safety. In order to pinpoint locations of trapped victims quickly and accurately it is critical to have precise location data. Proper emergency planning and organization takes a substantial amount of time, and having additional quick and accurate information on the locations of trapped, incapacitated or immobile personnel would greatly improve first responders situational awareness; which would improve safety for first responders and possibly increases the victims chances of rescue and survival.

As such, the need for a distinct indoor positioning rescue system is crucial in getting fast and reliable information that allows first responders to be dispatched within the builds in the most optimal and efficient manner. The Akriveia Beacon by TRIWAVE SYSTEMS focuses on improving the locating and rescue process of personnel trapped in buildings during or after small scale disasters such as fires and low magnitude earthquakes. This is done through a system of Ultra Wide-Band (UWB) Beacons and ID tags, and data processing unit for accurate, near real-time pin point location of trapped personnel.

Ultra-Wideband radio modules are small radio transceivers using radio spectrum within the ultra-wide band to communicate with one another. Each ID tag uses a UWB transceiver module to communicate with the beacon system configured with similar UWB transceivers. Given the time between sending and receiving transmission data, the distance can be estimated via RSSI or time of flight. The Beacons will then forward these distance estimations to a data processing unit using a closed WiFi network, where it can use trilateration methods to calculate the near real time location of each individual ID tag. The system design allows for multiple ID tags as well as more than three anchor beacons to provide more accuracy through redundancy, making it modular, extendible and reliable.

## 1.2 Scope

The Akriveia Beacon system is developed through three different phases of development as shown in Figure 1. The three different phases includes: the proof-of-concept phase, prototype phase, and final product phase. A high-level design of the system hardware and software is presented in this document to demonstrate the overall system architecture and functionality of the Akriveia beacon product. The design section of this document is divided into four main sections, overall system design, hardware design, electrical design, and software design. These design specification will indicate the components, implementations, requirements, and constraints that must be met and satisfied within the project time frame.



Figure 1: Development Cycle

## 1.3 Intended Audience

This document is presented by engineers at TRIWAVE SYSTEMS as a guide for the design and system overview of the Akriveia Beacon product. The intended audience of this document includes but not limited to, potential clients and/or partners, the supervising professors Dr. Craig Scratchley and Dr. Andrew Rawicz, associated teaching assistants and fellow TRIWAVE SYSTEMS members. The hardware and software engineers of the project can reference this document during the various stages of development and testing stages of the project for clarification. Near the completion of the prototype development phase the product will be tested against the cases specified in the test plan. Engineers responsible for performing quality assurance can refer to the Appendix of this document to ensure all safety concerns have been addressed and that the product fulfils all requirements and meets all expectations for proper usage.

## 1.4 Design Classification

For consistency purposes, the following design classification code convention is used to describe and organize design requirements listed throughout this document.

[ DES.SE.# - X ]

| Code | Definition  |
|------|---|
| DES  | Design abbreviation.  |
| SE   | Design Domain Abbreviation Code correspond with each Design requirements. (see Table 2) |
| #    | Design number ID  |
| X    | Development Stage Encoding (see Table 3)  |

Table 1: Design Requirement Encoding

| Requirement Domain | Abbreviation Code |
|--------------------|-------------------|
| System             | SY                |
| Hardware           | HW                |
| Electrical         | EC                |
| Software           | SW                |

Table 2: Design Domain Abbreviation Code

| Development Stage | Encoding |
|-------------------|----------|
| Proof of Concept  | C        |
| Prototype         | P        |
| Final Product     | F        |

Table 3: Development Stage Encoding

## 2 System Overview

The Akriveia Beacon indoor locating rescue system combines hardware, electrical, and software systems to detect and locate multiple occupants within a building during an emergency disaster situation. Each individual component of the system is developed separately in the PoC (Proof of Concept) phase; then partially integrated in the Prototype phase and fully integrated in the Final Product phase.

A high-level system overview presents three Locator Beacons, an ID tag, a data processing unit, and a graphical user interface (Figure 2). Using ultra-wideband (3.5-6.5GHz) wireless communication the Locator Beacons transmit signals to the ID tag to acquire a response. When the response returns back to the Beacon a time of flight measurement is acquired. The Time-of-Flight principle (ToF) is a method for measuring the distance between a sensor and an object, based on the time difference between the emission of a signal and its return to the sensor, after being reflected by an object. [5]. The ToF data will be forwarded to the portable data processing unit via a closed Wi-Fi network with UDP. Then the processing unit will calculate the distance and coordinates of the ID tags using trilateration algorithm. Afterwards, the coordinates results are displayed on a GUI for operators.

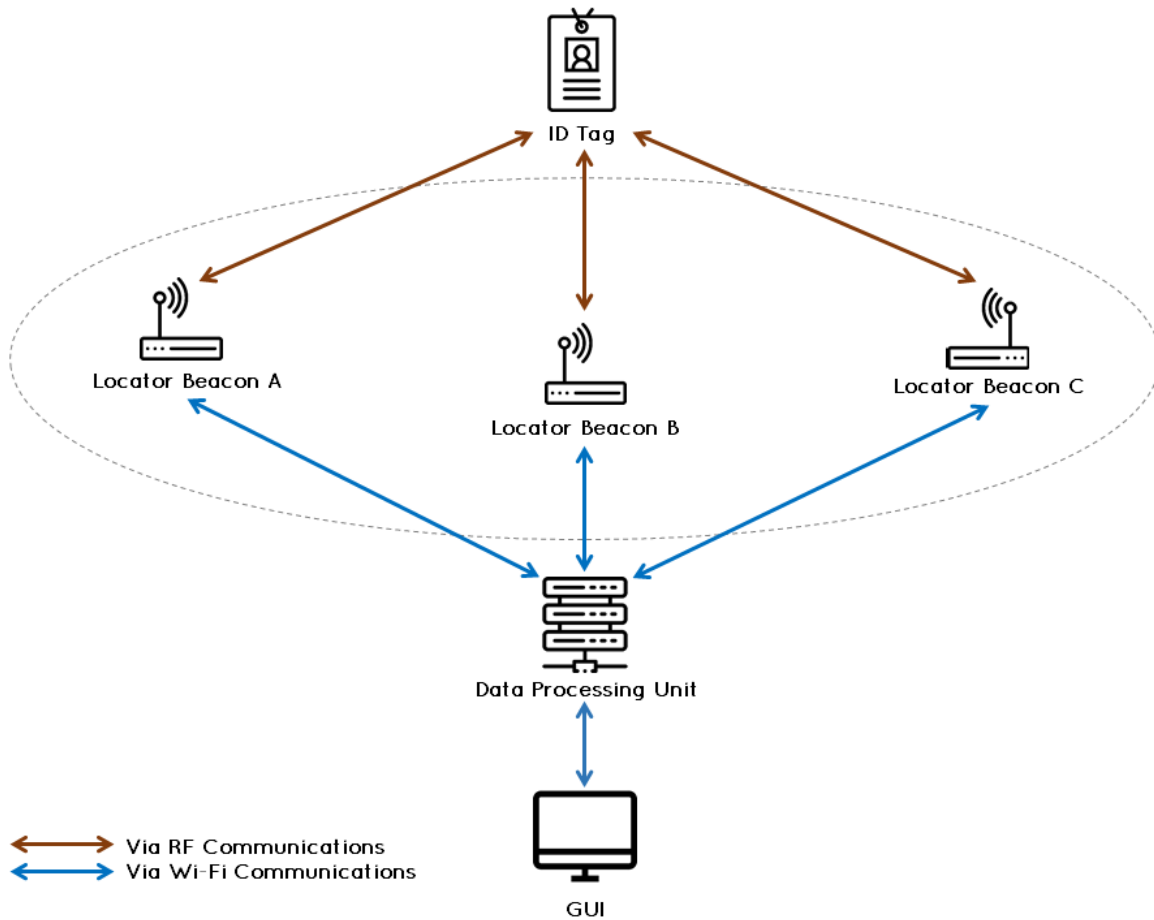


Figure 2: High Level System Layout

## 2.1 Proof of Concept

The Proof of concept phase demonstrates the feasibility and functionality of an indoor location determination system. The PoC system will evaluate how effective the trilateration method is for determining the distance and location of a mobile ID tag in two dimensional space as well as to establish initial feasibility for the system. The PoC will be developed similar to the system block diagram shown in figure 3.

ESP32 micro-controllers are used as the main controller units of the Beacon and ID Tags in PoC and further on. The ESP32 is an off the shelf, low-cost, low-power system on a chip micro-controllers with integrated WiFi and dual-mode Bluetooth. In the PoC, the system is designed to use Received Signal Strength Indicator (RSSI) from Bluetooth Low Energy (BLE) modules of the ESP32 to estimate distance between each beacon and ID tag. Each beacon determines the MAC address and a RSSI measurement from the advertising ID Tag and creates a data packet. The data packet is forwarded to the data processing unit - Raspberry Pi, via USB serial communications. The RSSI is then used to estimate distance between each ID Tag and the associating Beacon and the results of the trilateration method are output to a simple UI.

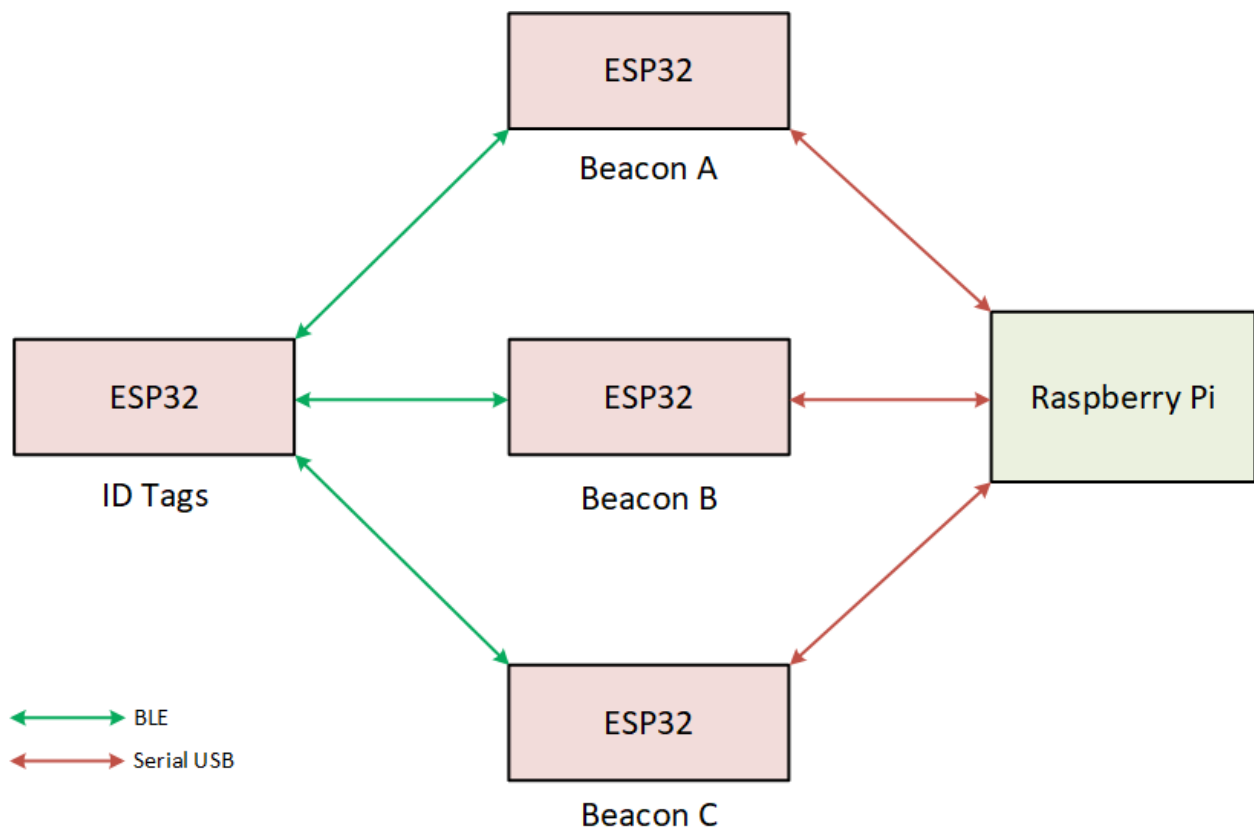


Figure 3: PoC System Block Diagram

## 2.2 Prototype

In the Prototype development phase the transceivers will be incorporated with Decawave DWM1000 UWB modules. The DWM1000 UWB uses radio frequencies in the range of 3.5 to 6.5GHz; this would significantly reduce issues of signal interference or multipath propagation which would occur by using RSSI with BLE. The DWM1000 will be incorporated as the transceiver with the ESP32 as the main MCU as shown in figure 4 below. RF data communication functions will be established between four UWB modules with one as the ID Tag and three as the Locator Beacons to demonstrate distance estimation with DWM1000 UWB modules. This will be achieved by using signal fingerprinting to determine transmitter properties such as ToF and unique Tag identifier. Furthermore, trilateration algorithms will be implemented on data processing unit to determine near real time location and coordinates of ID Tags. Initial Implementation of software stack on the data processing unit and development of GUI will occur during this phase as well.

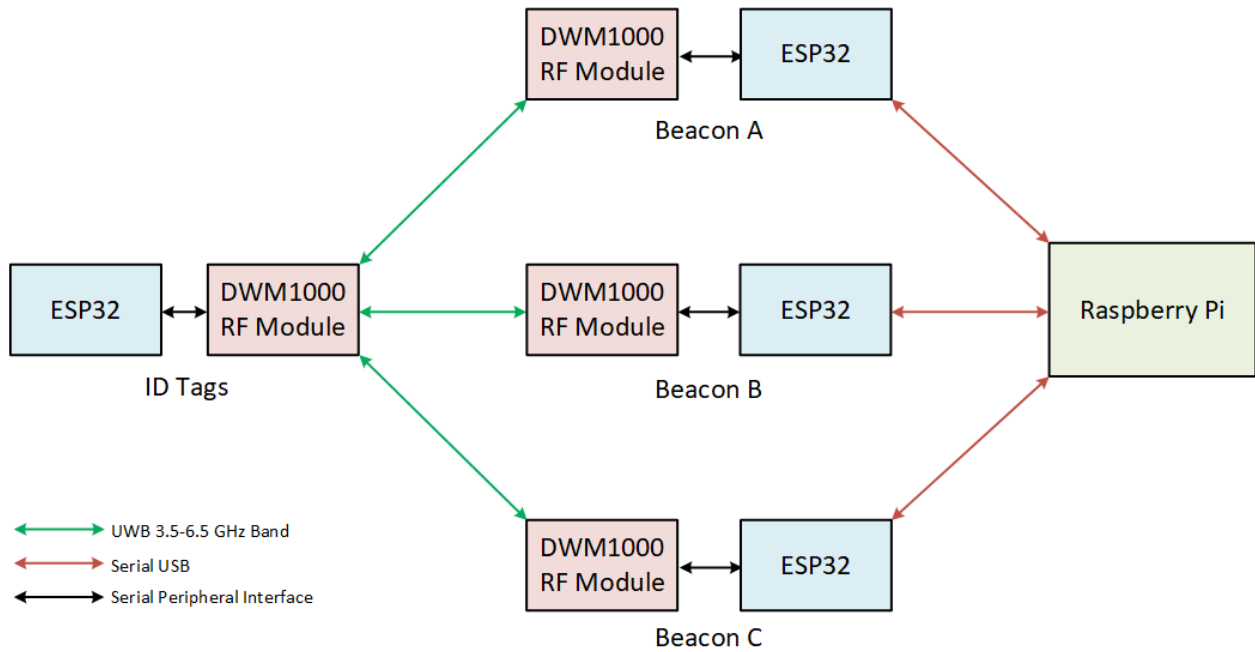


Figure 4: Prototype System Block Diagram

## 2.3 Final Product

The final product will demonstrate the fully functional indoor rescue system that detects the location of the ID tags and displays it accordingly on a GUI. Here the addition of ESP32's WiFi modules for Beacon to DPU communications can be seen (Figure 5), as the Beacon will communicate via WiFi communication with the data processing unit. The WiFi network will be a closed private network meaning that the network is only share between beacons and the data processing unit to ensure security, reliability and stability. Furthermore, implementation of a RF harvesting circuit for charging the ID Tag device during deep sleep mode will occur throughout this stage. All the components of the systems will be fully integrated as a close-to-production product. Component circuits and PCB footprint will be minimized and proper casing will be made to house all electronics. The data processing unit will provide the user with a full GUI to interact with along with fully implemented features such as importable blueprints and system configurations.

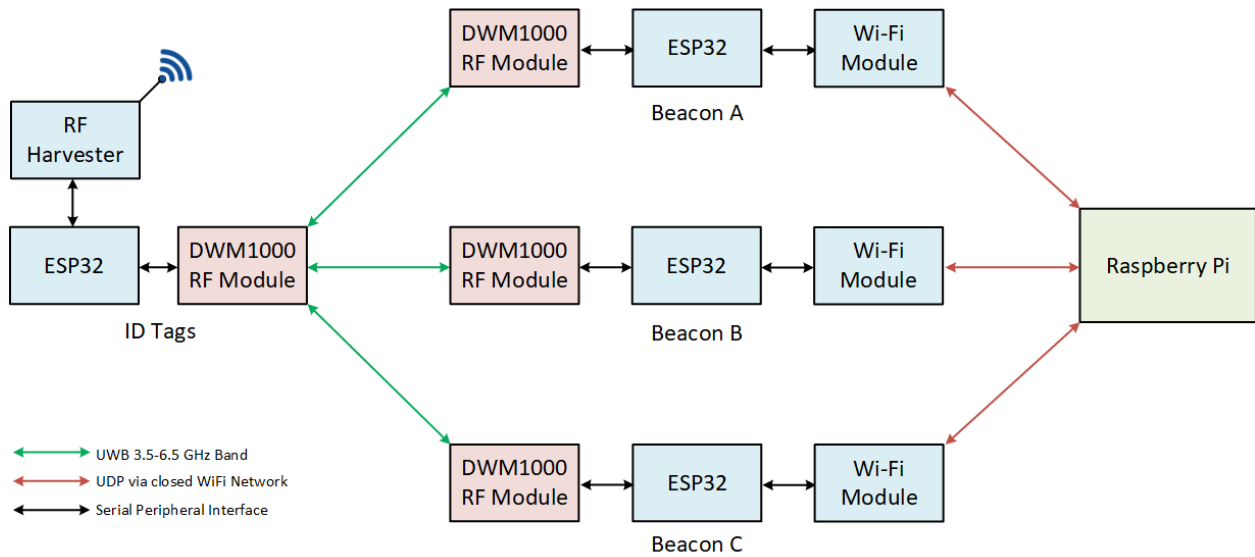


Figure 5: Final System Block Diagram



## 2.4 RF Fundamentals

Radio Frequencies (RF) is a common electromagnetic medium for communication systems to deliver digital information over the air from one point to another. Today, RF is used in many daily applications, such as FM radios and telecommunications. In fact, it is the discovery of RF signals that enables the deployment of many commercial wireless networks [6].

On the electromagnetic spectrum, RF spans from 30Hz to 300GHz in frequency. More specifically, Bluetooth ranges from 2.40 to 2.48GHz and UWB ranges from 3.1 to 10.6GHz. RF has two main attributes: amplitude and frequency [6]. The amplitude represents the strength of an RF signal. The measurement of amplitude is power which indicates the amount of energy required for a signal to propagate over a given distance. The amplitude of an RF signal is susceptible to degradation as it travels over a distance in air [6]. The rate at which an RF signal degrades follows the inverse square law shown in Figure 6, which states that generally signal strength decreases exponentially as distance increases. In theory, the degree of strength loss can be mitigated by having a more powerful signal to begin with provided that there is minimal obstacle against the signal on the way to its destination [7].

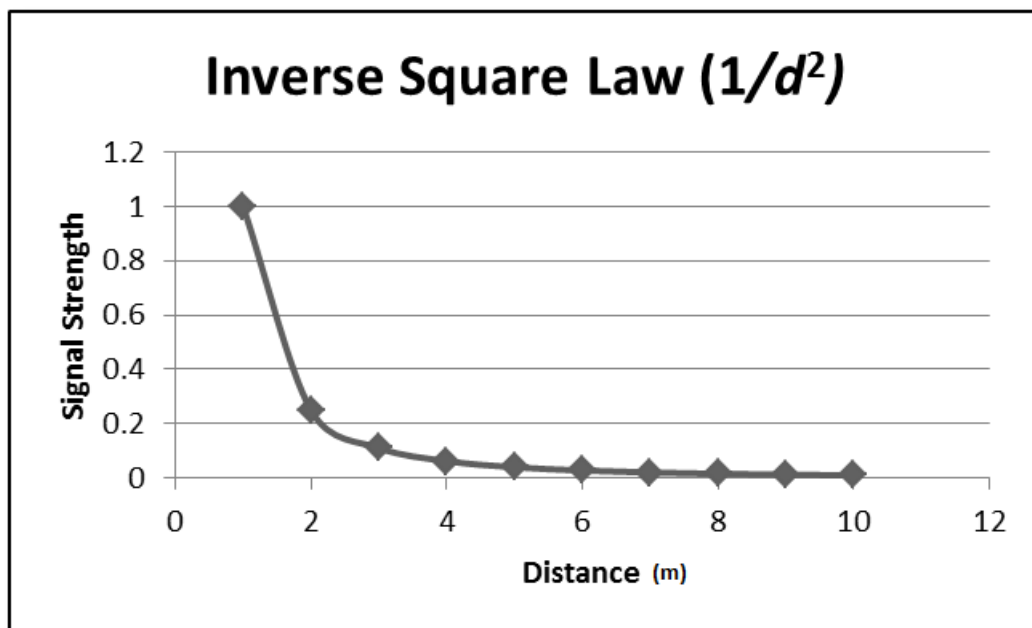


Figure 6: Relationship Between Signal Strength and Distance

Another factor of RF that influences distance propagation is signal frequency. In RF, frequency describes the number of signal repetition per second. RF frequency is measured in Hertz (Hz), which is the number of cycles occurring each second. For instance, an RF radio operating at 2.4GHz means that the signal includes 2,400,000,000 cycles per second [6]. The effects of different frequencies on distance propagation is illustrated in a Free Space Path Loss graph shown in Figure 7 below. The graph shows that there is positive correlation between signal strength degradation and frequency, where the higher the frequency, the more quickly signal-strength falls over distance [7]. In fact, this relationship is reasonable since the higher the frequency of a signal, the lower its wavelength.

