NUI Galway
OÉ Gaillimh

Asset location using low-cost beacons, smart roaming devices and cloud computing

Thesis DRAFT

Samuel Nixon

B.E. Electronic & Computer Engineering Project Report EE443

April 2019

Academic Supervisor: Dr. Edward Jones

Abstract

With the increase in shipping year on year and the continued decrease in industry margins, technology is being looked to for the answer to increasing operational efficiency. Technology provides industry the means to record and process data that has never before been available, with asset location being some of the most valuable data. Low-cost sensor hardware has never been more accessible, and with the continued popularization of cloud computing, deducing meaning from this data in a scalable way has never been easier. Using a combination of low-cost beacons, largely in-situ smart roaming hardware and cloud computing, this project presents a solution to the challenge of outdoor asset tracking.

This project implements a proof-of-concept cloud computing model, demonstrates how a location-aware smart-device can 'sniff' nearby devices and upload readings in real-time to the cloud and demonstrates a cloud-based localization algorithm using simulation software.

Technologies of note are WiFi, Bluetooth Low Energy, GPS, Cloud Services, AWS, Raspberry Pi, Python, Node.js.

Declaration of Originality

I have read and understood the NUI Galway Code of Practice.

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institution or tertiary education.

Information derived from the published or unpublished work of others has been acknowledged in the text and a list of references is given.

Signature: _____

Date: _____

Acknowledgements

I would like to thank the following people for their invaluable input to this project; Dr. Edward Jones, Martin Burke, Myles Meehan, Kenn Humborg and Simon Bradish.

Table of Contents

List of Symbols

List of Figures

List of Tables

Glossary

1 Introduction

The aim of this project is to design and test a fully scalable, low cost system to provide location tracking of assets, using a combination of intelligent hardware, simple hardware and cloud services.

This combination of different 'levels' of hardware allows a flexible on-the-ground solution that can be adjusted to suit real-life demands, with the ratio of smart to simple devices providing the accuracy desired for each application. Cloud Services used are provided by Amazon Web Services, a popular cloud services provider. In order to provide seamless on-demand scaling, the architecture of the cloud services is of particular interest. Such a system would find many applications in industry, particularly in transport and manufacturing. This project focuses on the application of such a system in a shipping yard, where assets consist of trailers, containers, forklifts, trucks and other machinery.

In modern times, cloud computing allows organisations to quickly and effectively build IT infrastructure and services. Utilising cloud services to provide a service that can automatically scale to handle any demands seemed interesting. Access to intelligent hardware with current connectivity capabilities has never been easier, potentially providing a means for a low-cost solution to asset tracking in a domain. This project seeks to explore the effectiveness of combining some of these intelligent devices with simpler, lower cost beacons to provide location information for objects, without the need to attach expensive data and GPS capable modules to every asset.

1.1 Background

Asset location has always been useful data to have, but companies generally have to rely on vague data such as last known city/distribution center and next city/distribution center. Fine grained data such as movement of an asset within a yard or site is normally unavailable. Solutions providing location data of Tractor/Trailer assets are common, but are tailored to long haul distances, rather than the real-time tracking of asset movement in a yard. With the increased ubiquity of hardware providing GPS monitoring, tracking of an expensive object such as a forklift within a yard has become possible, but what if the location of the loads the

forklift was moving in the yard was also desired? Attaching GPS and network capable hardware to load beds would be financially excessive, particularly compared to a solution where a forklift was able to report nearby assets, with location sensitivity.

Transportation companies are increasingly turning to technology to assist their business. Some countries have passed legislation forcing the use of technology to provide overview and compliance with existing legislation, such as driving working hours. In the US, use of an ELD (Electronic Logging Device) by professional drivers will be mandatory after 16 December 2019 [1]. These devices must GPS capabilities, and almost always provide internet access and Bluetooth. With hardware such as this already a requirement, there is opportunity to utilize this pre-existing hardware to provide further location services e.g. if a Tractor/Trailer arrives in a yard, it can scan for nearby Bluetooth devices during yard movements and loading/unloading, thereby providing data points to allow for context aware tracking of other assets.

This use of existing devices, combined with simple, off the shelf beacons on assets would provide a low-cost solution. Tractor/trailer readings might not be frequent enough due to the sporadic nature of their being on site, so attaching some intelligent device to a moving asset permanently located onsite, such as a forklift, could provide good readings. Site management workers might also be furnished with a tablet with which to monitor orders, read emails etc., which could also be loaded with an application that could scan for nearby devices using the tablet's built in Bluetooth and upload location data from the tablet's GPS radio to some central processing center.