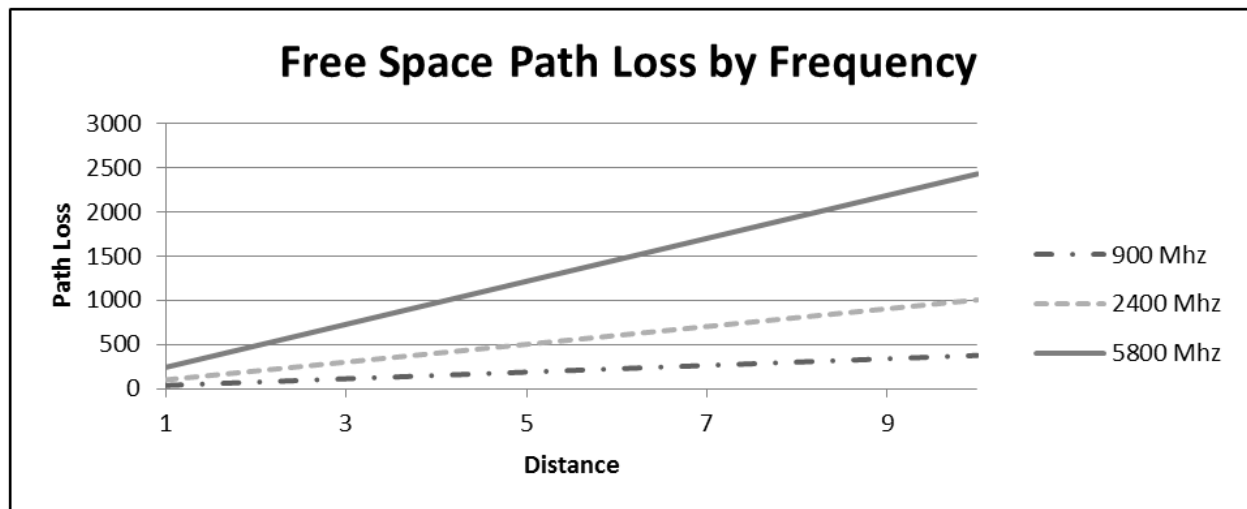


Figure 7: Effects of Different Frequencies on Distance Propagation

Besides RF signal attributes, environmental factors can also influence the integrity of an RF signal. Multipath is one of the most common and inevitable factor that can delay and distort a travelling RF signal. Multipath propagation is a phenomenon where an RF signal takes different paths when propagating from the source to the destination. As illustrated in Figure 8, contact with surfaces such as a desk or a ceiling contribute to the division of a signal into multiple signal portions to arrive at the destination at different times. This induces another phenomenon called multipath delay which causes the distortion of a signal's information. Since a signal is arriving as components, the receiver will not be able to interpret the information of the original signal during demodulation [6].

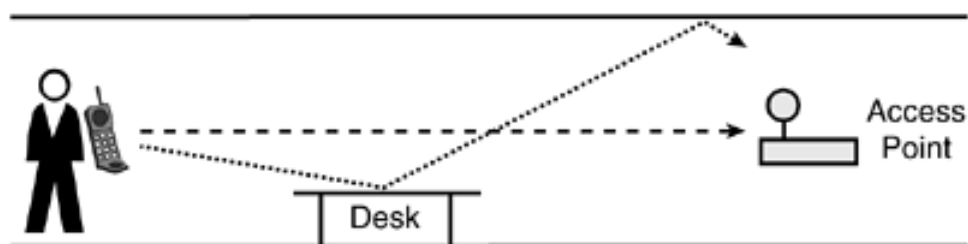


Figure 8: An Example of Multipath

## 2.5 Received Signal Strength Indicator

Received Signal Strength Indicator, or RSSI describes the amount of power detected by the receiver during the reception of a packet. It is a metric measured in dBm and can be acquired through BLE API of the ESP32 module. Signal strength is influenced by the inverse square law from the previous section, which states that signal strength decreases exponentially as the distance from the transmitter increases, hence it is possible to estimate the proximity of the receiver from the transmitter based on the changes in measured RSSI value. The distance conversion formula from RSSI to distance is shown below. In the PoC phase, Akriveia uses RSSI as the distance-related metric [8].

$$Distance = 10^{(Measured\_Power - RSSI)/(10 * N)}$$

Measured power is the expected RSSI at a distance of one meter. This constant value can be acquired by calibration through averaging sufficient number of RSSI values between transceivers placed at one meter apart [9].

N, is the environmental factor constant that ranges from values 2 to 4. It is a value set to adapt to the degree of indoor attenuation, interference and multipath [9].

## 2.6 Ultra-Wideband and Time-of-Flight

Time of flight, or ToF is a distance related metric that describes the propagation time of a radio signal between a sender and a receiver. Precisely, it is the time it takes for a radio signal to travel from a sender to receiver and back to the sender. Since radio waves are a type of electromagnetic radiation, the travel speed of a radio signal is the speed of light, which is approximately thirty centimeters per nanosecond. Given the speed of light constant and a ToF value, the distance between a sender and a receiver can be computed using the Distance Speed Time formula as shown below [8]. To measure ToF in the nanosecond scale, Akriveia will employ the Ultra-Wideband (UWB) technology by using the DWM1000 module in the prototype and final product phase.

$$Distance = Speed\ of\ light * ToF$$

In narrowband systems, measuring signal arrival times with accuracy is a difficult task without complex algorithms. Due to bandwidth limitations, radios in narrowband systems cannot resolve signals influenced by multipath and may assume all received subpath signals arrived at the same time [10]. Fortunately, multipath components can be resolved using UWB radios due to larger signal bandwidth. UWB radios are formerly known as pulse radios as they send signals in short bursts called chips. The chips are radio energy pulses that carry information over a large bandwidth to counter the effects of multipath fading and interference that are present in conventional narrowband signals such as 2.4G/5G WiFi. Due to the short duration of chips, the effects of multipath are also received as echoes that do not interfere with the original signal [8]. As a result, UWB provides a great solution to indoor localization by allowing high precision ToF measuring capabilities.

## 2.7 Trilateration Methods

The Akriveia 2-D indoor localization solution determines the x-y coordinates of ID tags by trilateration. The method follows a lateration scheme with absolute distances, which uses distance-related metrics such as RSSI or ToF to determine the distance between a sender and a receiver. In the case where there is a single beacon and ID tag, the distance between the two entities can be interpreted as the radius of a circle traced with the beacon centered as shown in figure 9.

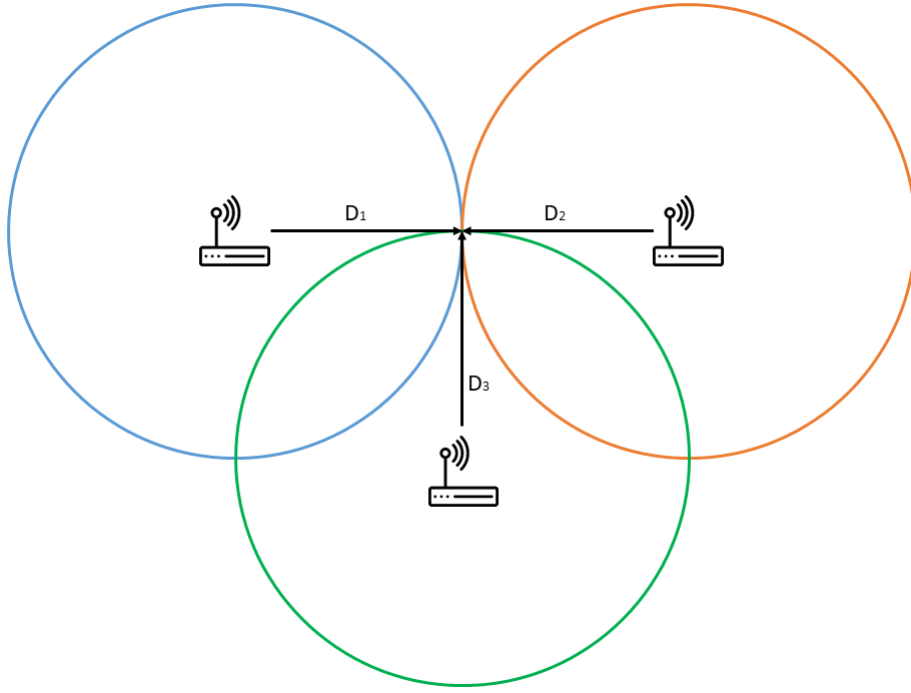


Figure 9: Trilateration Diagram

Such a circle takes on a standard mathematical form as shown in the equation below, where the variables  $x_0$  and  $y_0$  are the 2-D coordinates of the beacon position relative to its environment, and  $r_0$  is the distance between the beacon and the ID tag.

$$(x - x_0)^2 + (y - y_0)^2 = r_0^2$$

The same concept applies to the scenario with three beacons and an ID tag. Given the 2-D coordinates the three beacons and their individual distance with respect to the ID tag, three circles can be traced to form an intersection point at the location of the ID tag. Hence, three standard form circle equations are generated to form a system of equations with two unknowns as shown below. The solution to the unknowns, or the intersection coordinates of the ID tag can be obtained by solving such a system.

$$\begin{aligned} (x - x_1)^2 + (y - y_1)^2 &= r_1^2 \\ (x - x_2)^2 + (y - y_2)^2 &= r_2^2 \\ (x - x_3)^2 + (y - y_3)^2 &= r_3^2 \end{aligned}$$

The system of three equations can be ultimately simplified to the two equations shown below by expanding and equation elimination,

$$Ax + By = C$$

$$Dx + Ey = F$$

where the representation of constants A, B, C, D, E and F are as follows,

$$A = -2x_1 + 2x_2$$

$$B = -2y_1 + 2y_2$$

$$C = r_1^2 - r_2^2 - x_1^2 + x_2^2 - y_1^2 + y_2^2$$

$$D = -2x_2 + 2x_3$$

$$E = -2y_2 + 2y_3$$

$$F = r_2^2 - r_3^2 - x_2^2 + x_3^2 - y_2^2 + y_3^2$$

The system of two equations and two unknowns has the solution as shown below for the 2-D coordinates of an ID tag.

$$x = \frac{CE - FB}{EA - BD}$$

$$y = \frac{CD - AF}{BD - AE}$$

## 2.8 System Design Specification

The Akriveia Beacon consists of three main components: the Beacons, ID Tags, and the Data processing unit. The detailed specifications for each system will be outlined in their corresponding sections throughout the document. General system and performance specifications of Akriveia Beacon are listed in the table below.

|                      |   |
|----------------------|---|
| <b>REQ.SY.1 - C</b>  | The system must have two access modes: Emergency (for first responders), and Admin (for IT services)                                  |
| <b>REQ.SY.2 - P</b>  | Beacons and ID Tag must communicate with each other using BLE   |
| <b>REQ.SY.3 - P</b>  | Administrator must be able to access the system to perform health checks on the Beacons and ID tags                                   |
| <b>REQ.SY.4 - P</b>  | Administrator must be able to access the system to add/remove ID tags in the system database  |
| <b>REQ.SY.5 - P</b>  | The Beacons must use a WiFi mesh for forwarding data to DPU   |
| <b>REQ.SY.6 - P</b>  | The Beacons must contain a rolling buffer for ToF data for each id tag transmitted to   |
| <b>REQ.SY.7 - P</b>  | Each Beacons must use median of previous 10 samples of ToF data to sent to data processing unit                                       |
| <b>REQ.SY.8 - P</b>  | Beacons and ID Tag must communicate with each other using UWB (3.5-6.5GHz)  |
| <b>REQ.SY.9 - F</b>  | The System shall be easy to set up and tear down  |
| <b>REQ.SY.10 - F</b> | The System must remain operational during/after small scale disasters such a small fires and earthquakes with magnitude less than 6.9 |

Table 4: System Design Specification

### 3 System Components

#### 3.1 MCU - ESP32

ESP32 is a series of low-cost, low-power system on a chip microcontrollers with integrated Wi-Fi and dual-mode Bluetooth. Created and developed by Espressif, the ESP32 contains a Tensilica Xtensa LX6 microprocessor in both dual-core and single-core variations and includes a in-built antenna, power amplifier, low-noise receive amplifier, filters, and power-management modules (As shown in figure 10) [11].

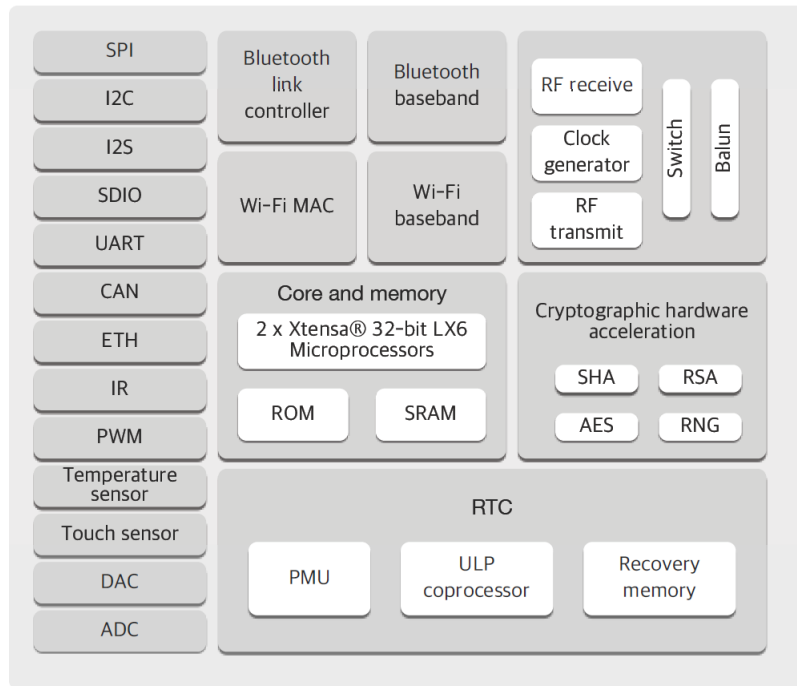


Figure 10: ESP32 Architect Block Diagram

For the proof of concept, the Bluetooth modules on the ESP32 will be used as the communication interface between the Beacon and the ID tags. Using BLE advertising, the ID Tags ESP32 broadcasts a unique MAC address. The Beacon ESP32 can collect all associating ID Tag ESP32s and capture the RSSI and MAC of these devices. These information are forwarded to the DPU for distances estimation and location plotting.



Figure 11: ESP32 Development Board



In the Final design both the Beacon and ID tags will use the ESP32 as the main controller unit but they will be also be incorporated with Decawave DWM1000 UWB modules as the transceivers. The ESP32's Serial Peripheral Interface (SPI) will be used to interact with the Decawave DWM1000 UWB modules for sending and receiving data. A mock-up circuit diagram presenting the SPI interface between the ESP32 and a breakout board for DWM1000 is shown in figure 12. In addition, the ESP32 features a deep sleep mode which reduces power consumption to about 10uA from 260mA (active operations), thus drastically increasing the battery life time which is an important constraint for the portable ID tags. In the event of an emergency, the ESP32 can be woken from deep sleep to start transmission by activating a touch pin (GPIO 26-23 shown in Figure 13). Furthermore, by utilizing the ESP32's wide range of capabilities such as WiFi or Bluetooth, a mesh network can be implemented to extend the communication range of the Beacons.

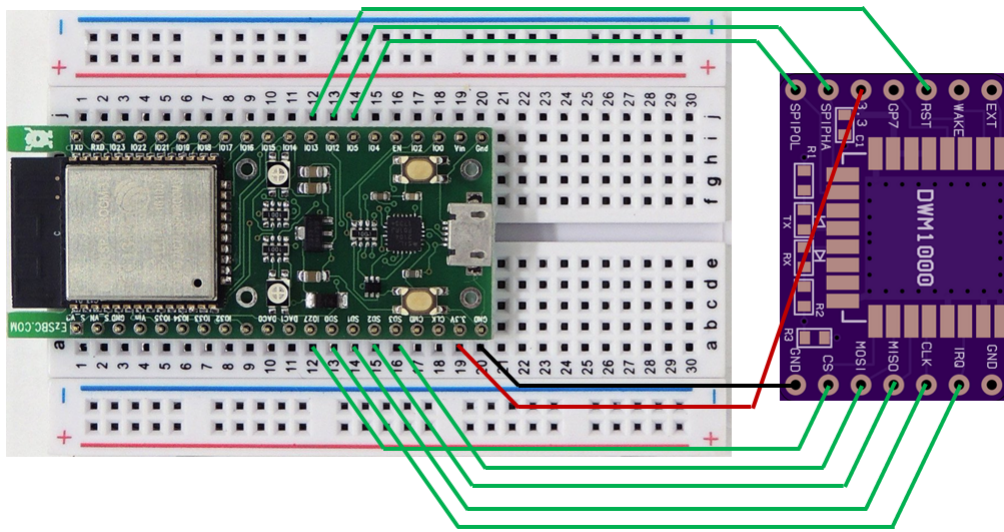


Figure 12: Circuit Diagram of ESP32 & DWM1000

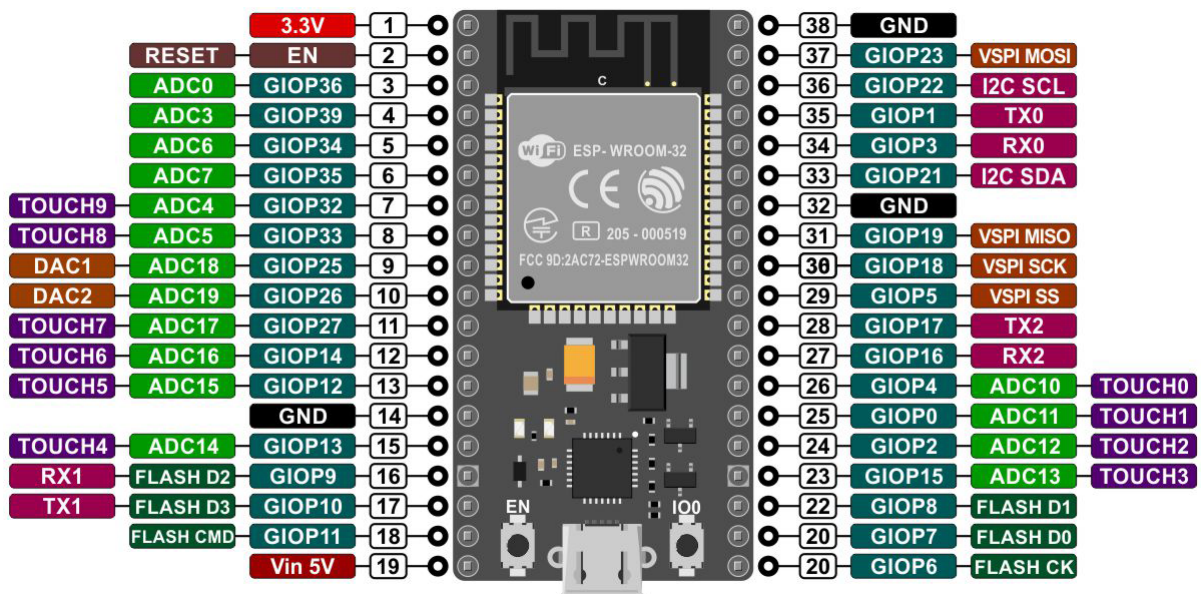


Figure 13: ESP32 Pin Layout



### 3.2 Transceiver - DWM1000

The Beacon and ID Tag communication will be done using ultra-wideband wireless communication in the prototype phase and beyond. The most optimal transceiver available on the market that best fits the needs and scope of this project is the Decawave DWM1000 UWB module (Figure 14). The DWM1000 is an IEEE 802.15.4-2011 UWB compliant and FCC/ETSI certified wireless transceiver module based on Decawave's DW1000 IC [12]. This module is a combination of DW1000 IC, a built in antenna, power management system, and clock control which allows for simple integrations into any system (Figure 15).

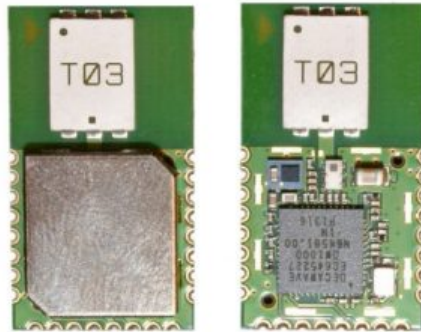


Figure 14: Decawave DWM1000 Modules

The module enables the location tracking of objects in real time location systems (RTLS) down to a precision of 10 cm indoors. It supports high range of communications data rates from 110 Kbps to 6.8 Mbps, with excellent communication ranges of up to 300m. The frequencies of operation in the range of 3.5 to 6.5GHz with seven distinct channels which would significantly reduce issues of signal interference or multipath propagation. Its small physical size allows the module to be implemented in highly cost-effective solutions. By using this module, integration with the Akiveia Beacon system is intuitive and simple; since the DWM1000 also offers a wide range of MCU support such as Arduino MCUs or the ESP32 MCUs.

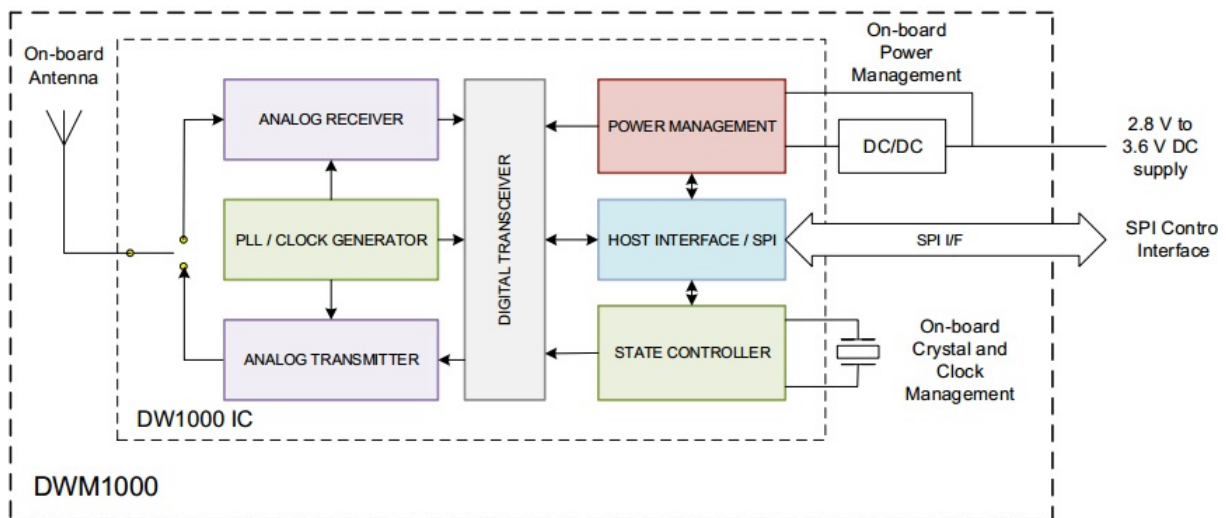


Figure 15: DWM1000 Internal Block Diagram

### 3.3 DPU - Raspberry Pi

The Data Processing Unit is a stand alone single board computer (SBC). For the demonstration of this project a Raspberry Pi 3 B+ is used as the DPU since it is an affordable and robust SBC, but the DPU in theory should be any electrical computer device that is capable of running a basic linux operating system; as the software stack is designed to operate on any linux based system. The Raspberry Pi product is cheap, portable and designed with an Cortex-A53 processor it meets the minimum requirements for the DPU.

The Raspberry Pi 3 B+ is a single board computer with a 1.4GHz 64-bit quad-core processor, dual-band wireless LAN, and Bluetooth 4.2/BLE [13]. The Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC at 1.4GHz quad-core processor allows complex tri or multilateration calculations done simultaneously for multiple ID Tags. With dual-band 2.4G and 5G IEEE 802.11.b/g/n/ac wireless LAN, the Pi can create a reliable network access for multiple ESP32 for data forwarding. At a power rating of 5V/2.5A DC the Pi requires minimal power to operate, which allows for a variety of power options during emergency disasters. Furthermore, the Pi is configurable with any linux based OS, such as Debian OS which is compatible with the software stack developed in Rust that will create a layer of user interface between the user and the Akriveia Beacon system.

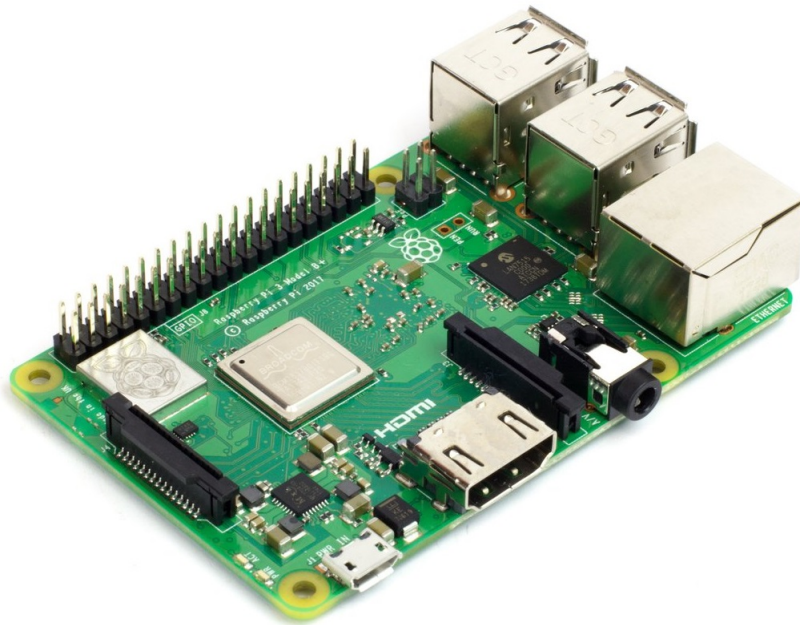


Figure 16: Raspberry Pi 3 B+ Model

## 4 Hardware Design

### 4.1 System Operation Modes

The Akrievia Beacon system will have two modes of operations, idle mode and emergency mode. Idle mode occurs during every day operation that is not under an emergency disaster. An emergency disaster is defined as a situation that poses an immediate risk to health, life, property, or environment. Most emergency disasters require urgent intervention to prevent a worsening of the situation. In emergency disaster situations the system will be triggered on to operate under the emergency mode following the system state diagram as shown in figure 17.

When system is under idle mode, the beacon system does not attempt to transmit location information from ID Tags to the DPU, as the ID Tags will be under deep sleep mode with the transceiver powered off. In idle mode the system is available for configurations such as adding, editing, and deleting ID Tags, beacons and blueprints. In the Event of an emergency, a trigger such as one that could be associated with a fire alarm will trigger the Akrievia Beacon system state into the emergency mode.

When system is under emergency mode, the beacons will attempt to establish a data pipeline using the UWB modules. And the ID Tags are triggered on by personnel carrying the device that are in need of rescue or assistance. Under emergency mode the DPU triggers the Beacons to start sending and receiving packets to determine ToF data from the ID Tags. The ID tags will receive request packet from beacon and transmit location data back to the beacon. Once the emergency situation is resolved the system can be switched back into idle mode by privileged administrators.

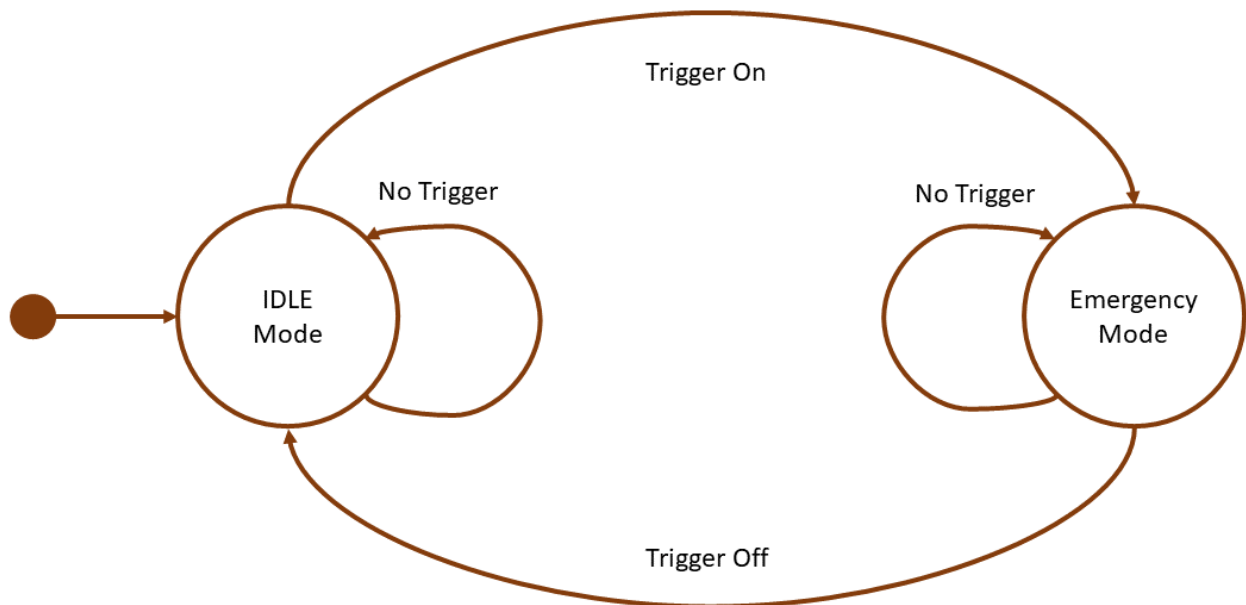


Figure 17: Akrievia Beacon System State Diagram

## 4.2 Communication Protocol

### 4.2.1 Beacon to ID Tag Communication

Communication between beacon and ID tag is facilitated on layer 1 the Physical (PHY) layer of the Open Systems Interconnection model (OSI Model model). The Physical layer is responsible for converting data into bits for transmission and converting received bits back into data. The logical connection formed between the two entities are the radio channels that are present in air. In the proof of concept phase, 2MHz wide BLE advertising channels 37, 38 and 39 with center frequencies at 2402MHz, 2426MHz and 2480MHz respectively can be used [14]. In the engineering prototype and final product phase, 500MHz wide UWB channels 1, 2, 3 and 5 with center frequencies at 3494MHz, 3993MHz, 4492MHz and 6489MHz respectively can be used [15]. The difference in frequency and power spectral density can be observed by the plot below (figure 18). The radio channels will be used at full duplex to allow simultaneous transmission in both sending and receiving directions. Digital data such as RSSI measurements, ToF measurements and MAC address are encoded into analog signals by means of modulation to represent the data with continuously varying electromagnetic waves. Modulation in both BLE and UWB form pulses of analog signals prior to transmission [14]. The process of sending and receiving these analog signals are handled by the integrated chip antenna of the ESP32 and DWM1000 modules. Upon reception of the analog signals, the receiver's demodulator and decoder will reverse the work of the sender to retrieve the digital data.

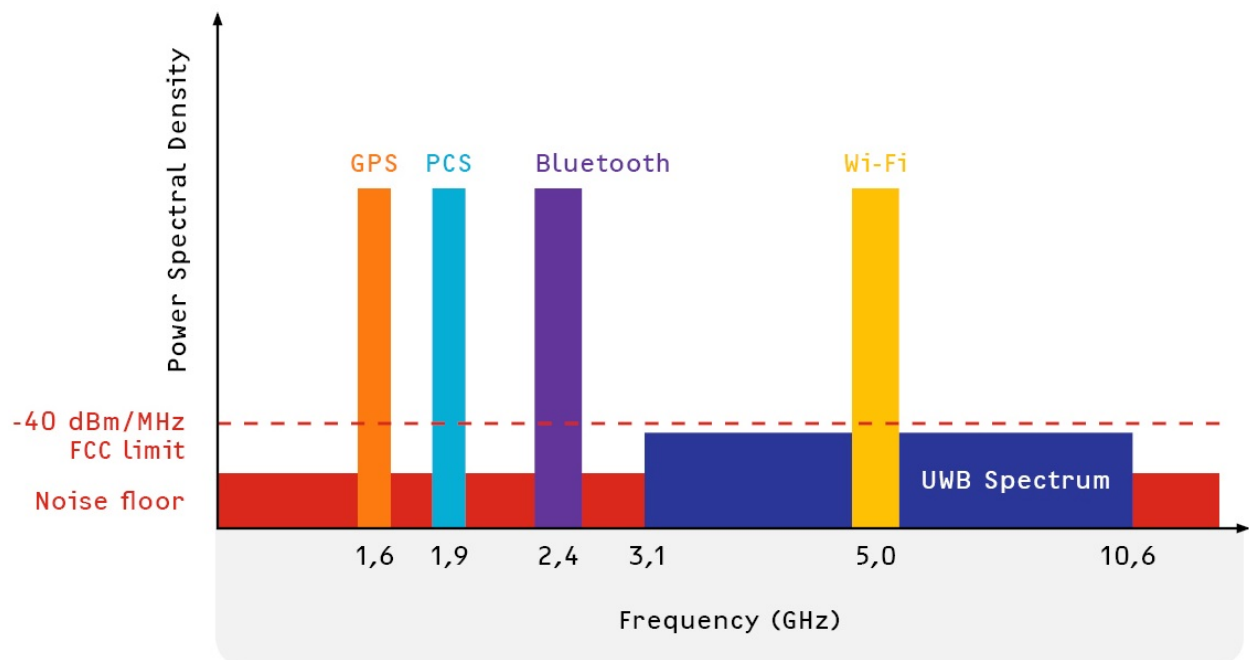


Figure 18: Common Signal Power spectral Density VS Frequency

#### 4.2.2 Beacon to DPU Communication

For the proof of concept and prototype, data-pipeline is a simple implementation with the serial interface. The Beacons will receive and send data with the data processing unit over the USB serial read and write interface. In the Final design the Beacons will use the WiFi module on the ESP32 to join a privately hosted access point created by a hostapd on the data processing unit (Raspberry Pi). Hostapd (Host access point daemon) is a user space software access point capable of turning normal network interface cards into access points and authentication servers. The current version supports Linux (Host AP, madwifi, mac80211-based drivers) and FreeBSD (net80211) [16]. Once each beacon is connected to the access point they will be assigned an IP by the Dynamic Host Configuration Protocol (DHCP) server. A one-to-many network will be created as shown in figure 2. Using User Datagram Protocol (UDP) communication over the networking layer (layer 2-3 on OSI model), the DPU can bind on the gateway IP and a specific port to listen for forwarded beacon data. To initiate control with the beacons, the DPU will bind to each beacon IP at a specified port and send commands via UDP.

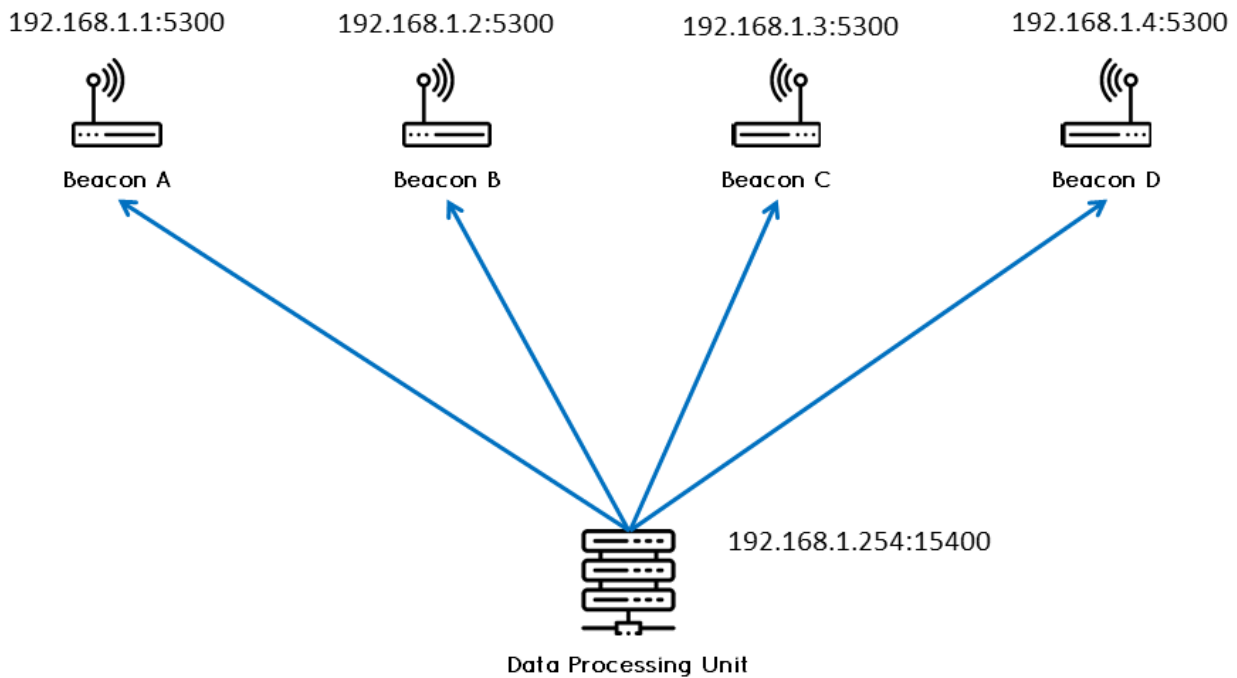


Figure 19: UDP Communication Network

### 4.3 Beacon Design

To control the beacons the DPU will send a command to the beacon and the ESP32 will evaluate the command and execute accordingly as shown in the beacon flow diagram in figure 20. When system is under emergency mode, the DPU initiates a start command to the Beacons. Once the Beacons acknowledges the start request, it attempts to establish data pipelines to each of the ID Tags in range of the UWB transceiver. After a successful connection, ToF data between beacon and ID tag will be collected and forwarded to the data processing unit for trilateration calculation and error processing before displaying to the user via GUI. On the DPU data of each id tag encountered will be stored in a hash table entry, with the MAC address of each ID tag as the key. The distance measurement are store a rolling buffer for each id tag so that data points can be averaged, reducing estimation error.

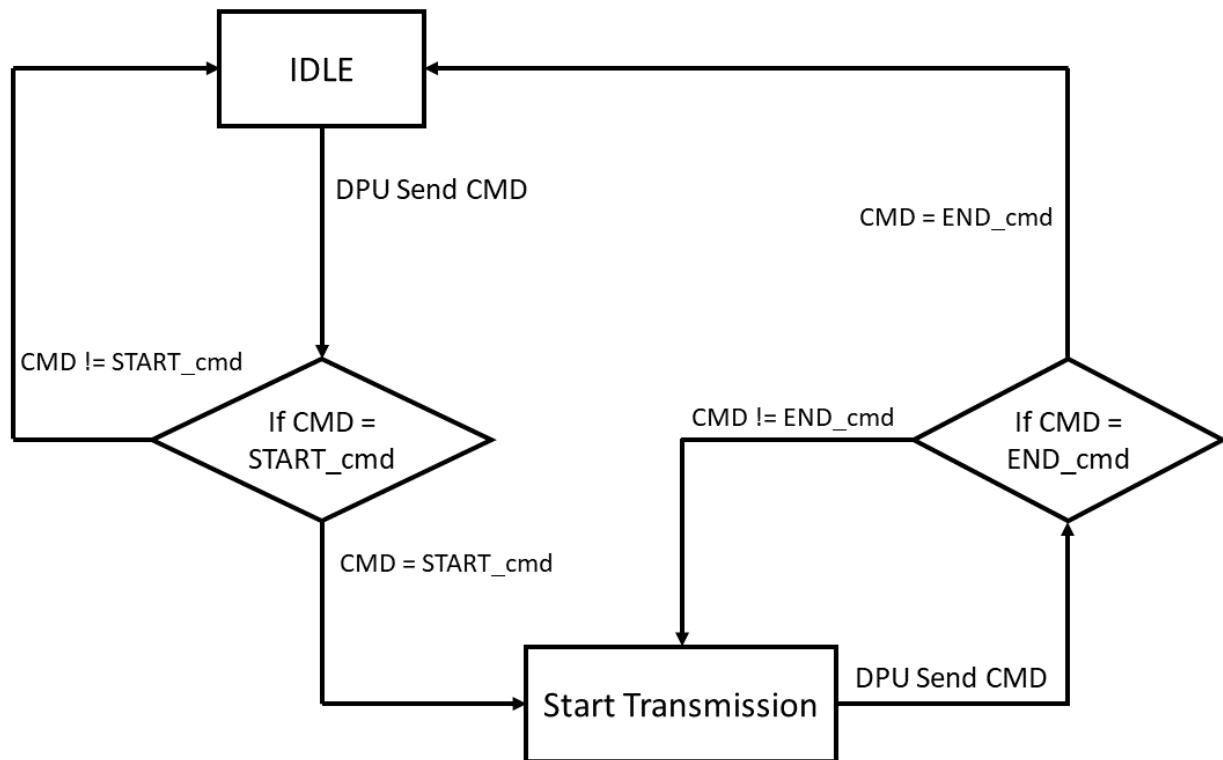


Figure 20: Beacon System Flow Diagram

A 3D appearance mock-ups of the ID Tag was done in Solidworks and can be seen in the Computer Aided Design (CAD) representation, figure 21. The Beacon units, which will relay location data from the ID Tags to the Data Processing Unit (DPU), consist of an ESP32 module, a DWM1000 UWB transceiver, a 9V lithium ion battery, and a power cable. These components of the device will be contained in an encasing made from PLA plastic. Each beacons will have LEDs indicating the power and transmission state, and a reset button for resetting the device state.

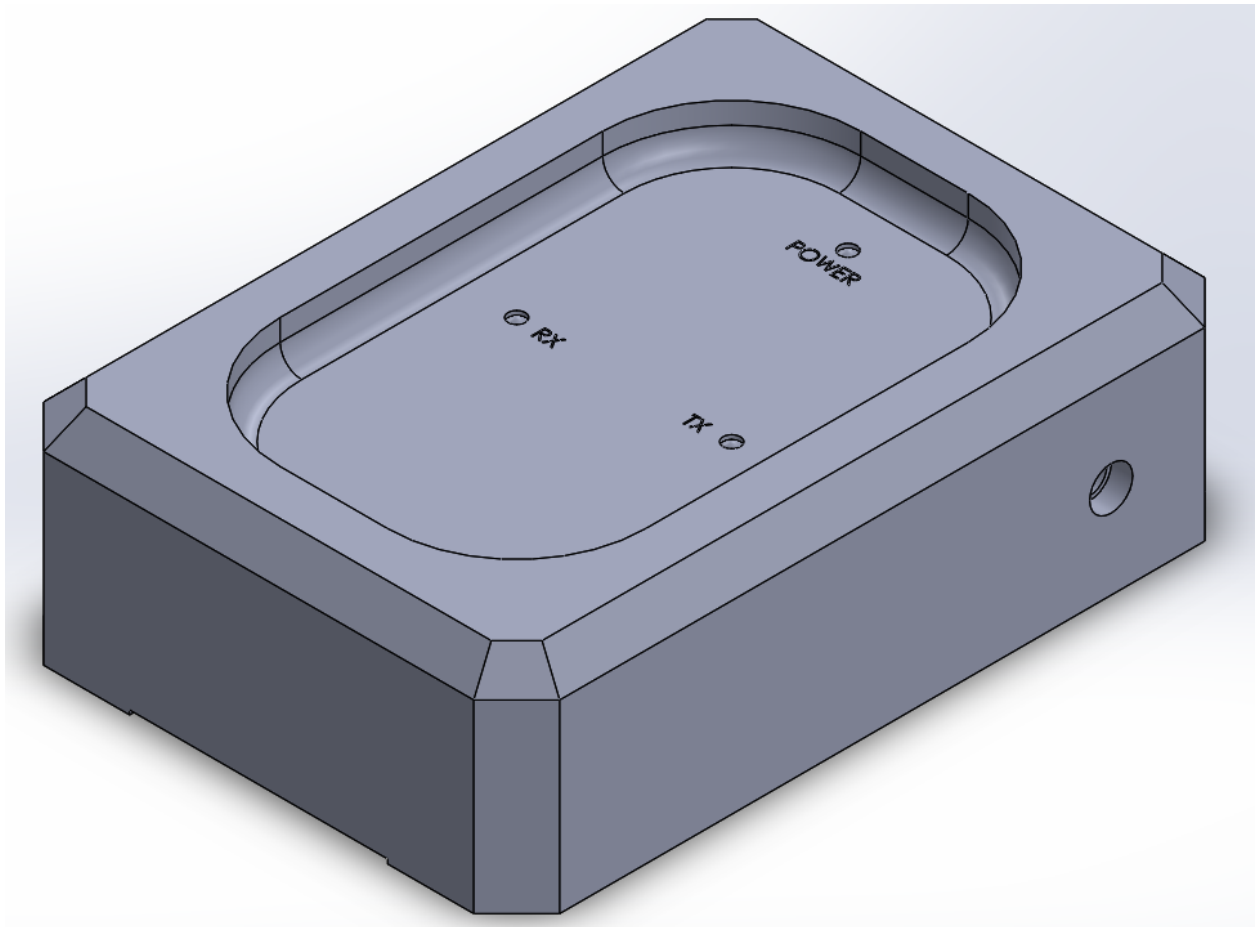


Figure 21: CAD representation of Beacon

#### 4.4 ID Tag Design

The ESP32 is used in the ID tag as it contains a function called deep sleep mode, in which the ID Tag has minimum power consumption and does not transmit data for location tracking. During idle mode, the ID tags are operating under deep sleep mode for power conservation drawing only 10uA of current. Therefore, real time tracking outside of emergency mode is not possible as insufficient power is provided to the DWM1000 modules. The ID Tag will be activated in an emergency situation via capacitive touch button located on the device. As user wake the ID tags by triggering the touch button the ID tags will attempt to connect to the Beacons. If the system is not in an emergency state, the ID Tag will fail to establish a connection and briefly show the charge level before returning to deep sleep mode. If system is under the emergency mode, the MCU on ID tags will be able to establish a data pipeline with the beacons on specified UWB channel, and the ID tag will then transmit an acknowledge packet back to the beacon to produce location data as described in the system flow diagram in figure 22. Only during active emergency mode will the beacons be requesting for establishment of data pipeline and to initiate DWM1000 Ranging state.