Access to fine-grained location data would provide the data to optimize operations with asset utilization, load-balancing of assets and improved security of assets. Benefits of a RTLS (Real-Time Location System)

- **Reduced Downtime.** Distance travelled by an asset can be tracked, allowing maintenance intervals to be adjusted.
- **Improved Security.** With real-time location data, the disappearance of an asset can be tracked

- **Safety.** With RTLS, location of assets relative to each other and employees can be monitored, and with enough data the trajectory of assets predicted.
- **Improved Vehicle utilistation.** With accurate vehicle monitoring, operational load can be balanced across a fleet, and empty runs avoided.

Here, an asset can refer to any of the following; a vehicle, a trailer, machinery, a container, cargo, workers.

1.2 Problem Statement

Location data must be acquired from devices both with on-board location sensitivity and no location sensitivity. This data must be uploaded to cloud services which must use this data to reason the actual location of all objects – those with location knowledge and those without. Cloud services must provided a means to query location estimations. Cloud services must be designed to scale from a small number of requests to a large number of requests automatically.

1.3 Societal Impact

The US introduced a mandate that most commercial vehicles must have an Electronic Logging Device (ELD) in use from 2017. The EU mandates use of digital tachographs. Often this functionality is implemented using a device equipped with GPS, a network connection, and Bluetooth. These devices are small computers that connect to sensors on equipment and monitor equipment status and provide functionality such as routing, messaging, entertainment and others. That is to say, the hardware for implementation of a smart device is already in place.

Mobile devices held by staff can potentially be used as smart devices, with the installation of an app on the mobile device.

The primary benefit of asset tracking in a commercial setting is having traceability of asset. From traceability we can derive improvements in productivity, security, accountability, safety and preventative maintenance and repair.

Improvements in productivity can be acquired by monitoring the utilisation of assets and using this data to make a decision on how many assets are required to complete the tasks needed, and if any are surplus to requirements. Likewise, it may be found that the process could be improved with additional assets. Improvements in productivity can also be realized by reducing the time lost by looking for a certain asset e.g. if a trailer must be delivered to a different location, knowing the current location of the trailer in the current yard can reduce the time wasted by the driver looking for the right trailer. This reduction in time can decrease the likelihood of delays further in the process and increase utilisation of the driver's HOS (Hours of Service) or relevant permissible working hours, allowing the driver to spend these hours adding value to the organisation. Back office productivity can also be increased by reducing any effort needed to point the driver to the location of the trailer.

Security of assets can be improved through constant awareness of assets location. Rather than relying on staff to notice the absence of an asset, asset tracking software can highlight the absence of an asset where an asset is expected to be. The morning after a yard has experienced a security breach, staff have instant access to an account of all assets still in the yard, and the last known location of any assets that have gone missing. In situations where multiple teams share assets such as forklifts or machinery, staff can forget to register the use of an asset and even to return the asset. Where multiple sites within an organization share assets, this can cause lost time and increase the cost of the operation. By automatically tracking the location of assets and adding geofences to sites or even within sites, back office personnel have access to the location of an asset, thus eliminating any doubt as to the asset's location.

Improvements in accountability can be realized through the ability to pinpoint the location of an asset throughout a process of, for example, fulfillment. Goods such as temperature sensitive foods must be kept within strict temperature requirements. The Food Safety Authority of Ireland recommends that meat remains at the same temperature at all times during transport [2]. Excess loading and unloading times at fulfillment centers can be a source of heat contamination for such goods, where they are moving from one temperature controlled environment to another. The customer can be provided with a log of

loading/unloading times on delivery, helping to assert the condition of the goods. Excess loading and unloading times can cause other problems such as delays further on in the process or delays for trailers needing to be loaded at the same dock. Excess stay durations at a dock can be highlighted to back office staff for action to be taken.

Real-time asset tracking can improve safety on sites where there is a high safety requirement such as on construction sites. On such sites, there is a high volume of workers and high traffic of machinery on site. Operators of site machinery often have many blind spots around their machine and are reliant on spotters to assist them in their movements. If the location of the machine and the location of workers on site was known by a solution in real-time, the machinery operator could be presented with a display of the locations of nearby workers and proximity alerts. This could be useful to a crane operator that does not always have complete visibility of the area surrounding the load they are delivering to a location on site. Information on the proximity of workers to the load could provide useful intelligence.