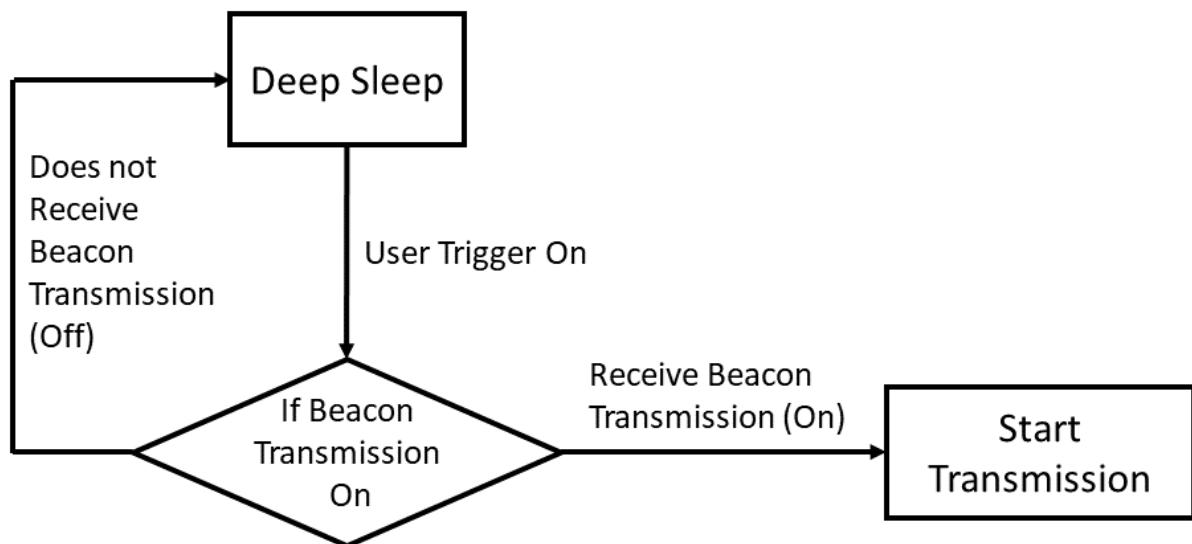


Figure 22: ID Tag System Flow Diagram



A 3D appearance mock-ups of the ID Tag was done in Solidworks and can be seen in the CAD representation, figure 23. The ID Tags, the part of the system being tracked by the Beacons, will be powered by a rechargeable 4-5 V battery with minimal 3000mAH, which will be charged over time by a RF harvester (see sec. 5.2). The ID Tag will consist of an ESP32 MCU module, and a DWM1000 UWB transceiver chip. The internal components of the ID Tag will be contained in a PLA plastic shell with LEDs indicating the power state and transmission activity of the ID Tag.

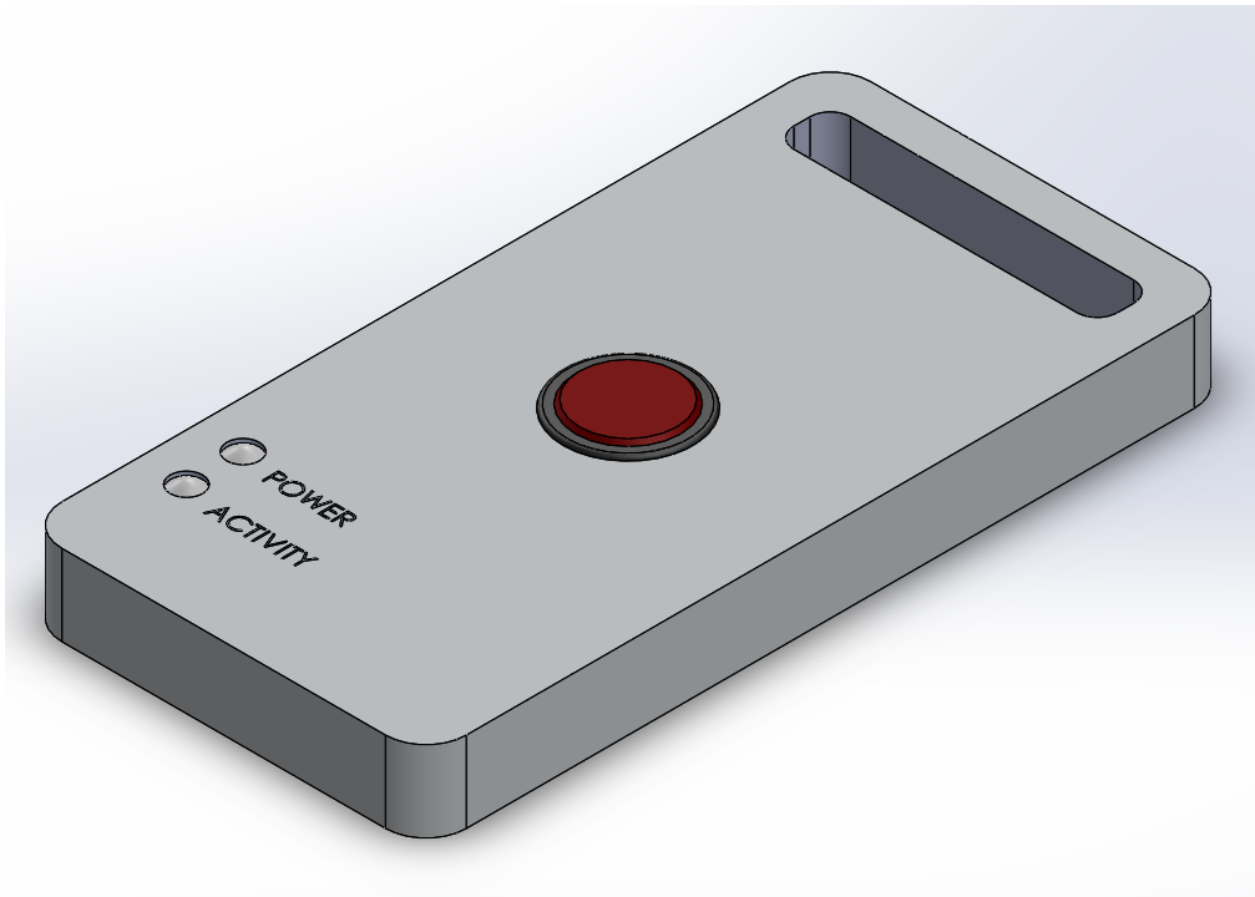


Figure 23: CAD representation of ID Tag

## 4.5 Hardware Design Specification

TRIWAVE SYSTEMS has chosen to develop the proof-of-concept prototype using 2.4GHz Bluetooth radio modules for initial testing of the overall system feasibility and the accuracy of the trilateration method. The system will then be improved upon by incorporating Decawave DWM1000 Ultra-Wideband (UWB) modules for further prototyping. ESP32 Micro-controller Units (MCUs) will be used to control the beacons and ID tags and a Raspberry Pi 3 B+ will be used as a data processing unit. The requirements below detail the requirement for hardware functionality of the Akrivia Beacon system.

|                     |  |
|---------------------|--|
| <b>REQ.HW.1 - C</b> | The Beacons must use ESP32 as the microcontroller unit and transceiver |
| <b>REQ.HW.2 - C</b> | The ID Tag must use ESP32 as the microcontroller unit and transceiver  |
| <b>REQ.HW.3 - P</b> | ID tag broadcast duration must be at least 1 hour long upon activation |
| <b>REQ.HW.4 - P</b> | ID tag must return to deep sleep mode after broadcasting period        |
| <b>REQ.HW.5 - P</b> | The Beacons must use Decawave DWM1000 UWB modules as transceivers      |
| <b>REQ.HW.6 - P</b> | The Beacons must use ESP32 as the controller units                     |
| <b>REQ.HW.7 - P</b> | The ID tags must use Decawave DWM1000 UWB modules as transceivers      |
| <b>REQ.HW.8 - P</b> | The ID tags must use ESP32 as the controller units                     |

Table 5: Hardware Design Specification

## 5 Electrical Design

### 5.1 Power Management

Developing a product that operates in emergency and disaster situations means that each component will have to operate in an extremely wide range of conditions. For this reason, each component of the system will include a backup power supply. Since size is not as much of a constraint for the beacons and data processing unit components of the system, it is easier to be more liberal with the size of the backup battery. The beacons and data processing unit is allowed an additional 9V 4000mAH rechargeable lithium ion battery backup.

Power consumption will be an important aspect to focus on because the ID Tags are a wearable electronic device, therefore, the device battery has to maintain charge over very long periods of time. One of the restrictions that must be considered is related to the size of the ID Tags; if an employee must wear their ID tag over long periods of time, as would be expected by users of this system, a large and heavy ID Tag is not an option. During prototype phase of the development stage, a 5V rechargeable lithium ion battery at 4000mAh will provide sufficient in power delivery and size for the ID tags.

The ESP32 chip is a Dual-Core 32-bit microprocessor along with 448 KB of ROM, 520 KB of SRAM and 4MB of Flash memory. It also contains WiFi module, Bluetooth Module, Cryptographic Accelerator (a co-processor designed specifically to perform cryptographic operations), the RTC module, and other peripherals as shown in figure 24 [17]. During normal active mode of operation where the ESP32 is sending and receiving data, the power consumption requires between 160-260mA. Under the assumption that the ESP32 is on and transmitting at all times and with a 5V, 4000mAh battery, the ID Tag device can achieve a lifetime of only 15 hours as calculated in the equation below.

$$\text{BatteryLifeTime} = 4000\text{mAh} / 260\text{mA} = 15.38\text{Hours} \quad (1)$$

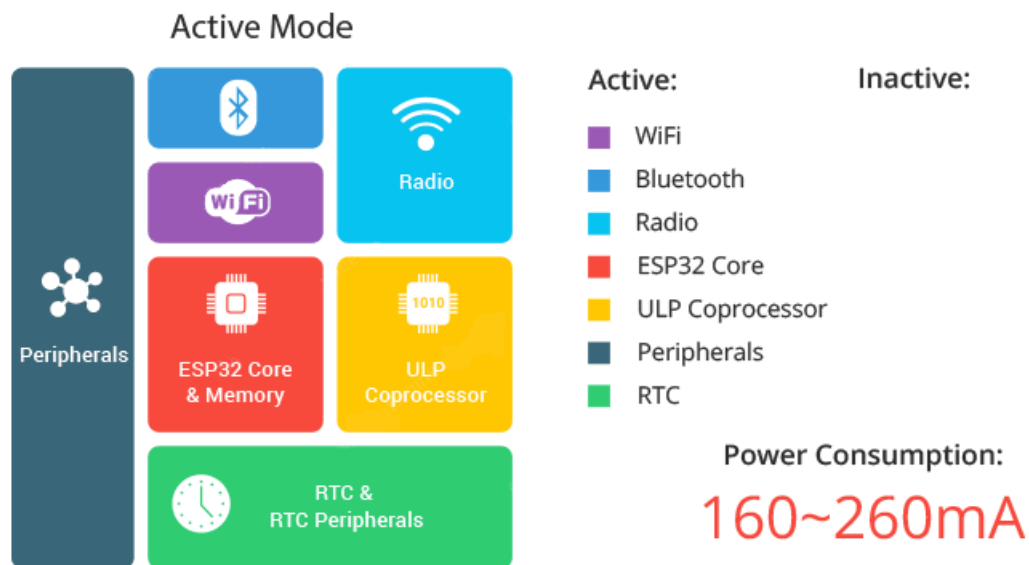


Figure 24: ESP32 Active Mode Power Usage

To optimize the battery life time when not under emergency mode, the ESP32 has an advanced power saving mode called Deep Sleep mode. Power consumption for ID tags can be cut back by utilizing ESP32's Deep Sleep Modes and dramatically extending the battery life. In deep sleep mode, the CPU, most of the RAM and all the digital peripherals are powered off. The only parts of the chip that remains powered on are: RTC controller, RTC peripherals (including ULP co-processor), and RTC memories (slow and fast) as shown in figure 25 [17]. The chip consumes around  $10\mu\text{A}$  under deep sleep mode and using calculation shown in equation (2) the battery life time can be extended up to 400,000 hours. In the event of an emergency disaster, the ESP32 can then transition from deep sleep mode to active mode using the ESP32 Wake-up source by touch configuration. The ESP32 consists of touch GPIOs that can be used to trigger wake up by an interrupt at the RTC module during deep sleep mode. The touch button sensor of the ID tags are connected to a touch GPIO on the chip to enable the feature.

$$\text{BatteryLifeTime} = 4000\text{mAh} / 10\mu\text{A} = 400,000\text{Hours} \quad (2)$$

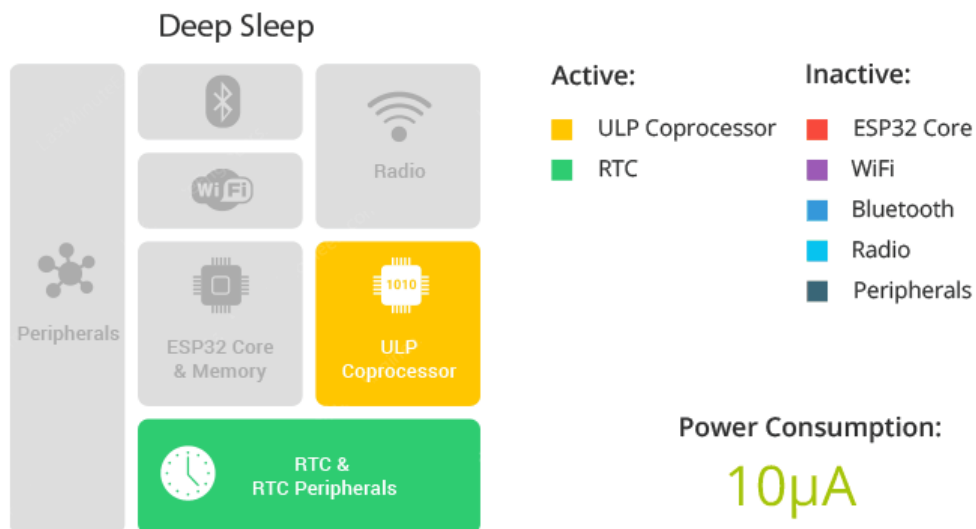


Figure 25: ESP32 Deep Sleep Mode Power Usage

## 5.2 RF Harvester

Radio Frequency is an abundant source for energy harvesting especially in a radio wave rich environment. When Radio Waves reach an antenna it causes a changing potential difference across the antenna. The potential difference causes charge carriers to move along the length of the antenna in an attempt to equalize the field, and the RF-to-DC integrated circuit (Figure 26) is able to capture energy from the movement of those charge carriers. The energy is stored temporarily in a capacitor and then used to create a desired potential difference at the load [18].

Ambient Radio Waves

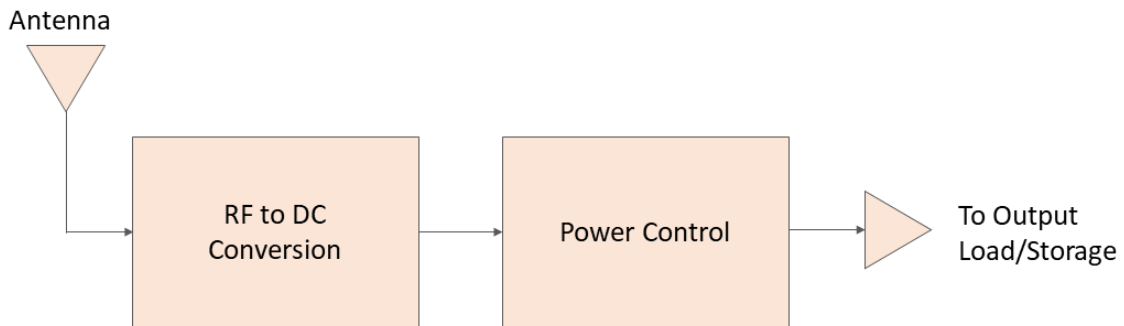


Figure 26: RF Harvester Block Diagram

There will be a demonstrable RF harvester circuit similar to Figure 27 that would convert ambient radio signal to DC voltage to charge the ID tag under deep sleep mode. During initial testing, the harvesting was able to collect up to 100 mV, however the ESP32 deep sleep mode requires 10  $\mu$ A. Load calculation and capacitance optimization will need to be performed to further improve the reliability of the harvester. Once RF harvester test circuit produces adequate results in the prototype phase, the circuit will be implemented onto the ID Tag devices as PCB components. The RF harvester circuit designed for charging the ID tags will be collecting ambient RF power created from ambient WiFi and cellular signals to maintain charge for the ESP32 during deep sleep mode.

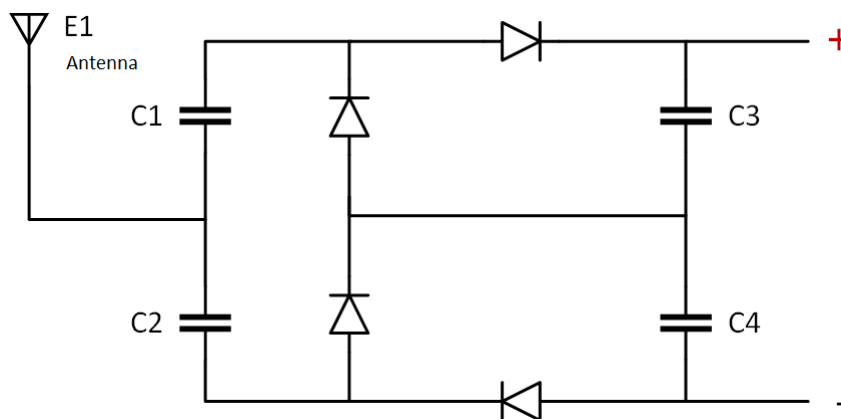


Figure 27: RF Harvester Circuit Diagram

### 5.3 Electrical Design Specification

The Akriveia Beacon system is designed to operate under emergency disaster situations, it is crucial that sufficient power is provided to each device at any given time. In order for the system to be reliable and, the electrical systems must be robust and efficient. TRIWAVE SYSTEMS has compiled a strict set of electrical requirements that ensures the beacons and ID tags will operate in a safe and efficient way.

|                     |  |
|---------------------|--|
| <b>REQ.EC.1 - C</b> | Each beacon shall be powered through standard North American power outlets (120V AC, 60Hz, type A/B) |
| <b>REQ.EC.2 - C</b> | The Data Processing Unit (DPU) will be powered over USB by the device that it is plugged in to       |
| <b>REQ.EC.3 - P</b> | The ID tags must have a manual switch to toggle power on   |
| <b>REQ.EC.4 - P</b> | The ID Tags must be powered by 5V rechargeable lithium ion battery                                   |
| <b>REQ.EC.5 - P</b> | The ID tags must be under deep sleep mode drawing no more than 10 uA when not in emergency mode      |
| <b>REQ.EC.6 - F</b> | The beacons must use rechargeable 9V lithium battery as a backup power supply                        |

Table 6: Electrical Design Specification

## 6 Software Design

### 6.1 Software Overview

The Akriveia system is comprised of 3 different devices, namely: the data processor, beacons, and ID tags; this device diversity plays a large role in the software choice for the beacons and ID tags. The Akriveia data processor is the primary access point for Administrators and First Responders, of whom will be presented a Graphical User Interface served by the data processor in addition to handling commands from the client. The software on the Data Processing Unit(DPU) is written entirely in Rust, including the static client side webpage. The requirements for each device are different, however the ID tags and beacons are able to share a similar environment; as such the Akriveia Beacon System will require two different software environments, one for the DPU and one for the ID tags and beacons.

### 6.2 Software Stack

Akrivia Beacon is composed of two primary languages, the DPU is implemented entirely in Rust while the beacons and ID tags use the Arduino language.

#### 6.2.1 Software Environments

The data processing unit requires an operating system (OS) to perform CPU scheduling for the multi-threaded application, WiFi drivers, and a proper file system to handle a database. As such, Linux or more specifically the Debian distribution of Linux was chosen as the DPU operating system.

Debian is widely praised for its stability, which makes it ideal for a data processing as the Akriveia Beacon System is designed to operate under emergency situations. Debian brings in the Aptitude package manager for dependency management, which allows for quick installation of required driver updates, operating system updates, and any additional packages required. Debian's Aptitude has the advantage compared to other package managers because it removes the need to worry about dependencies, manual installation, and even installation of sub dependencies. As Debian is a Linux distribution, it has widespread hardware adoption across many different hardware architectures - primarily X86-64 and ARM64 - giving Akriveia the flexibility to run on the most basic single board computers.

Linux has many different flavors, such as the Arch Linux which uses a different package management paradigm called Rolling Release, favouring earlier adoption of newer packages to more quickly introduce features at the cost of stability. Rolling Release are unsuitable for production servers due to their instability. Another alternative to Debian is Windows, however Windows suffers from high power usage, poor package management, and heavy reliance on proprietary software, whereas Linux is open source, free of cost, and has a more inclusive licence.

### 6.2.2 Software Languages

The DPU is written in Rust. Rust is a relatively new language as it was created in 2006 [19] and its first stable version was released in 2015 [20]. Taking inspiration from C++ and Haskell in its design, Rust is a systems language designed for stability and robustness. Rust eliminates entire classes of errors using the borrow checking system, a feature unique to this language. The borrow checking system defines a strict set of rules to follow, and fails to compile if these rules are broken - even if the application is otherwise logically and syntactically correct. As a systems language, Rust is compiled to binary rather than executed as a script through an interpreter, however, this does not stop it from being a higher order language. Rust does not expect developers to manage allocation and deallocation of memory directly, nor does it have a runtime Garbage Collector; rust relies on the Borrow Checker and lifetime system at compile time to statically analyze memory usage and automatically places the allocations and deallocations as needed. The benefits of the Borrow Checker do come at a cost, resulting in slow compile time and high learning curve. The features of Rust will be used throughout the DPU, as the backend webserver, the data processor, as well as the on the browser in the form of webassembly.

The beacons and ID tags are written in Arduino language. The Arduino language is merely a set of C/C++ functions and is specifically designed for embedded programming, allowing use of higher level features when desired while also giving the developer absolute control over how memory is managed. The Arduino environment performs a few transformations to the Arduino sketch before passing it to the avr-gcc compiler. On Arduinos, memory is very limited so control is key to fit the program within the small space constraints.

### 6.2.3 Software Standards

Rust code will follow the the Rustfmt standard of source layouts, Rustfmt is a tool provided by the Rust project that automatically formats all source in the standard format. The Rust compiler also provides a few warnings for common formatting issues builtin, which will be heeded. Arduino language will follow the ISO C/C++ programming standards [21]. The Arduino style guide [22] presented are more ad hoc than a full style guide.

### 6.2.4 Frameworks

Akriveia Beacons' main DPU framework is called Actix. Actix is an Actor Framework that follows the Actor Model paradigm of multi-threaded workloads. The Actor Model operates on objects called actors to orchestrate concurrent computations, it does this by treating actors as separate entities that execute on an event loop on one or more threads, where the event loop simply looks at a list of events generated by actors and executes each event in order as long as it is not blocked by other events. Events are generated when actors pass messages to each other, in response the actor that receives the message can modify its local state, create other actors, send messages to other actors, execute arbitrary logic, and finally send a response to the message. This programming paradigm prevents the need for locks (and by extension, deadlocks) because the message passing mechanism alleviates the need to manually manage concurrency, opting to use the abstractions instead.

Actix was chosen primarily to give the flexibility of multi-threading without incurring the typical thought process overhead associated with multi-threading. Additionally, as a framework rather than a library, Actix serves as the basis to make REST webserver through the package Actix Web, which is further discussed under the sections 6.2.5 and 6.5.1.



### 6.2.5 Libraries

The Akriveia Beacon system will contain the following libraries.

|                                |  |
|--------------------------------|--|
| <b>Actix Web</b>               | The Actix Web library is a crate that extends the functionality of the Actix framework to create HTTP web servers.   |
| <b>Yew</b>                     | Yew is a crate that extends Rust's ability to compile to Web Assembly by adding the ability to dynamically generate html on the client in response to user events and DPU responses. |
| <b>Serial Port</b>             | Enables serial communication over USB between the DPU and beacons.   |
| <b>Rust Standard Libraries</b> | This is the standard Rust runtime library, and contains many useful boilerplate functions and generic types. This library is included with the basic installation of Rustc.          |

Table 7: Akriveia Beacon Dependencies - Rust

|                   |   |
|-------------------|---|
| <b>Arduino</b>    | The standard set of Arduino libraries based off of the standard C library tailored specifically for Atmel Atmega AVR microcontroller chips. |
| <b>ESP32 WiFi</b> | Arduino core for ESP32 WiFi/BLE chip  |
| <b>DWM1000</b>    | A library that offers basic functionality to use Decawave's DW1000 chips/modules with Arduino   |

Table 8: Akriveia Beacon Dependencies - Arduino

## 6.3 Model-View-Controller

The model-view-controller (MVC) paradigm is a common method of separating concerns between modules. It is an architectural pattern that separates concerns in a general way, allowing for code reuse and improving parallel development, it is also a common practice which allows others familiar with the paradigm to ramp up quickly on new projects. Unsurprisingly, the MVC is split into three components, the model, controller and view. The model describes data, data manipulation, as well as data storage. The view describes how to view the model. The controller acts as the means to communicate between the view and model based on data inputs, and is also capable of higher order operations such as between multiple models.

As the Actix Web library is less mature than frameworks in other languages such as Python's Django or Ruby's Rails, the DPU implementation will contain larger portion of boilerplate when compared to those other libraries. This allows for greater flexibility in implementation, at the cost of marginally higher development time. A benefit to writing a more tailored implementation, is that MVC and frameworks that closely follow MVC do not have a good answer for organizing functionality that falls outside of the architectures coverage such as background tasks. Akriveia is required to maintain communication with many beacons and perform calculations asynchronously from web requests, which is the perfect use case for the actor framework which fills the void of the model-view-controller.

## 6.4 Threading Model

The purpose of multi-threading the DPU is to make use of the hardware provided, since all modern server and desktop processors have at least two dedicated hardware cores. Additionally, it is important for the DPU to have realtime responsiveness on the webserver, requiring that large computations such as trilateration for many beacons must be offloaded to other threads, keeping responsiveness high since the webserver thread is not blocked on computation while a request comes in. An additional consideration specifically for the proof of concept which uses blocking serial communication to communicate to the beacons, meaning that each beacon requires a dedicated thread to communicate with the DPU.

Rust and Actix make multi-threaded development easier by using message passing for inter-thread communication, at the cost of performance when compared to traditional mutexes and atomics. The implementation of message passing is a thread safe queue using mutexes and atomics to ensure data integrity, where multiple threads are able to send commands to another thread, which receive the message without blocking either thread. To further discuss the Actix threading model, some definitions are required:

1. Actor: A class that contains message handler callbacks.
2. Arbitor: A thread pool, where each thread in the pool is an event loop.

Once instantiated, the Arbitor spawns as many threads as indicated by its constructor and waits for actors to be spawned within the thread pool. By default, the number of threads in the pool is the number of cores on the CPU. The Arbitor is an environment for actors to execute, and as actors are spawned into the thread pool, they can send messages to other actors, or create other actors in response to external events such as from the file system or HTTP requests. As shown in Figure 28, the arbitor manages its own threads, which are all spawned at startup rather than on demand. Within each thread, the arbitor executes the event loop, which polls its queue, and acts on any messages in the queue by delegating the task to the actor the message is bound to. To pass a message to another actor, the sender must have the address of the recipient so that the arbitor can determine the destination, rather than a broadcasting system; this is shown in Figure 29.

The beacons and ID tags do not have a full operating system, so they can only rely on single threaded interrupt handling to "multitask". Interrupt handling does not provide full concurrency, instead it time slices the single core to provide the illusion of multitasking. The basics of Interrupt handling is that it calls a function as a result of an event such as an external device raising a bit high on the CPU, or when a timer goes off. In the Proof concept, the bluetooth library uses this functionality to handle connections. When transitioning to WiFi and UWB, similar functionality will be implemented in each respective technology's library.

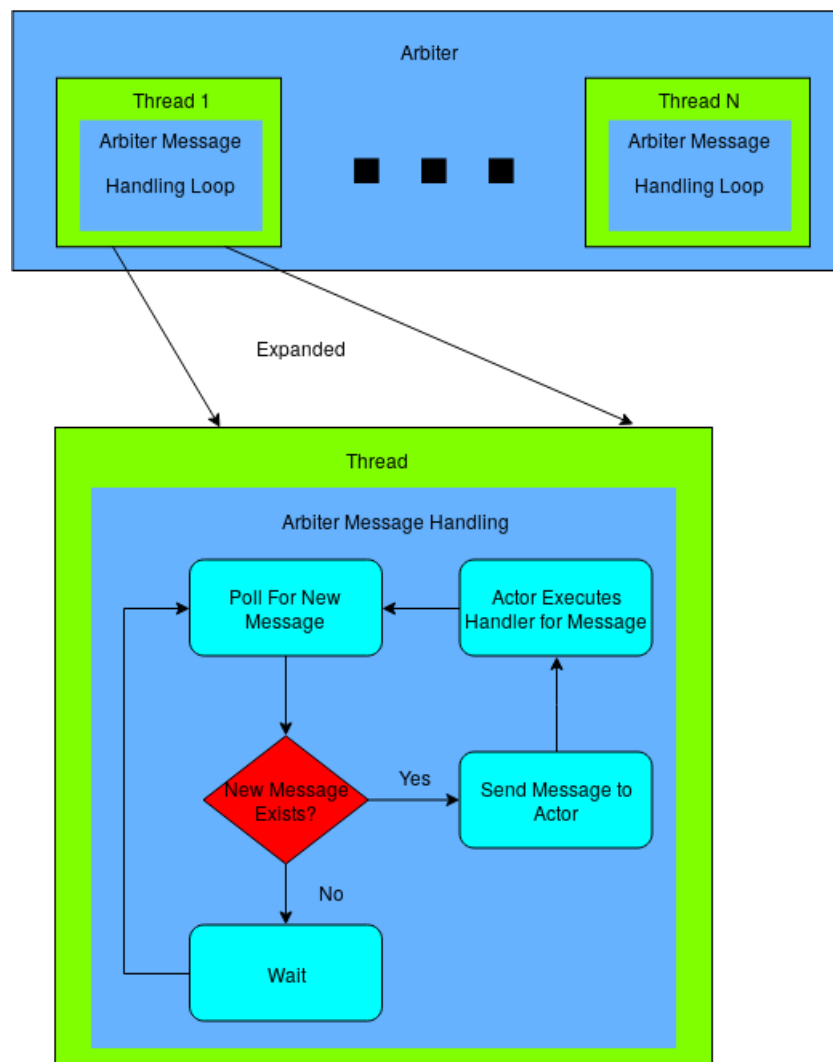


Figure 28: Actix Thread Model

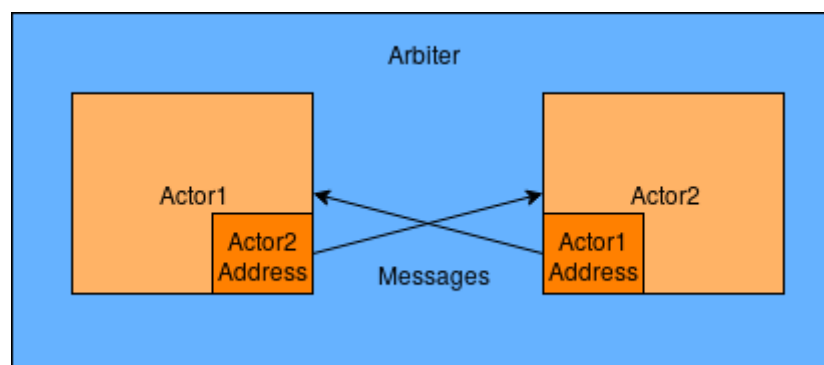


Figure 29: Actix Message Passing

## 6.5 Data Processor Software Architecture

In the proof of concept, the DPU creates an HTTP server actor and a beacon manager actor. The HTTP server waits on web requests, delegating tasks to the beacon manager and polling it for the necessary information. On startup, the beacon manager queries the number of serial arduino devices, spawning a thread for each device. Each thread spawned by the beacon manager represents a serial connection to a beacon. The beacon manager sends messages using a traditional Multiple Producer Single Consumer Message Queue (MPSC) defined in the Rust standard library, while each serial communication thread returns messages back to the manager through the actix message passing system. Figure 30 gives a graphical representation of the previous description. Actix uses MPSC internally for its message passing system.

The prototype and final versions bring in major changes to the software design. As shown in figure 31, a major change is the removal of the serial connection actor which is replaced with the UDP actor. The beacon manager remains, but instead controls the UDP actor rather than the serial connections. An additional role of the beacon manager is to delegate computation of trilateration to the trilateration processor actor, which will save data in memory until there are at least 3 data points for a single ID tag to determine its position. The trilateration processor makes the positional calculation and saves new position to disk via the ID tag model, which abstracts the database query. The beacon and map models perform a similar role, abstracting their associated database queries into a single location following MVC. Controllers mainly handle the model they were paired with, however the ID tag controller will also query the trilateration processor for real-time positional data and the beacon controller commands beacons indirectly through the beacon manager.

By design, the DPU is a client-server architecture, where the DPU is the server for a browser based client. The client-server architecture lends itself to be very flexible and is a very heavily used model for both consumer systems and emergency response systems alike. Once the backend starts executing, any computer able to access the IP of the DPU will have access to the system assuming the network is configured in such a way to allow this, and that the user has the correct credentials. Conversely with no additional development work the same DPU when hooked up to a monitor can act like a client based application by simply accessing localhost on a browser. Variations and alternatives to the client server-architecture were considered, but did not meet the needs of the Akriveia Beacon System, below are some of the choices considered and why they do not fit the role.

An electron application that utilizes browser technologies merges the backend and frontend together, behaving like a client based application, which will reduce deployment flexibility because a monitor connected directly to the DPU will be the only access point to the GUI, however the display will potentially need to be accessed by multiple users at a time, and will introduce high coupling between the frontend display and backend computation. Electron applications are also highly single threaded, meaning that they will have higher difficulty in utilizing all cores on the CPU, wasting money on hardware that could otherwise be used. Electron applications are also fixed to using a single language, Javascript, for both processing and display, whereas a webserver fixes the GUI implementation to Javascript however leaves the backend to a much broader range of languages where processing performance is more critical.

Compared to a GUI implemented using graphics driver interfaces such as OpenGL, Vulkan, DirectX, or any higher level libraries that provide interfaces to graphics drivers, a GUI implemented in HTML and Javascript will take much less time to implement because both Javascript and HTML abstract away memory concerns such as pointers, and will never have memory related runtime errors, as guaranteed by the browser. Browsers have widespread adoption, in every GUI based operating system at least one browser is installed by default, making them widespread and widely used. Due to widespread adoption, browsers provide the flexibility to display the GUI either on the computer hosting the server itself, or just as easily the GUI can be accessed through another computer at the customers discretion.

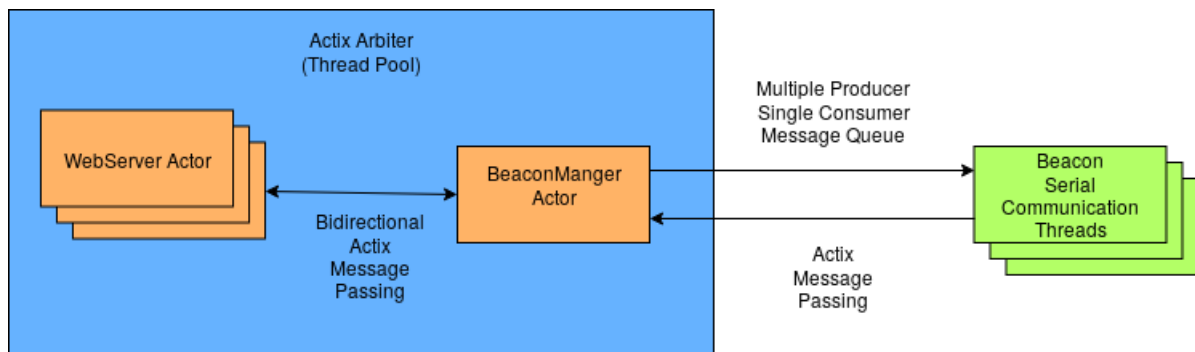


Figure 30: Proof of Concept Software Architecture

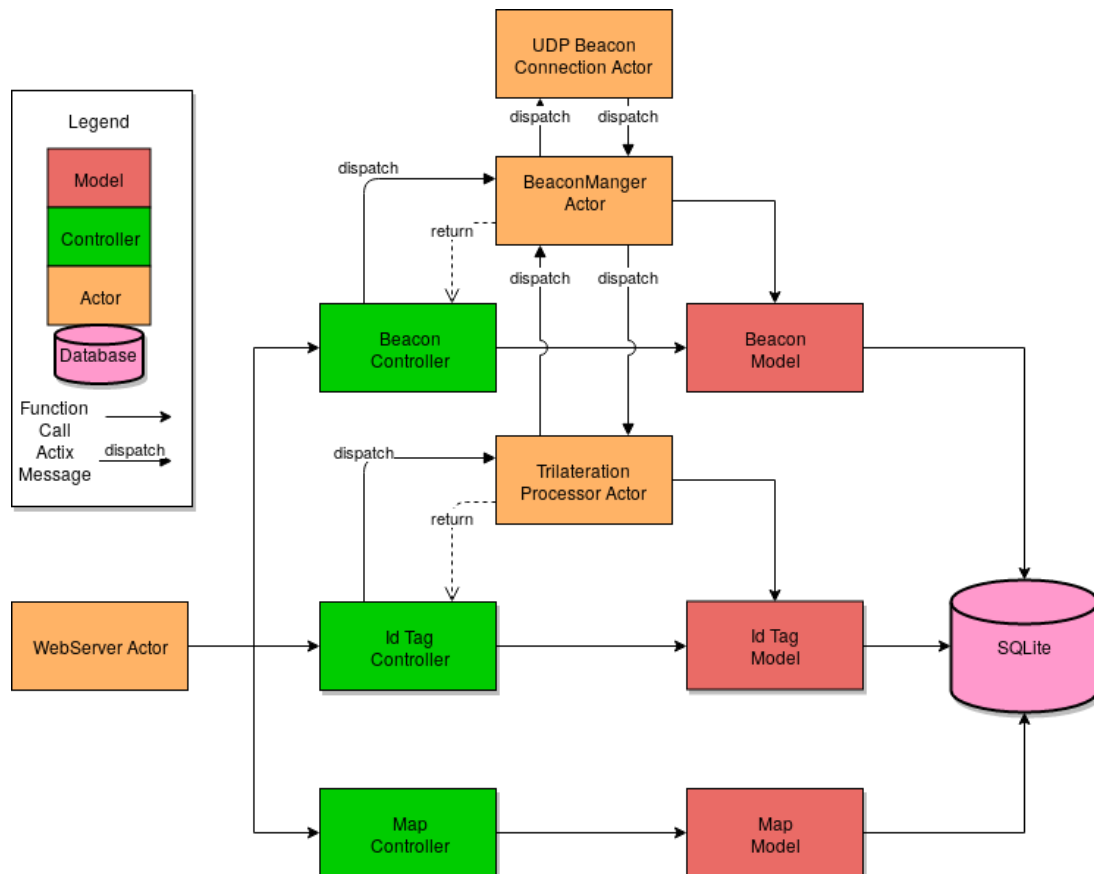


Figure 31: Final Software Architecture

### 6.5.1 Webserver Subsystem

The purpose of the Webserver Subsystem is to bridge the GUI and backend by providing a data transfer interface in the form of HTTP REST calls. A Webserver architecture is chosen because of its simplicity to implement, widespread adoption, and flexibility. The view is discussed further in Section 6.9: View.

The Webserver Subsystem will be designed to serve data to a browser, including the HTML and Javascript files (GUI files) that are displayed by the browser. Once the browser has the GUI files, it will make additional requests to the backend to keep updated information displayed on the GUI. The Webserver Subsystem will only allow logged in users to access sensitive data, and only if they have the correct access rights. Electron applications and traditional rendering APIs were also considered, however they each have trade-offs that do not fit with the design requirements of Akrievia.

A direct alternative to Actix Web is a web framework called Rocket. Rocket behaves more like a traditional web server: supporting Server Side Rendering and json parsing out of the box. This functionality comes at a cost in multiple ways, first rocket is primarily single threaded, second Rocket adds new features at the cost of reducing flexibility, and third rocket does not directly support the Actix Actor system, leaving all non-webserver code up to the developer to implement on their own. In comparison, Actix Web does support and extend the Actor system which makes for a more integrated and unified codebase, and by using the Actor system within the webserver, multithreadability is easily added as needed on a per endpoint basis. Rocket's additional out-of-the-box features are really just Rust crates in disguise, and so by using Cargo(Rusts package manager for crates) the same functionality can be quickly implemented within an Actix webserver at very little additional time cost.

### 6.5.2 Beacon Manager Subsystem

The role of the beacon manager is to commands to and from the individual beacon connections, and to give an abstraction for external systems that need not concern themselves with communication details, or how many beacons there actually are. The Beacon Manager is a singleton, which means there is only one of them in the entire system, and has implications on the design of the rest of the system. The most notable design influence is that the beacon manager can quickly become a bottleneck if it performs too much processing directly, and so to prevent this the beacon manager offloads as much work as it can to other subsystems.

### 6.5.3 Serial Beacon Communication Subsystem

In the proof of concept design, this actor deals with blocking serial communication with dedicated threads to keep the connection alive and to prevent web requests from blocking. This subsystem will remain as bare bones as possible, because it only serves the purpose of quickly creating a working system and will be discarded in future iterations of the DPU.

#### 6.5.4 Trilateration Processing Subsystem

In the final design, the trilateration processing subsystem takes in time of flight data messages from the beacon manager. The trilateration processor waits until it receives enough data points, and once it has 3 data points from 3 different beacons for one id tag it can perform a location calculation. While waiting for a full set of data, the incomplete set of data will be stored in a hash table where the mac address of each ID tag is the key and the contents are the array of data points. Once the array of data points hits the threshold size of 3 from adding a new data point, the processor can perform the trilateration calculation. The trilateration calculation can then be used to update the database entry for the corresponding ID tag by the processor.

To avoid going to the database for real-time updates of the ID tag locations, the ID tag controller will directly query the Trilateration Processor for the most up to date location information, reducing latency. The existence of the Trilateration Processing subsystem is necessary, because the beacon manager will quickly become a bottleneck due to the fact that it is a singleton in a multithreaded system. As such, the manager should do as little direct processing and blocking operations itself and instead favour offloading the tasks to other subsystems.

#### 6.5.5 UDP Beacon Communication Subsystem

In the final design, the UDP Beacon Communication Subsystem actor will communicate with the beacons over the UDP networking protocol. UDP was chosen because it has less overhead than TCP, and is more suited for real-time applications such as Akrievia.

### 6.6 Database

Akrievia Beacon System requires a database to persist map, beacon, and user information. In the time constraints of Capstone, the database will be a central location for data. Ideally a production system would support replication of the database to another location, but this is out of scope for the purposes of this project. SQLite was chosen primarily because of its simplicity to set up and minimal footprint. Most commonly used in phones and packaged alongside python, SQLite has a stable file format and large userbase, and because of this has built a reputation for not failing.

SQLite is open source, and because of this issues can directly be brought up to the developers along with help from a large amount of experience from the internet due to the large userbase. Open source projects are also freely available to browse to learn directly from the codebase when necessary. In Akrievia, the database is accessed through model objects to keep concerns local to a single set of similarly structured files, and to keep re-usability up in the codebase.