For a shipping yard operator, the primary motivations for implementing an asset tracking system are financial and safety. Having a dry van trailer out of use for a year can result in a revenue loss of $84,000 [3], which equates to a revenue loss $230 per day of inactivity. Increasing the utilization of a fleet of 200 dry van trailers by 1% could therefore result in a revenue increase of $430, 000. The Federal Motor Carrier Safety Administration (FMCSA) produced a report on driver detention [4] that claims detention of drivers at facilities is associated with reductions in annual earnings of between $1.1 billion and $1.3 billion. The report also claims a 15 minute increase in average dwell time (time spent by a truck at a facility) increases the expected crash rate by 6.2%. The report specifies driver detention as ‘time at shipping and receiving facilities beyond that legitimately needed for cargo loading and unloading’. Clearly financial and safety gains can be made by industry simply by increasing efficiency of driver turn-around at facilities. An organization on it’s own can also utilize reduced detention to drive efficiency and reduce costs. As industry margins get tighter, and with large disrupters like Amazon joining the market, any areas in which an operator can increase efficiency can greatly benefit their profit margin.

1.4 Project Objectives

The objective of this project is to design and build a system capable of providing the following features:

- Network and GPS enabled devices ('smart') scanning for beacons ('dumb') in range
- Scalable cloud services to process data uploaded from 'smart' devices and estimate location of all devices to within 1 meter.
- An API to allow querying of location information for a given device for reporting purposes
- Simulation to test system with simulated devices

1.5 Main Project Components

The overall desired system is shown in Fig. 1.1 below. The system must be capable of utilizing both real and simulated data and providing reporting capabilities on this data. Basic reporting capabilities should allow querying of a device's current location, which the system will determine based on historical locations for this device with some level of accuracy.

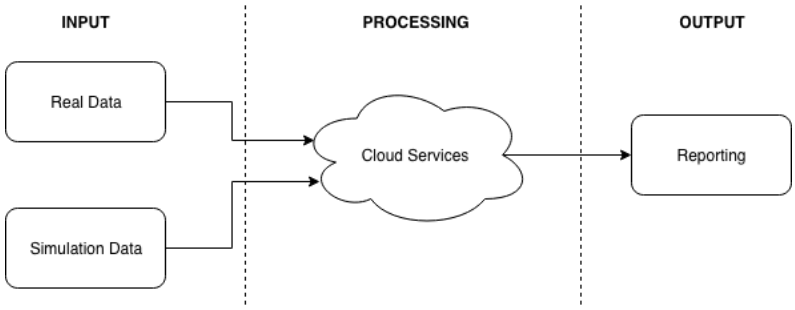

Fig. 1.1. Overall Project

The overall system, which can be broken down into three main components;

- Hardware
- Cloud Services
- Simulation

This thesis is divided into chapters arranged in the following order; Background Research, Design and Implementation, Experiments and Results, Discussion and Conclusions.

Background Research explores the background of the project, including state of the art, societal impact and component level research. Design and Implementation details the solution developed. Experiments and Results reports experimentation completed as part of the project. Discussion and Conclusions assesses the work completed and provides reflections as well as suggestions for future work.

2 Background Research

This chapter provides a detailed examination of solutions in use in industry as well as solutions from published works in State of The Art. The impact the project could have on society is explored in Societal Impact. The main components of the project are divided into the following sections; Hardware Research, Cloud Services Research, Localization Algorithm and Simulation Research. Each section researches a particular component of the project.

2.1 State of The Art

GPS based asset tracking solutions are fairly common place in industry. Many providers such as Orbcomm, WirelessLinks, Verizon Connect offer asset tracking solutions. These solutions normally leverage a GPS tracker per asset approach, whereby a GPS tracker is installed on each asset. This approach works well and is necessary when a fleet of assets is primarily distributed, but for an application where the asset is expected to remain within a yard or site a GPS module per asset can be unnecessarily when there are other available options.