### 6.7 Models

Models are used to group database queries and model operations into a single file for each model, increasing reusability of the system. Akrievia is expected to maintain three different types of models: maps, ID tags, and beacons. The following tables 9, 10, and 11 show expected data entries and their type for each model, along with the purpose of each entry.

| User Model           |                 |   |
|----------------------|-----------------|---|
| Data Name            | Type            | Explanation   |
| User Type            | String          | Indicates the type of user, either admin, employee, emergency contact, or first responder.                    |
| Employee Id          | String          | Employee ID for each user, for internal housekeeping of the company that purchases Akriveia Beacon.           |
| Full Name            | String          | Human readable name to identify the user on lists.  |
| Phone Number         | String          | Phone number to contact the user.   |
| Emergency Contact    | ref:User        | Reference to another user as emergency contact. Emergency contacts will not have an employee id or tag id.    |
| Notes                | String          | Notes about the user, i.e. allergies or disabilities, etc.  |
| Id Tag Number        | 64 bit Integer  | table key, unique, id for each user.  |
| Coordinates          | 2D Vector       | Last known location of the user within the map.   |
| Last Seen Timestamps | Unix Timestamps | To identify the ID tag data is stale. Used to determine if the employee is at work while the disaster occurs. |
| Map ID               | ref:Map         | Last known map the user was located at.   |

Table 9: User Model

| Beacon Model |         |   |
|--------------|---------|---|
| Data Name    | Type    | Explanation   |
| MAV address  | String  | unique device identifier for beacon, table key.   |
| Name         | String  | Human readable String for the device.   |
| Map ID       | ref:Map | Reference to a floor/map - this is really just the floor number. Shows the many-to-one relationship between beacons and floors. |
| Notes        | String  | Additional Beacon information (i.e. floor and location).  |

Table 10: Beacon Model

| Map Model   |        |  |
|-------------|--------|--|
| Data Name   | Type   | Explanation  |
| Name        | String | Human readable floor name for gui.   |
| Floor ID    | String | Table key, unique floor number of the map, this is not an integer to accomodate the possibility of odd floor naming conventions. e.g. floor 1A, or basement. |
| Bitmap data | Binary | Bitmap for the floor blueprints to view on the gui.  |

Table 11: Map Model



## 6.8 Controllers

Controllers are part of the MVC philosophy, unfortunately Actix does not directly support the controller concept, however this functionality can be implemented manually using Actix Web primitives. Each controller exposes available operations for the frontend to manipulate data in a controlled and secure manner. The controller manages data using the models to perform direct manipulations and can aggregate data that cannot otherwise be done by models. Table 12 shows the list of expected controllers in the DPU, one to match each model.

| Map Model       |  |
|-----------------|--|
| Controller Name | Explanation  |
| Maps            | Handles requests for map operations such as to create, update, delete the map instances.             |
| Beacons         | Handles requests for beacon operations such as to create, update, delete beacon the model instances. |
| Users           | Handles requests for user(id tag) operations such as to create, update and delete user instances.    |

Table 12: Controllers list

## 6.9 View

The view is part of the MVC philosophy, since its main job is to display data to the GUI in an intuitive way. In Akriveia Beacon, the view is a static webpage served by the Actix Webserver, and compiled from the backend. Traditionally, browsers render HTML and Javascript served by the backend webserver, however, in this project the frontend is written entirely in Rust and compiled to a small HTML stub and webassembly which then dynamically generates more HTML for the browser to render. The benefit of this approach is that it creates strong separation between the view and controller, and also greatly reduces the number of endpoints that need to be implemented on the controller. This way it can reduce code duplication and surface area of the webserver, which lower the possibility of exploitation by reducing exposure to a small set of well defined operations.

Client side Rust is possible via the Yew crate, which is a Rust package that leverage the web assembly LLVM backend for Rust. Yew adds additional facilities to Rust for webassembly generation such as macros that allow HTML to be written in lined to Rust, similar to Reacts JSX. Yew also supports the concept of Agents, which is their version of Actors inspired by Actix and Erlang. Yew also has the ability to interact with npm packages, in case the functionality is not already implemented in Rust. As Yew is influenced by React, it is very possible to simply write the GUI using React.

The drawback of using React is that it is written in Javascript, meaning that an implementation using React would make the language of the backend different from the language of the frontend creating overhead in development. Furthermore, react contains strict definitions and interfaces which would need to be implemented twice, once in each language adding tedious boilerplate. The modern Javascript development environment as a whole is also getting fairly unwieldy, as of 2019 more packages are being added which each introduce transformation steps of the code. An example of this workflow could involve writing typescript which is transformed into Javascript, then minimizing the Javascript into unreadable but more performant minimized Javascript. Additional steps would similarly be added for Babel scripts. In the example both of the steps are added to the build manually using webpack which incurs more development overhead to understand and configuration.

Yew on the other hand has an opaque compilation step that transforms Rust code into webassembly, which does not require any additional build steps to configure other than installing the package. One of the downsides to using Rust compiled to webassembly is that it simply cannot be debugged since it is more similar to binary, but this issue has not arisen yet since framework is very simple to use. It does seem, in theory, possible to compile Rust to Javascript which can be debugged in the browser, but debugging Javascript which was generated from another language will likely not be a very fruitful exercise.

Server Side rendering of templates using libraries such as Askama and Tera were also considered instead of creating a single page website. This method of rendering is fantastic for form submissions, however real time and dynamic pages strain the abilities of server side rendering. Akrievia will require real time updates of maps, which will look and perform much better when rendered on the client side rather than having the backend needlessly re-render the same page with different data. Typically, when real time functionality is required for server side rendered applications, some pages are converted to using client side rendering while others remain server side rendered, requiring libraries like React or JQuery. By simply using client side rendering, the frontend is consistently implemented throughout, rather than requiring more than one type of rendering. Please see Appendix B: User Interface and Appearance for additional details on the UI layout.

## 6.10 Security

Due to the properties of Rust, many security concerns regarding bad input data and memory issues are mitigated. One such mitigation is built in bounds checking of all array accesses, meaning that its not possible to read or write memory that has not been properly allocated by the executing process. An additional level of security is that rust reduces the number of invalid computational state, removing the burden from the programmer which could potentially get it wrong, which would create vulnerabilities. As rust is a strongly typed language, it also benefits from static type checking which eliminates entire classes of bugs that also could lead to vulnerabilities.

All stored user data will be encrypted using the latest encryption algorithms, namely the sha3-512 algorithm. In memory data will be unencrypted for as short of a time as possible to reduce the surface area of potential attacks.

## 6.11 Software Design Requirements

The Akriveia Beacon system is composed of an intricate software stack developed in rust. The data processing unit is the main computational unit and will be implemented with trilateration algorithms to locate ID tag positions in near real time. In order for the software system to be reliable, secure and accurate the following design specifications were made.

|                     |  |
|---------------------|--|
| <b>REQ.SW.1 - C</b> | The DPU software stack must be implemented using Rust  |
| <b>REQ.SW.2 - C</b> | The DPU must contain a Linux based operating system  |
| <b>REQ.SW.3 - C</b> | Each beacon must acknowledge a stop receiving command to stop sending ID tag location data to data processing unit |
| <b>REQ.SW.4 - C</b> | Each beacon must acknowledge a start sending command to start sending ID tag location data to data processing unit |
| <b>REQ.SW.5 - P</b> | The admin must login through credential verification system before before accessing system data                    |

Table 13: Software Design Specification

## 7 Conclusion

As urban centers around the world experience rapid growth and changes so does the risk of being potentially trapped within buildings during disasters. The time period right after a disaster strikes is the most critical time for saving victims lives. In current practices, first responders have limited time to evaluate the situations when they arrive on the scene of disaster and must take crucial actions accordingly. Searching the incident building for possible victims is one of the major tasks undertaken by first responders after an incident occurs. The lack of timely information could be the difference between life and death in such situations.

As such, a reliable and accurate indoor location rescue system is needed to aid first responders in locating trapped personnel. The Akriveia Beacon is a system of anchor beacons and ID tags controlled by ESP32 micro-controllers and communicating via Decowave DWM1000 UWB modules, along with trilateration algorithm to accurately obtain near real time location of trapped personnel within buildings during the event of a disaster. The location data is then reported to a portable data processing unit via a closed WiFi network, which can be interacted with directly by emergency first responders and operators to provide accurate and reliable information for the search and rescue effort.

The system overview, design, and constraints of the Akriveia Beacon system are clearly established in this document, as well as presenting the complete system design specifications. By provided a detailed outline of the design specifications this document is intended to be used as a design reference for the engineers at TRIWAVE SYSTEMS, as well as to provide detailed insight for the hardware and software designs required for the Akriveia Beacon product. These design specification outlines high level system architectures, system functions and implementation of the Akriveia Beacon product through three different phases of development including: the proof-of-concept (completed August 2019), prototype, and final product (completed December 2019).

As a product that could potentially affect the outcome of disaster relief operations, the Akriveia Beacon is designed with the utmost care. As aforementioned, TRIWAVE SYSTEMS is dedicated to creating a reliable and robust system design to improve disaster search and rescue operations with human safety as the pivotal focus.

## 8 References

- [1] Statistics-Canada. *Interview on Rust, a Systems Programming Language Developed by Mozilla*[online]. Aug. 2014. URL: Available%20at:%20https://www150.statcan.gc.ca/n1/daily-quotidien/160916/dq160916c-eng.htm%20[Accessed%208%20June.%202019]..
- [2] Statistics-Canada. *Fire-related deaths and persons injured, by type of structure*. 2019. URL: [Online] .%20Available:%20150.statcan.gc.ca/t1/tbl1/en/tv.action?pid=3510019501. [Accessed:%2021-May-2019]..
- [3] A. Rajabifard H. Tashakkori and Mas Kalantari. *A new 3D indoor/outdoor spatial model for indoor emergency response facilitation*. 2015. URL: Available:%2010.1016/j.buildenv.2015.02.036..
- [4] A. Rajabifard H. Tashakkori and M. Kalantari. *FACILITATING THE 3D INDOOR SEARCH AND RESCUE PROBLEM: AN OVERVIEW OF THE PROBLEM AND AN ANT COLONY SOLUTION APPROACH*. 2016. URL: Available:%2010.5194/isprs-annals-iv-2-w1-233-2016..
- [5] Terabee. *Time-of-Flight principle: Technologies and advantages - Terabee*. 2019. URL: [Online] .%20Available:%20https://www.terabee.com/time-of-flight-principle/.%20[Accessed:%2022-Jun-2019]..
- [6] eTutorials.org. *Understanding RF Signals*. 2018. URL: [Online] .%20Available:%20http://etutorials.org/Networking/wn/Chapter+3.+Radio+Frequency+and+Light+Signal+Fundamentals+The+Invisible+Medium/Understanding+RF+Signals/.%20[Accessed:%2025-Jun-2019].
- [7] H. Wolverson. *Path Loss*. 2015. URL: [Online] .%20Available:%20http://wisptools.net/book/bookc3s1.php. [Accessed:%2025-Jun-2019].
- [8] E. Farella D. Giovanelli. *RSSI or Time-of-flight for Bluetooth Low Energy based localization? An experimental evaluation*. 2018. URL: [Online] .%20Available:%20https://hal.inria.fr/hal-01995171/document.%20[Accessed:%2025-Jun-2019].
- [9] iotbymukund. *How to Calculate Distance from the RSSI value of the BLE Beacon*. 2016. URL: [Online] .%20Available:%20https://iotandelectronics.wordpress.com/2016/10/07/how-to-calculate-distance-from-the-rssi-value-of-the-ble-beacon/.%20[Accessed:%2025-Jun-2019].
- [10] spirent. *Fading Basics*. 2018. URL: [Online] .%20Available:%20http://www.spirent.cn/gui/~media/fc29acc7f9934903a6cf7dbc809249ae.ashx.%20[Accessed:%2028-Jun-2019].
- [11] espressif. *Development Board Espressif Systems*. 2019. URL: Available%20at:%20https://www.espressif.com/en/products/hardware/development-boards%20[Accessed%2028%20Jun.%202019]..
- [12] Decawave. *DWM1000 Module - Decawave*. 2019. URL: [Online] .%20Available:%20https://www.decawave.com/product/dwm1000-module/.%20[Accessed:%2022-Jun-2019]..
- [13] Raspberrypi.org. 2019. URL: [Online] .%20Available:%20https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/.%20[Accessed:%2028-%20Jun-%202019]..
- [14] R. Heydon. *An Introduction to Bluetooth low energy*. 2016. URL: [Online] .%20Available:%20https://datatracker.ietf.org/meeting/interim-2016-t2trg-02/materials/slides-interim-2016-t2trg-2-7.%20[Accessed:%2028-Jun-2019].

- [15] Sewio. *UWB Technology*. 2018. URL: [Online] .%20Available:%20https://www.sewio.net/uwb-technology/.%20[Accessed:%2028-Jun-2019] .
- [16] W1.fi. *hostapd: IEEE 802.11 AP, IEEE 802.1X/WPA/WPA2/EAP/RADIUS Authenticator*. 2019. URL: [Online] .%20Available:%20http://w1.fi/hostapd/.%20[Accessed:%2003-%20Jul-%202019] ..
- [17] lastminuteengineers. *Insight Into ESP32 Sleep Modes & Their Power Consumption*. 2019. URL: Available%20at:%20https://lastminuteengineers.com/esp32-sleep-modes-power-consumption/%20[Accessed%2028%20Jun.%202019] ..
- [18] T. ARTICLES et al. *Wireless RF Energy Harvesting: RF-to-DC Conversion and a Look at Powercast Hardware*. 2019. URL: [Online] .%20Allaboutcircuits.com,%20Available:%20https://www.allaboutcircuits.com/technical-articles/wireless-rf-energy-harvesting-rf-to-dc-conversion-powercast-hardware/.%20[Accessed:%2031-May-2019] ..
- [19] Abel Avram. *Interview on Rust, a Systems Programming Language Developed by Mozilla*. Aug. 2012. URL: https://www.infoq.com/news/2012/08/Interview-Rust/%20[Accessed:%2021-June-2019] ..
- [20] Graydon Hoare et al. *Releases.md*. May 2015. URL: https://github.com/rust-lang/rust/blob/master/RELEASES.md%20[Accessed:%2021-June-2019] ..
- [21] Bjarne Stroustrup and Herb Sutter. *C++ Core guidelines*. May 2019. URL: https://github.com/isocpp/CppCoreGuidelines/blob/master/CppCoreGuidelines.md%20[Accessed:%2021-June-2019] ..
- [22] *Arduino Style Guide*. May 2019. URL: https://www.arduino.cc/en/Reference/StyleGuide%20[Accessed:%2021-June-2019] ..

## 9 Appendix A: Supporting Test Plans

All test cases for each of the four test plans follows the below format. These test cases are derived from the requirements of the *Design Specifications*.

[ **TEST**-#]

| Code        | Definition              |
|-------------|-------------------------|
| <b>TEST</b> | Test Plan abbreviation. |
| #           | Test Case ID            |

Table 14: Test Case Encoding

| Planning Stage   | Abbreviation Code |
|------------------|-------------------|
| Proof of Concept | C                 |
| Prototype        | P                 |
| Final Product    | F                 |
| Usability        | U                 |

Table 15: Planning Stage Abbreviation Code

Each test case is grouped into these four sections and their IDs start with a letter representing the group. These letters are shown in bold above. The *Design Specifications* test plans consists of Test cases with expected results. These test cases have been modified from the previous test plans from the *Requirements Specifications* to reflect current project progress.

## 9.1 PoC Test Plan

The Proof of Concept (PoC) Test Plan includes testing procedures for verifying and validating the *Design Specifications* under a formal test environment. The goal of the PoC Test Plan is to ensure the basic requirements stated in the *Requirement Specifications* are refined and developed into test cases. Three main goals for the PoC testing are as follows:

1. 2.4 GHz chips are able to receive and transmit data accurately
2. ESP32 (MCU) receives the transmission data
3. Raspberry Pi receives data serially from ESP32 MCU

| PoC Test Plan |   |  |
|---------------|---|--|
| ID            | Test Cases  | Expected Results   |
| P-01          | ESP32 bluetooth modules detects bluetooth servers in the proximity.                                       | Console Serial output shows all nearby bluetooth devices with MAC addresses.   |
| P-02          | ESP32 uses the RSSI formula to output values of nearby bluetooth devices.                                 | ESP serial out displays RSSI of surrounding bluetooth devices by MAC addresses.  |
| P-03          | RSSI measurements on serial output should vary at different distances.                                    | RSSI measurements at 0.1m, 0.5m, 1m and 2m. The RSSI measurements should become more negative as distance increases.                     |
| P-04          | Measure RSSI measurements at 1m to get measured power   | Log the average RSSI at 1m at different locations. The average measurements at each location is the measured power at specific location. |
| P-05          | ESP32 beacon should only output ESP32 ID tag MAC address.   | Serial output should output RSSI of ID tags. Beacon programming filters out all RSSI measurements besides ID tags.                       |
| P-06          | ESP32 deep sleep mode activates after 2min period of inactivity.  | ESP32 shuts off main processor and peripherals. MCU consumes only 10uA in sleep mode.  |
| P-07          | DPU transfers data through serial to a ESP32 beacon.  | "Start" command sent serially by the DPU and received serially by the ESP32 beacon.  |
| P-08          | Raspberry Pi (DPU) should receive RSSI measurements through serial communication from all 3 ESP32 beacons | Console output of the received RSSI measurements from 3 beacons.   |
| P-09          | RSSI measurements are taken from all 3 beacons and converted into distances by Distance-RSSI formula      | Console output of approximate distances of the 3 beacons to the ID tag.  |

Table 16: PoC System (General) Test Plans - Part 1



| PoC Test Plan |  |   |
|---------------|--|---|
| ID            | Test Cases   | Expected Results  |
| P-10          | Distances from the Distance-RSSI formula are used in the Trilateration algorithm to calculate approximate location | Location of the ID tag is given as (x, y) coordinates with (0, 0) being the location of the bottom left beacon. |
| P-11          | Real-time tracking of the ID tag is calculated and displayed on the Raspberry Pi console output                    | Location of the ID tags in (x, y) are constantly displayed on the serial output of DPU.                         |

Table 17: PoC System (General) Test Plans - Part 2

## 9.2 Prototype Test Plan

The Prototype Test Plan includes testing procedures for verifying and validating the *Design Specifications* under formal test environment. The goal of the Prototype Test Plan is to ensure the requirements stated in the *Requirement Specifications* are met. Three main goals for the Prototype testing are as follows:

1. UWB chips are able to receive and transmit data accurately
2. ESP32(MCU) integrates with the UWB modules and receives transmission data
3. Raspberry Pi (DPU) calculates the location coordinates from Time of Flight (ToF)

| Prototype Test Plan |  |   |
|---------------------|--|---|
| ID                  | Testing Criteria   | Observations  |
| T-01                | UWB chip and breakout board are soldered properly. SPI Bus Pins are operating as expected                          | MISO/MOSI pins are outputting expected values. SCLK is receiving proper clock signals. IRQ pin is generating interrupts and CS pin remains at the correct active-low state. |
| T-02                | UWB chip and breakout board are soldered properly. Pins are operating as expected                                  | Device sleep activate by EXTON. GPIO7 transmits data. RSTn pin resets DWM1000. WAKEUP pin trigger DW1000 to wake.   |
| T-03                | ESP32 transfers data from the UWB module via SPI communications.   | ESP32 receive expected data from SPI and output to console. pins are functioning as expected.   |
| T-04                | UWB chips receive and transmit simple messages to and from other UWB chips through UWB frequencies of 3.5-6.5 GHz. | Received data is shown serially on the consoles for the chips.  |
| T-05                | ESP32 are woken out of deep sleep by the GPIO #26 pin. ESP32 should wake the UWB through the WAKEUP pin.           | ESP32 and UWB are in active state; Start operating as programmed.   |
| T-06                | ESP32 go into sleep after 2 minutes of inactivity. UWB module is sent to sleep through the EXTON pin.              | ESP32 and UWB are in deep sleep mode.   |
| T-07                | ID tag receives "emergency" signal from beacons. UWB chip ensure ESP32 remains awake through the EXTON pin.        | ESP32 and UWB remains in active state. ESP32s are awake and do not go back to sleep.  |
| T-08                | ID tags are to be powered by the battery.  | ID tags are functional using battery power.   |
| T-09                | MCU receives timestamps from UWB and calculates ToF from sent and received data.                                   | Serial output of ToF measurement. There should be manual verification of calculation.   |

Table 18: Prototype Test Plans - Part 1

| Prototype Test Plan |   |   |
|---------------------|---|---|
| ID                  | Testing Criteria  | Observations  |
| T-10                | ToF are taken and used in the distance-ToF formulas to generate coordinates.  | Distances are displayed on serial output. Require manual verification of calculations.            |
| T-11                | Distances estimation from ToF is sent to DPU for trilateration calculation    | Serial output from Raspberry Pi showing the x-y coordinates from trilateration algorithm.         |
| T-12                | DPU should display x-y coordinates on webpage.                                | Console output of x-y coordinates of the ID tag in reference to the bottom left beacon as (0, 0). |
| T-13                | Real-time tracking of the ID tag in x-y coordinates on the console            | Webpage should show real-time changes in the x-y coordinates of the ID tag.                       |
| T-14                | DPU hosts the web server. Webpage can be accessed through DPU's wifi network. | Devices can access the webpage browser.   |

Table 19: Prototype Test Plans - Part 2

### 9.3 Final Product Test Plan

The Final Product Test Plan includes all the testing procedures for verifying and validating the *Design Specifications* under a formal test environment. The goal of the Final Product Test Plan is to ensure the basic requirements stated in the *Requirement Specifications* are refined and developed into test cases. Three main goals for the Final Product testing are as follows:

1. 2.4 GHz chips are able to receive and transmit data accurately
2. Arduino micro-controllers(MCU) receives the transmission data
3. Raspberry Pi receives data serially from Arduino M

| Final Product Test Plan |   |   |
|-------------------------|---|---|
| ID                      | Testcases   | Observations  |
| F-01                    | Test for the accuracy of UWB ToF to ensure accuracy is within $\leq 1\text{m}$ .                                      | Different distances set up to test ToF estimation accurate within error range of 1m.  |
| F-02                    | Emergency signal initiates emergency mode for all beacons and ID tags.  | All beacons are now in emergency mode. ID tags woken up will go into emergency state and stay woken up.   |
| F-03                    | Beacons forward data from farthest ID tags to the Raspberry Pi (DPU) so that all ID tags can be displayed on the map. | All ID tags distances are received by the DPU.  |
| F-04                    | ID tags should drain minimal amount of power ( $\leq 10\mu\text{A}$ ) when in deep sleep mode.                        | Measured current drawn by ID tag in deep sleep is less than $10\mu\text{A}$ .   |
| F-05                    | ID tags are woken up by a button press and checks for emergency signals from beacons.                                 | ID tags should go into emergency mode if beacons are sending emergency signals. Otherwise, ID tags should go into deep sleep mode after 2 minutes of inactivity |
| F-06                    | Beacon uses UDP protocol for communication to send out packets of data continuously without handshaking.              | Beacons should continue to send and receive timestamped packets of data.  |
| F-07                    | The ID tag's components, ESP32 wifi and MCU module, UWB module and housing LED, are all powered by the battery.       | All 3 components are operational and can all be powered by the battery.   |
| F-08                    | The beacon's components, ESP32 wifi and MCU module, UWB module and housing LED, are powered through AC outlet.        | All 3 components are operational and no issues occur.   |

Table 20: Final Product Test Plans - Part 1

| Final Product Test Plan |   |  |
|-------------------------|---|--|
| ID                      | Testcases   | Observations   |
| F-09                    | All the components of the ID tag are integrated together: ESP32 (MCU) and wifi modules, UWB modules , the battery and ID tag housing. | ID tag is functional and all components are working together.                          |
| F-10                    | Beacons use UDP protocol for communication to the Raspberry Pi (DPU) for real-time tracking.  | Raspberry Pi updates the ID tags location in real-time to provide up-to-date tracking. |
| F-11                    | Beacons run on emergency battery power when A/C power supply is unavailable.  | When power is disconnected, Beacons run no battery power until A/C power is available. |
| F-12                    | Raspberry Pi (DPU) displays the ID tag locations on a scaled map GUI.   | All ID tags are shown on the scaled map with an accuracy of $\leq 1\text{m}$ .         |
| F-13                    | Move the ID tags around the floor to check for real-time tracking.  | GUI map shows ID tags current location with $\leq 5\text{s}$ delay between location.   |
| F-14                    | Check if the MAC address and status in the GUI match with the devices.  | ID tags/Beacon statuses in the GUI are accurate and reflect the current situation.     |
| F-15                    | Test for any bugs or possible unauthorized access to data in the system.  | Automated tests to look for any bugs or issues that arise. Manual testing is required. |

Table 21: Final Product Test Plans - Part 2

## 9.4 Usability Test Plan

The Usability Test Plan includes all the testing procedures for verifying and validating the Usability Specifications under a formal test environment. The goal of the Usability Test Plan is to ensure the basic requirements stated in the **Appendix B: User Interface and Appearance** section of the document are met. Three main goals for the Usability testing are as follows:

1. ID tags and Beacons are intuitive to use and implement
2. GUI is easy to navigate and access information for Primary Users
3. GUI is easy to edit and customize configuration for Secondary Users

| Usability Test Plan |   |  |
|---------------------|---|--|
| ID                  | Test Cases  | Observations   |
| U-01                | Connecting to the Beacons' wifi network should be simple and quick.   | Select the network on the device's available wifi network. Input the password and the device should be connected.  |
| U-02                | Test if webpage address is correct and on-line.   | Inputting the webpage address should bring up the login page for Triwave Systems Akriveia.   |
| U-03                | Test if the login credentials entered matches the one in the system database. Give an appropriate response. | Entering the incorrect login credentials gives an login error. Prompts the user to try again. Entering the correct login credentials goes to the System Status or user homepage. |
| U-04                | Test if primary users sees only the Primary User Map View and Status View.                                  | Primary user should not have access to any system configurations, adding map views or adding user views.   |
| U-05                | Test if secondary users can see Add Map View, Beacon View and Add User Views.                               | Secondary user should have access to any system configurations, adding map views or adding user views.   |
| U-06                | Adding maps, beacons or editing user information should be intuitive and simple.                            | IT personnel with minimal training can add maps, beacons or edit user information without frustration or difficulty.   |
| U-07                | Installing new beacons should be intuitive and straight-forward.  | Adding beacon page is easy to navigate and form is simple to fill out. IT personnel without any training of Akriveia system should fill out the form without difficulty.         |

Table 22: PoC Software Requirement Test Plans - Part 1

| Usability Test Plan |   |   |
|---------------------|---|---|
| ID                  | Test Cases  | Observations  |
| U-08                | ID tag design has clear indication where the button is pressed.               | Users with no training understands intuitively where to press on the ID tag.  |
| U-09                | In emergency state, ID tag button when pressed will flash green continuously. | ID Tags LED light will continue to flash green in emergencies.  |
| U-10                | ID tags LED flashes the correct light in non-emergency mode                   | LED will flash green for 1 sec if ID tag is operational and has no problems LED will blink yellow for 3 sec if ID tag battery is low and need replacement. LED will blink red for 3 sec if ID tag is not operational or has issues. Battery is on critical low power. |
| U-11                | In emergency state, beacons will flash green continuously.                    | Beacons LED light will continue to flash green in emergencies.  |
| U-12                | Beacons LED flashes the correct light in non-emergency mode                   | LED will flash green for 1 sec if Beacon is operational and has no problems LED will blink yellow for 3 sec if Beacon has been reset. LED will blink red for 3 sec if ID tag is not operational or has issues.  |

Table 23: PoC Software Requirement Test Plans - Part 2

## 10 Appendix B: User Interface and Appearance

### 10.1 Introduction

The User Interface Design Appendix provides a detailed description and analysis of the Akriveia Beacon System in terms of the design, communication, and operations with the intended users. The Akriveia Beacon system is an advanced indoor location rescue system designed to aid first responders during an emergency search and rescue situation by providing accurate location of trapped victims within commercial buildings. The primary users to interact with the Akriveia Beacon system will be first responders such as fire fighters or emergency management personnel. The secondary users to interact with the system would be administrators or IT technicians that would utilize or perform maintenance and upkeep of the system. Lastly, the tertiary users would be the employees of the company using the Akriveia Beacon system. The employee will only interact with the system by turning the ID tags on during an emergency. Since the system is intended to operate under extremely stressful environments and situations, the user interface must be as clear and intuitive to use as possible to ensure that first responders can operate at peak efficiency along side the Akriveia Beacon system.

#### 10.1.1 Purpose

The focus of this user interface design appendix is to act as a reference for engineers at TRI-WAVE SYSTEMS throughout development. In order to create an interface that is both clear and intuitive for the intended user, the hardware and software interfaces must follow strict design standards and requirements. Such standards and requirements will ensure that during the intended operating scenario, the system would not cause users unexpected error due to implications of insufficient operating knowledge or unforeseeable circumstances. These design requirements will be presented in conjunction with the three specific development phases: the proof-of-concept phase, prototype phase, and the Final product phase.

#### 10.1.2 Scope

This document section includes detailed overview of user and technical analysis, engineering safety and standards, and usability testing in order to provide sufficient understanding of the user interface for the Akriveia Beacon system. As a system to be operating under disaster or emergency situations, the Akriveia Beacon must require some form of basic user knowledge in order for different parties of the user base to operate the system sufficiently. Outline of the required user knowledge allowing basic usage of the Akriveia Beacon system will be presented. As well as the following seven fundamental technical analysis principles will be considered when making design choices for the user interface: discoverability, feedback, conceptual models, affordances, signifiers, mappings, and constraints; as outlined from Don Norman's *The Design of Everyday Things* [23]. Finally, the appendix will include analytical and empirical system test plans with different scenarios that is aimed at testing how the Akriveia Beacon system would operate under each specified condition during each stage of the development cycle. The test cases covered will provide additional quality assurance of the final product to ensure that the Akriveia Beacon system is both accurate and reliable for its intended purpose.



## 10.2 User Analysis

The Akriveia Beacon system has three levels of targeted users: emergency first responders, IT or systems administrators, and company employees. Emergency first responders are considered to be the primary users, the system will provide high level layer interaction where only the most necessary elements are provided allowing intuitive access, decreasing the possibility of mistakes. The IT/system administrators are secondary users, who provides maintenance and upkeep of the system. Tasks such as managing ID tags, accounts, performing system wide maintenance, and perform potential upgrade, repair, and replacement procedures. Lastly, the tertiary users would be employees of the company using the Akriveia Beacon system. Employees would wear the ID tags alongside their everyday carry items such as mobile phones or access cards. The different levels of users will require different levels of interaction between them and the system. As such, each level of users will have requirements, access and control of the system. This section of the document will outline the different levels of user interactions, user background requirements, and detail some of the basic use cases for the primary, secondary, and tertiary users of the Akriveia Beacon system.

The primary users of the Akriveia system are the emergency first responders who are the first to arrive and provide assistance at the scene of an emergency, accident or disaster. First responders typically include paramedics, emergency medical technicians, police officers, firefighters, rescuers, and other trained professionals. In the intended situation where a search and rescue operation will be performed, the targeted primary user to be interacting with the system will be considered to be the person who is the top executive rank or commanding officer (Fire Chief) of the fire department on scene. The fire chief will cover the standard operating guidelines (SOGs) include basic communications with firefighter units deployed into buildings [24]. For the primary user, brief background knowledge on the operation of basic electronic equipment such as laptops and tablets are essential. Since user interaction between the primary user and the Akriveia Beacon system is through a graphical user interface hosted on basic electronic equipment, the user is required to have some form of familiarity with such devices. Once the system is incorporated more into current infrastructure, training could also be provided to the primary users if needed. Furthermore, basic comprehension of blueprint reading and ability to recognition and understanding of simple legends, icons, and other associated information on the UI is required. In addition, the primary users should have the grammatical prowess to understand the English language. Having fulfilled these user requirements, primary users can optimally benefit from the Akriveia Beacon system.

The secondary users of the Akriveia system are system administrators or IT technician that will be performing registration, and maintenance of the system. For secondary users, formal technical background is required; since the secondary users will be performing tasks such as installation and configuration of appropriate software and functions according to specifications. As well as to ensure the security and privacy of the networks and computing systems. Their primary tasks include managing ID tag accounts associated with employees, perform inspection of equipment such as beacons, ID tags, and data processing unit. As well as to ensure functionality of the system by enabling and operating the system during disaster drills.

Lastly, the tertiary users are considered to be the employees of the company that will be incorporating the Akriveia beacon system into their infrastructure. The tertiary users require minimal interaction with the system during normal every day operation since they will only need to carry the ID tags on them while on company grounds. However, during emergencies or disaster, if needed the tertiary user must be able to enable the ID beacons to enable broadcasting of their location so that the primary users are able to locate them through the GUI provided by the system.

## 10.3 Technical Analysis

This section will analyze the consideration for Seven Elements of UI Interface as outlined by Don Norman for the Akriveia Beacon system; which includes the following design factors, discoverability, feedback, conceptual models, affordances, signifiers, mappings, and constraints [23]. By incorporating these design element in to the system, the usability and quality of the final product can be substantially improved.

### 10.3.1 Discoverability

Discoverability: Is it possible to even figure out what actions are possible and where and how to perform them? In the context of product and interface design, discoverability is the degree of ease with which the user can find all the elements and features of a new system when they first encounter it [25]. The Akrivia beacon system is designed to operate under emergency situations and disasters, which means that it is paramount that the UI creates discoverability for its users. The overall interaction should be simple enough for each level of users to comprehend and understand without the need for much interpretation.

Primary user - First Responders' main point of interaction with the Akriveia Beacon system is through the GUI. The GUI will be simple with intuitive design allowing for quick understanding of the system and any relating concepts. For prototype UI design, the interface will be quick to access and contains a scaled blueprint of the structure with colored indicators will conveying the location of victims on the map view (similar to figure 36).

Secondary user - system and IT administrators will have more in depth access to the system. Since they are required to manage profiles related to each employee and their associated ID tag, the GUI needs to be simple and robust (see section 10.4.2 UI Mock-Ups). Actions such as adding, removing, and editing profiles or system configuration must be intuitive.

Tertiary users - The ID tags are small in design and resembles an access card so the user know how to wear the device. In the case of an emergency the employees must trigger a simple touch button to enable broadcasting of current location to the beacons. Otherwise the user must remember to keep ID tag on person while on company property.

### 10.3.2 Feedback

Feedback - There is full and continuous information about the results of actions and the current state of the product or service. After an action has been executed, it is easy to determine the new state. As the main layer of interaction between the users and the system, the GUI must provide visual indicators for any actions performed. Indicators such as confirmation messages and UI element state changes will be shown. Most importantly, during an emergency the system will be in active mode, icons and indicators on the GUI must be updated in near real time to provide constant visual feedback to the users. Furthermore, beacons and ID tags will also provide visual feedback to its users via simple LEDs indicate active or inactive system; as well as other information such as battery level or state of data transmission.

### 10.3.3 Conceptual models

Conceptual Models - The design projects all the information needed to create a good conceptual model of the system, leading to understanding and a feeling of control. The conceptual model enhances both discoverability and evaluation of results. The Akriveia Beacon system for primary users creates a conceptual model in the form of a floor plan represented through the GUI. Primary users will easily be able to relate the model to the real life layout of the building and operate accordingly. For secondary users, the system model function much like any web application that they have previously encountered allow them to easily navigate the UI and perform necessary tasks. For tertiary users the ID tag device only has one button and a few status indicating LEDs for clear interactions.

### 10.3.4 Affordances

Affordances - The proper affordances exist to make the desired actions possible. Clarity of the design creates a relationship between the look and intended use of the product which allows users to quickly understand the correct operations for the system. Some affordances of Akriveia Beacon are: Beacons and ID tags have clear LED indicators to display system status; The ID tags have bright colored button to indicate the interaction point for the user; The GUI have clear and concise labels, text, and color to simplify interactions with the user; The GUI will show location of ID tags clearly with colored indicators to identify location.

### 10.3.5 Signifiers

Signifiers - Effective use of signifiers ensures discoverability and that the feedback is well communicated and intelligible. Basic colors such as green, amber, and red are used to indicate system status such as good, okay, and bad. These three basic colors will be used throughout the product to indicate system status. LEDs on the Beacon and ID tags will use indication colors to show device status. The GUI will use initiation colors to show system status and ID tag last ping time.

### 10.3.6 Mappings

Mappings - The relationship between controls and their actions follows the principles of good mapping, enhanced as much as possible through spatial layout and temporal Contiguity. Some examples on the system include: activated tabs on the GUI are underlined with a bold color to show users the current selected tab. Pop up messages and text box will be provided when users interact with the GUI to show the UI element that they are interacting with. The ID tags and Beacons will output device status using LEDs whenever a system change occurs due to user interaction. This allows the user to receive feedback from the devices.

### 10.3.7 Constraint

Constraints - Providing physical, logical, semantic, and cultural constraints guides actions and eases interpretation. Adding constraints to the design will limit the number of actions that the user can perform with the device. One major constraint is that once the Akriveia Beacon system is activated it can not be deactivated until the situation is resolved. Only a high level system administrator will be able to access and disable the system. This constraint prevents having the system shutdown accidentally or unexpectedly. Another constraint is ensuring the system GUI is accessible on a tablet device. Mobile devices actions have to be limited in order to provide an intuitive user interaction since the only input are through a touchscreen. This constraint creates intuitive interaction between the GUI and the users.

## 10.4 Graphical Representation

### 10.4.1 UI State Diagrams

The graphical user interface will be designed for two distinct users interacting with the system. The first is primary user, the emergency first responder. This user will only need access to the map view and system status overview, their user interaction state diagram is shown below.

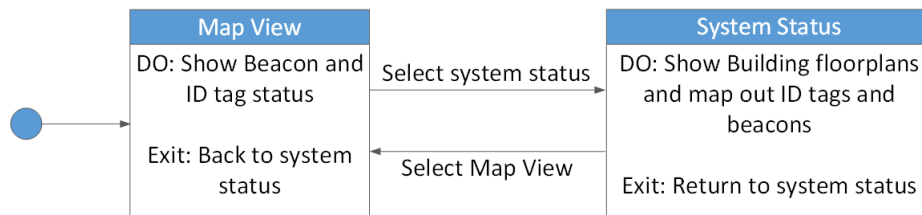


Figure 32: UI State Diagram - Primary User

The secondary user will be the administrator of the system. For secondary users the interaction is much more complex and require a more in depth level of user interaction. The secondary user interaction state diagram is shown below.

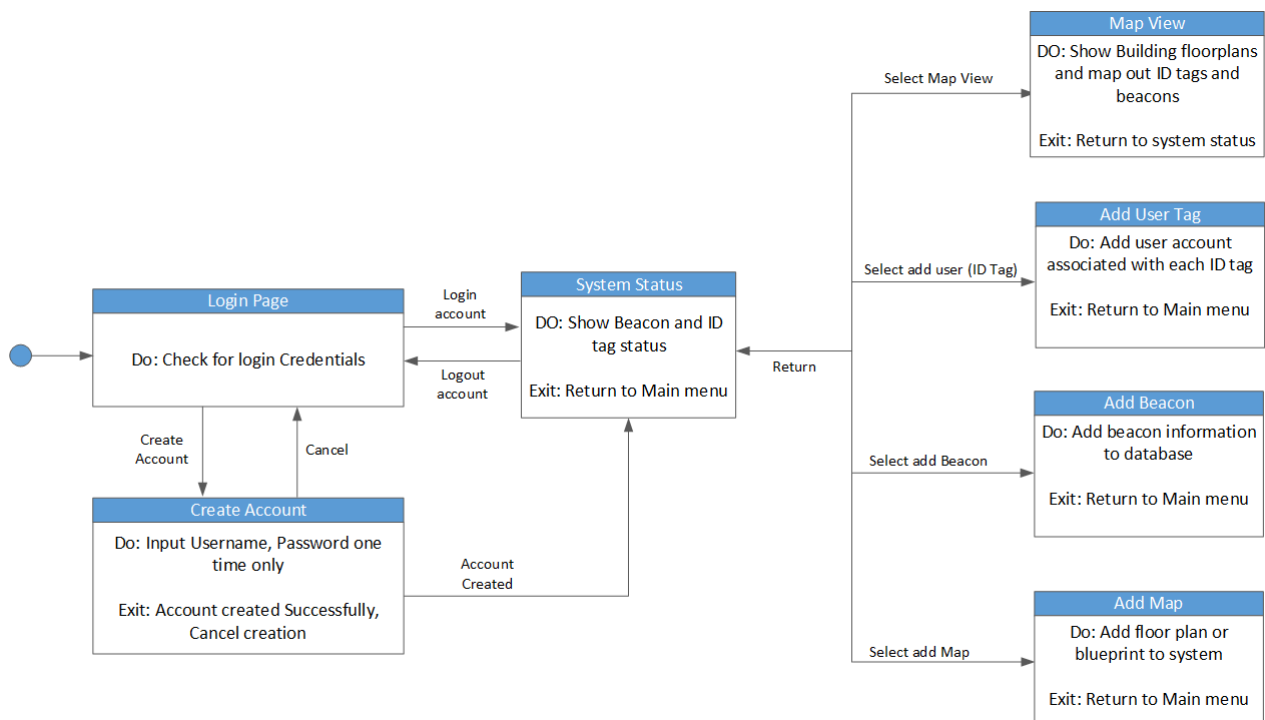


Figure 33: UI State Diagram - Secondary User

### 10.4.2 UI Mock-Ups

In the Proof of concept a simple console output displaying only the basic information such as the MAC address, RSSI and distance calculations between each beacon and id tag would be shown. Similar to the figure presented below. This UI is just to demonstrate the feasibility of the initial beacon systems.

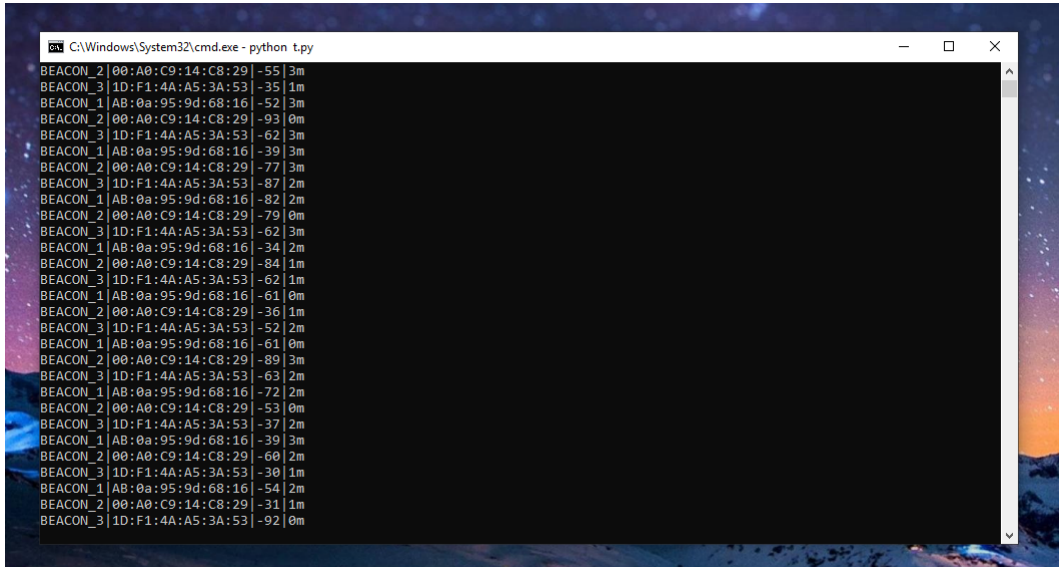


Figure 34: PoC Console UI

In the prototype phase of development, the user interface would resemble a wireframe of the final implementation. A simple box map is used to display user location in near real time similar to the figure below.

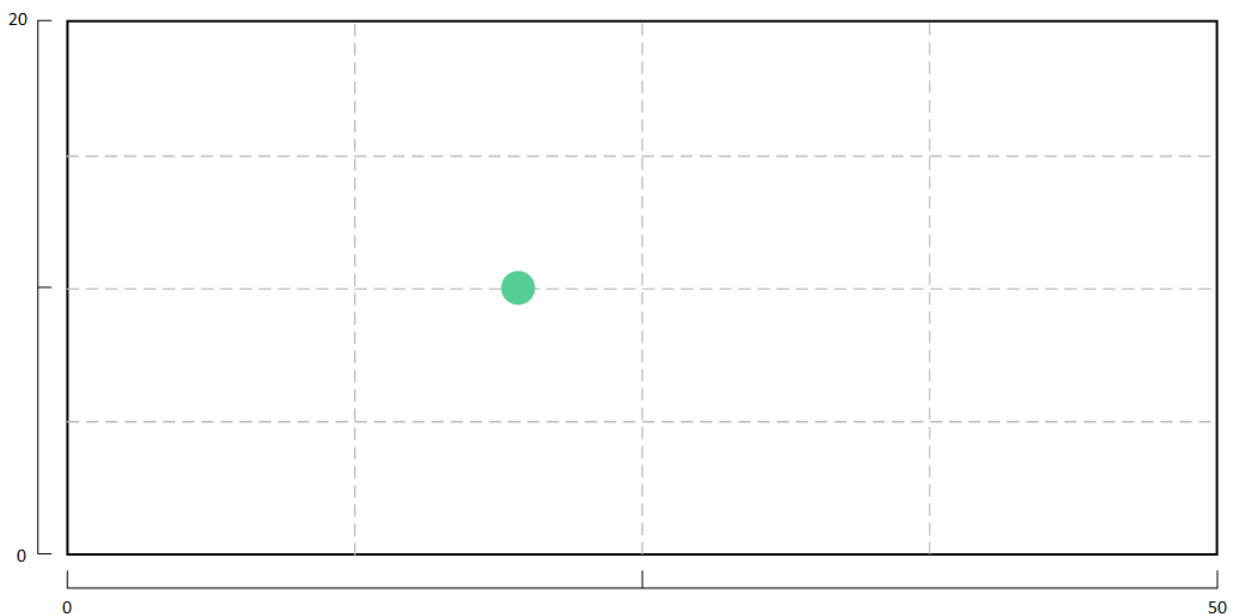


Figure 35: Prototype Box Map Layout View

In the Final phase of development the user interface would be complete, resulting in a UI that would fulfil the necessary needs of all users of the system. The primary user would have access to the map view and the system status view shown in figure 36 and 37.

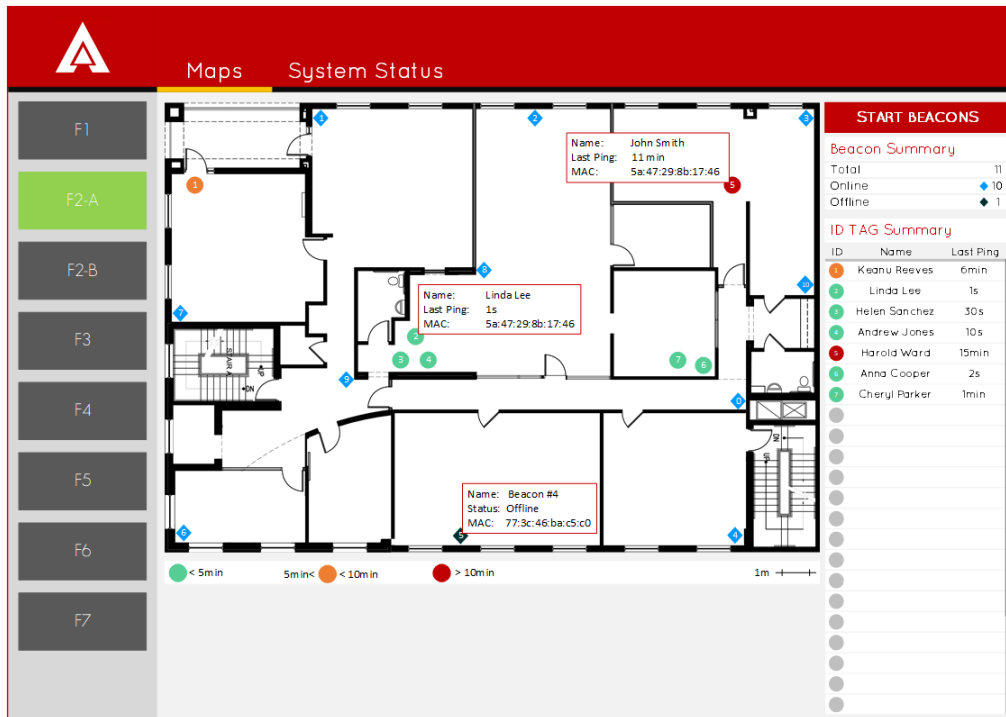


Figure 36: Primary User Map View

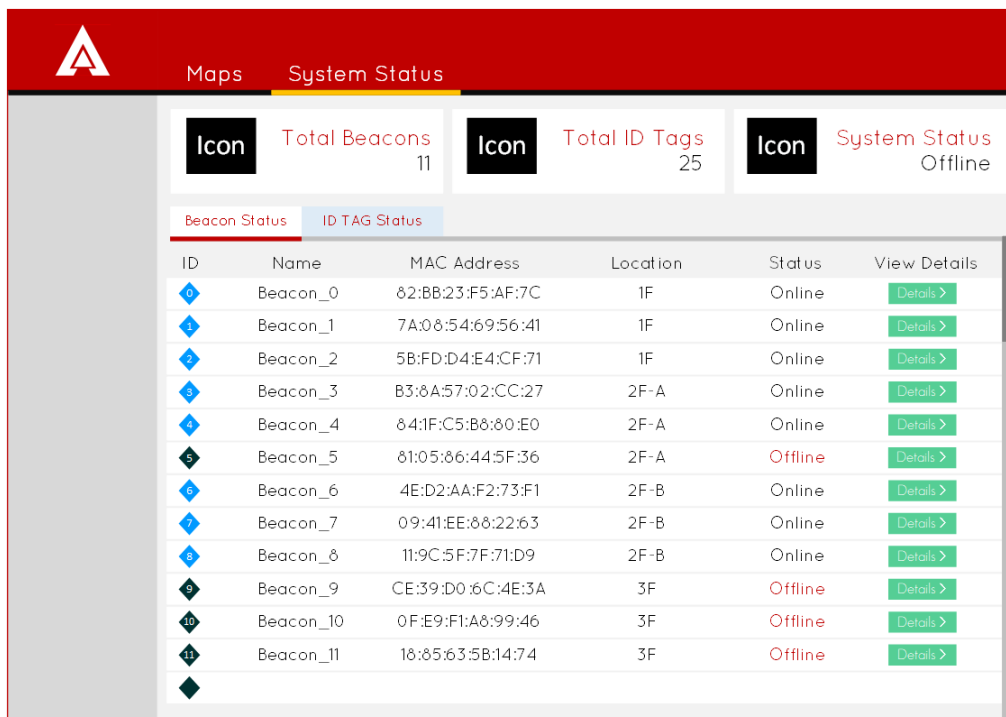


Figure 37: Primary User System Status View

The secondary user, the system administrator will be interacting with the system configurations. The administrator will be performing tasks such as adding/edit beacons, users, and maps or floor plans. The GUI for secondary users are shown in the below figures.

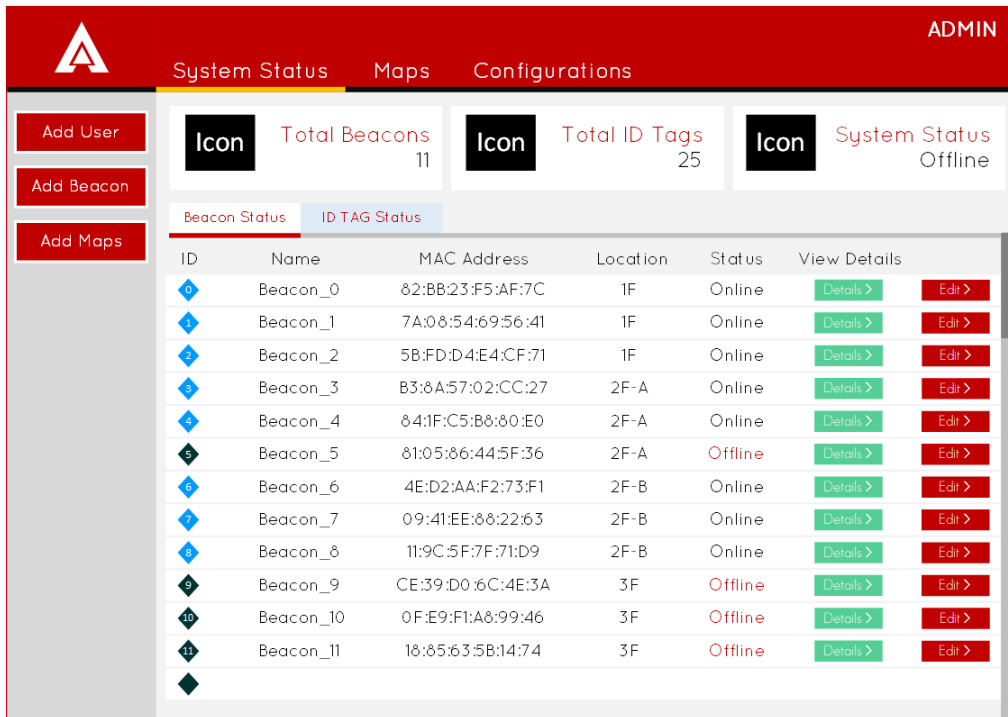


Figure 38: Secondary User System Status View

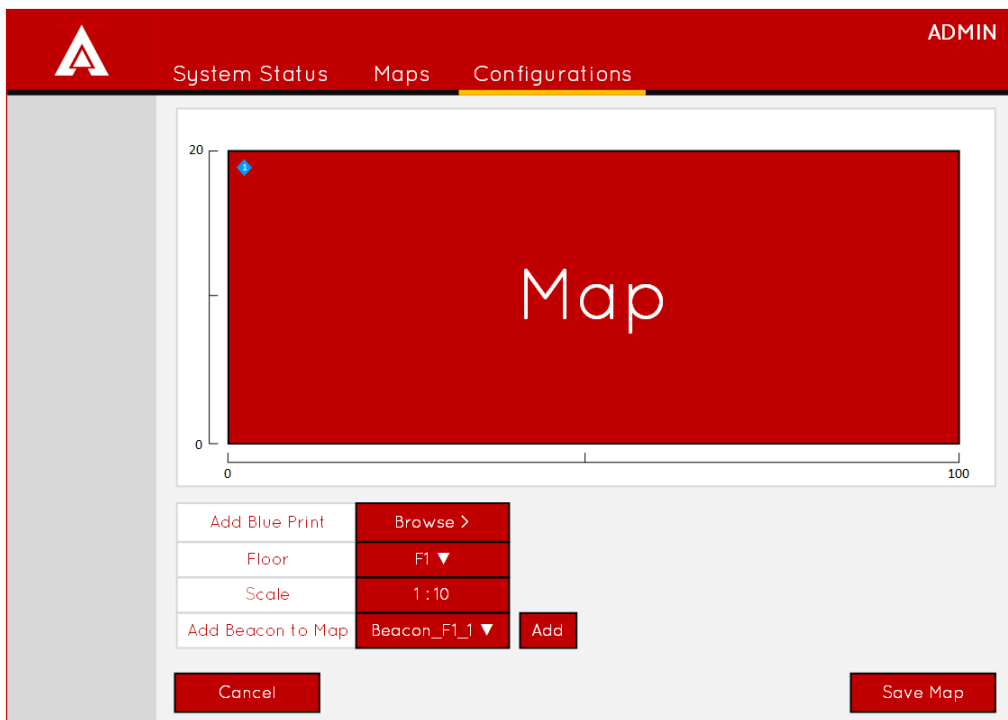



Figure 39: Add Map View



System Status

Maps

Configurations

ADMIN

Name:

Beacon\_F1\_1

Floor

F1 ▼

ID Tag MAC:

20:14:C6:A2:90:66

Notes:

Beacon Version: V1.2.0


Calibrating required

Installed near kitchen (50,10)

Cancel

Save Beacon

Figure 40: Add Beacon View




System Status

Maps

Configurations

ADMIN



Add User Image

Browse >

Name:

Tomas Jones

Age:

27

Gender:

Male ▼

Address:

8888 UNIVERSITY DR, BURNABY, BC V5A 1S6

Phone:

778-123-1234

Emergency Contact:

Tiffany Jones (778-555-8888)

ID Tag MAC:

FD:D2:CB:94:A1:F1

Notes:

Heart Condition, Asthma

Cancel

Save User

Figure 41: Add User View



## 10.5 Engineering Standards

To create an intuitive user interface for optimal user experience, the team at TRIWAVE SYSTEMS will be following several Engineering Standards throughout the development of for the Akriveia Beacon system user interface. The proposed user interface features two branch of components, hardware and software. The hardware components comprising of electronic components such as DWM1000 UWB radio modules and ESP32 micro-controllers for the device. The software components is split up between data collection from the hardware and data processing. The software also serves as a point of communication and control for the user. The two branches should always inform the user of system operations with easy to understand and highly visible status displayed through the user interface. Interface should also designed in a way that potential errors are kept to a minimum.

These engineering standards published by the IEEE, the IEC, the ISO, and the CSA Group will ensure an intuitive, safe and reliable user interface. Note that some of the standards pertain to the user interface while others pertain to general safety guidelines. The Akriveia Beacon UI design will be built by and tested against the engineering standards listed in the following table.

| Standard Code           | Description   |
|-------------------------|---|
| CSA-C22.2 NO.61508-1:17 | Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 1: General requirements [26]                       |
| IEEE 1621-2004          | User Interface Elements in Power Control of Electronic Devices Employed in Office/Consumer Environments [27]  |
| IEC TR 61997            | Guidelines for the user interface and accompanying word choices [28]  |
| ISO 20282               | Ease of operation for everyday products [29]  |
| IEC TR 61997            | Guidelines for the User Interface in Multimedia Equipment for General Purpose Use [30]  |
| IEEE P360               | Standard for wearable consumer electronic devices [31]  |
| C22.2 NO.0.23-15        | General Requirements for Battery-Powered Appliances [32]  |
| IEC 60417               | Graphical symbols for use on equipment [33]   |
| IEC 62366-1             | Guidance on Usability engineering for software [34]   |
| ISO/IEC 26907:2009      | Information technology – Telecommunications and information exchange between systems – High-rate ultra-wideband PHY and MAC standard [35]           |
| ISO/IEC 24730-62:2013   | Information technology – Real time locating systems (RTLS) – Part 62: High rate pulse repetition frequency Ultra Wide Band (UWB) air interface [36] |

Table 24: Engineering Standards

## 10.6 Analytical Usability Testing

The Akriveia beacon user interface will be tested to determine the state of its usability at various stages of development. The analytical testing phase outlines testing procedure that will be done by the engineering and design team using heuristic usability evaluations. Each evaluator will independently examine the UI and check for compliance and usability. After collecting the results the team will discuss and compile possible solutions to usability issues and generate a list of solutions. Finally, the redesign will be implemented and regression testing will be done. The analytical usability testing will take place during the prototype and final product phases of development as the user interface is well defined during these two stages. The testing procedure will follow the steps described below.

### Step 1: Usability Research Data Collection

The first step is to collect data generated by the usability test. Each evaluator will perform tasks outlined in the analytical usability testing procedure under Appendix A. From the procedures performed issues will be highlighted and documented. Each issue will have the following:

- An issue identification (ID).
- Note where it happened (screen, module, UI widget, flow, etc.).
- Task the user was engaging in.
- Concise description of the issue.

Data collected will be shown in a table similar to the table below:

| ID | Where      | Task                      | Description                                | P1 | P2 | P3 |
|----|------------|---------------------------|--|----|----|----|
| 1  | Login Page | Login with wrong Password | No error message for wrong user name input | X  | -  | -  |
| 2  | Map View   | Click on beacon icon      | Beacon info text too small                 | -  | X  | X  |

Table 25: Usability Test Results

### Step 2: Issue prioritization

Once sufficient testing has been performed by evaluators of the team, issues must be prioritized as time and resources are limited for this project. Each usability issue receives a grade of severity, influenced by factors such as:

- Task criticality: Impact on user if the task is not accomplished.
- Issue frequency: How many times an issue has occurred with various participants.
- Issue impact: How much has it impacted the user trying to accomplish the task.

### Step 3: Solution Generation

With the combined feedback and evaluations, the engineers at TRIWAVE SYSTEMS will re-evaluate possible UI designs for each usability issue that occurred during testing to determine the best and optional solution. A list of recommendations and solutions will be generated with usability test results. For each design decision several alternative solutions must be generated to include other possible ways to address the issue.

## 10.7 Empirical Usability Testing

This section details the completed empirical usability testing with users and outlines the methods of testing required for future implementations. Empirical usability testing will be carried out by the engineers at TRIWAVE SYSTEMS to systematically determine the usability of the user interface design of the Akriveia Beacon System.

During empirical usability testing, testing will be carried out in cycles with real users consists of volunteer participants. The first cycle occurs near the end of the Prototype phase and the second cycle occurs near the end of the Final Product phase. Testing will be done with two small groups of participants that are unfamiliar with project development environment. First group will be asked to perform usability test cases outlined in Appendix A. An observer will document actions and observations of the testing process as well as to keep note of average time to complete each task, the amount of errors and error rate, number of tasks completed, and perform a sequence analysis. Issues will be represented similar to the method mentions in 10.6 Analytical usability testing. With the collected data the designers will re-evaluate the user interface for possible solutions for issues. After re-design and implementation a second small group of participants will be asked to perform the same tasks as the first group

From the results generated by participants the following usability elements will be addressed throughout the two testing cycles and development stages.

- **Easability:** The familiarity and intuitiveness of the system and how comfortable the users are with the user interfaces in general.
- **Navigation:** The reliability of the navigation sequences are, how easy is it for the users to understand paths, and/or short cuts. Can the users easily retrace their steps or go back to previous states if they have made a mistake?
- **Responsiveness:** Does the users receive sufficient feedback from interacting with the system?
- **Intuitiveness:** How quickly can a new user familiarize themselves with the user interface? Whether or not the users are able to perform tasks within a certain amount of time?
- **Robustness:** Safety and reliability of the device and system are addressed by eliminating or minimizing potential error (slips and mistakes) and enabling error recovery.

By following these usability testing methods mentioned above for the Akriveia Beacon, the engineers and designers at TRIWAVES SYSTEMS can ensure a reliable and intuitive user interface will be produced to meet the needs of its end users.

## 10.8 Conclusion

The User Interface design for the proof-of-concept, prototype, and the final product of the Akrieva Beacon system developed by TRIWAVE SYSTEMS is detailed in this appendix section. Currently the conceptual framework of the Akrieva Beacon system is under development, with the primary circuitry and initial interfaces under design and early implementations. The major feature to be implemented is the wireless communication interfaces and protocols between anchor beacons and ID tags; the control and data processing unit are also in parallel development. The final project goal is to showcase the Akrieva as an accurate, reliable, modular, and simple solution in providing automated indoor location tracking with near real time and multi-tracking capabilities.

This appendix outlines a study of the user analysis, and technical analysis for interaction between the system and its intended users. As well as an overview of engineering safety and standards to ensure that the Akrieva Beacon system is safe and reliable for its designated users. Analytical usability testing and empirical usability testing will be performed to ensure that the basic system functionality, aesthetics, stability, and reliability are completely sufficient with of all planned requirements satisfied.

With the help of the Akrieva Beacon system, under the intended scenario, primary users of the system such as fire fighters and first responders will be able to locate trapped personnel with minimized search time; therefore, lowering rescue time and allowing for higher survival rate for trapped victims during the event of disasters.

## 11 Appendix References

- [23] D Norman. *The design of everyday things*. 2013. URL: New%20York:%20Basic%20Books..
- [24] Mick M Dugan. *Home*. [online]. 2019. URL: Available%20at:%20https://www.firerescuemagazine.com/articles/print/volume-3/issue-9/firefighting-operations/how-to-properly-search-a-fire-building.htmlhttps://www.firerescuemagazine.com/articles/print/volume-3/issue-9/firefighting-operations/how-to-properly-search-a-fire-building.html%20[Accessed%2021%20Jun.%202019] ..
- [25] M Rouse. *What is discoverability (in UX design)? - Definition from WhatIs.com*. 2019. URL: Available%20at:%20https://whatis.techtarget.com/definition/discoverability-in-UX-design%20[Accessed%2021%20Jun.%202019] ..
- [26] Store.csagroup.org. *CAN/CSA-C22.2 NO. 61508-1:17 — Product General Requirements - Canadian Electrical Code Part II — CSA*. 2019. URL: [online] %20Available%20at : %20https://store.csagroup.org/ccrz-ProductDetails?sku=2704154%20[Accessed%2021%20Jun.%202019] ..
- [27] Standards.ieee.org. *IEEE 1621-2004 - IEEE Standard for User Interface Elements in Power Control of Electronic Devices Employed in Office/Consumer Environments*. 2019. URL: [online] %20Available%20at : %20https://standards.ieee.org/standard/1621-2004.html%20[Accessed%2021%20Jun.%202019] ..
- [28] Webstore.iec.ch. *IEC TR 61997:2001 — IEC Webstore*. 2019. URL: Available%20at : %20https://webstore.iec.ch/publication/6269%20[Accessed%2021%20Jun.%202019] ..
- [29] ISO. *ISO 20282-1:2006*. 2019. URL: [online] %20ISO.%20Available%20at : %20https://www.iso.org/standard/34122.html%20[Accessed%2021%20Jun.%202019] ..
- [30] Shop.bsigroup.com. *PD IEC TR 61997:2001, IEC TR 61997:2001 - Guidelines for the user interface in multimedia equipment for general purpose use*. 2019. URL: [online] %20Available%20at : %20https://shop.bsigroup.com/ProductDetail/%20[Accessed%2021%20Jun.%202019] ..
- [31] Standards.ieee.org. *P360 - Standard for Wearable Consumer Electronic Devices - Overview and Architecture*. 2019. URL: [online] %20Available%20at : %20https://standards.ieee.org/project/360.html%20[Accessed%2021%20Jun.%202019] ..
- [32] Scc.ca. *CSA C22.2 No. 0.23-15 — Standards Council of Canada - Conseil canadien des normes*. 2019. URL: [online] %20Available%20at : %20https://www.scc.ca/en/standardsdb/standards/28121%20[Accessed%2021%20Jun.%202019] ..
- [33] Shop.bsigroup.com. *PD IEC TR 61997:2001, IEC TR 61997:2001 - Guidelines for the user interface in multimedia equipment for general purpose use*. 2019. URL: [online] %20Available%20at : %20https://shop.bsigroup.com/ProductDetail/%20[Accessed%2021%20Jun.%202019] ..
- [34] Blog.cm-dm.com. *IEC 62366-1 and Usability engineering for software - Software in Medical Devices, by MD101 Consulting*. 2019. URL: [online] %20Available%20at : %20https://blog.cm-dm.com/post/2018/07/06/IEC-62366-1-and-Usability-engineering-for-software%20[Accessed%2021%20Jun.%202019] ..
- [35] ISO/IEC. *ISO/IEC 26907:2009*. 2019. URL: [online] %20ISO.%20Available%20at : %20https://www.iso.org/standard/53426.html%20[Accessed%2021%20Jun.%202019] ..
- [36] ISO/IEC. *ISO/IEC 24730-62:2013*. 2019. URL: [online] %20ISO.%20Available%20at : %20https://www.iso.org/standard/60379.html%20[Accessed%2021%20Jun.%202019] ..