Sanghyun Son et al. [5] introduce an approach for asset tracking in a container yard by using both static and moving readers to estimate the location of moving tags. In order to track the location of yard tractors, readers were placed on both light towers and mobile cranes. The wireless communication used was based on IEEE 802.15.4a. The approach yielded an estimated tracking error of approximately 6.3m, which they found to be close to the accuracy achievable using GPS tracking. Using the known location of beacons, the location of moving tags can be computed based on readings from multiple beacons as long as line of sight is available. This approach provides an interesting solution to asset tracking in a yard application, however, this approach requires the installation of radio infrastructure in-yard, which can be prohibitively expensive. Line of sight was required to communicate with the tags, which is why readers were placed on cranes that had permanent line of sight with yard tractors.

Orbcomm [6] offer an industrial solution for outdoor asset tracking that consists of attaching location aware modules to every asset a user wants to track. Each module is equipped with a cellular connection that it uses to communicate with cloud hosted servers that provide back-

office administrators with control over the system. This approach allows every asset to be completely independent and self-sufficient in terms of reporting it's location. This approach allows a device to update it's location independent of it's proximity to a yard, the only requirements are that the device has a GPS fix and a data connection. As these devices have GPS modules onboard, they are inherently power hungry. Orbcomm have solved the issue of supplying power to these devices by equipping them with solar panels, allowing reported maintenance-free operation for up-to 10 years.

Other solutions use static beacons and moving tags, with BLE radio. This approach can utilize Bluetooth radios already built into many consumer devices that workers might be carrying, such as tablets for taking inventory and such. This approach works on reasoning from proximity readings the location of devices. This approach requires installation of beacons within the yard at regular intervals.

Using beacons installed at regular intervals adds to the time and cost of setting up such a system, as well as adding another component that must be maintained.

A hybrid approach such as the one suggested in this project has not to the author's knowledge been implemented in industry, where existing devices commonly found in trucks and machinery can be piggybacked to locate low-cost beacons placed on other important assets. This solution has a low set-up cost and time as only low-cost beacons need to be attached to assets and software added to devices already in use. This solution also adds minimal maintenance to the maintenance costs already associated with maintaining the 'smart' devices. This use of adding functionality to hardware already in use is a novel one.

2.2 Hardware Research

This section of research explores hardware possibilities for the project. First, hardware requirements are laid out, then each piece of hardware is tackled individually.

2.2.1 Requirements

To prove this concept could work, two types of devices are needed, a smart device and a dumb device.

The smart device must have some onboard processing and memory and is assumed to be plugged into a permanent power supply, so battery life is not too important. It is envisioned that smart devices will live on machinery such as forklifts or tractor units i.e. units that can supply power to the device whenever the machine is moving. These smart devices must have some method of communicating to the internet onboard, this could possibly be over WiFi or Cellular Network. The smart device must also have knowledge of it's own geographic location. These smart devices are being modelled after ELD and AOBRD devices currently required by US and EU law on tractor units.

The dumb device must have knowledge of it's own identity, and the ability to broadcast this identity to a smart device. This onboard radio will need further research. It is envisioned that these dumb devices would be affixed to objects such as trailers, which have no guarantee of having access to power for extended durations. These dumb devices must also be cheap and off-the shelf units, to help make the whole solution cost-effective. Dumb devices should be easily affixable to an important object and should require no maintenance. Their batteries should be capable of lasting for months at a time. These dumb devices should be almost disposable, and after their batteries have died, they can be replaced.

2.2.2 Research and Analysis of Radio Frequency Communication

There are a wide variety of technologies used for wireless communication available commercially. Some of these technologies are very similar in nature but have very different use cases. These technologies have various ranges, complexities, costs and accuracies.

As this project is exploring the use of low-cost beacons that require little installation and maintenance, the beacons must be reliable and simple. To allow for quick installation, a beacon should be a standalone unit capable of operating without the need for external power. As such it will need to be battery powered, and battery-life will be a primary consideration. Generally, range and battery life are closely related, as the larger the range of

a devices the more power it will consume to reach this range, so battery capacity along with power consumption will dictate battery life.

The mobile device will likely need to be connected to external power and can be assumed to have either a constant source of power (as in a machine when the machine is running), or some facility to charge the device. The mobile devices will need GPS radio in order to get a reading for its own location. It will also need to have an onboard module capable of communicating with the asset beacons. The mobile device must have a data connection, either in the form of a SIM card based Network connection or WiFi.

2.2.2.1 WiFi

WiFi uses radio waves to exchange data between devices. Based on the IEEE 802.11 [7] standards WiFi is commonly found in computers and mobile devices. It is primarily used to connect many devices to the same network, typically to allow access to the World Wide Web. The term WiFi encompasses multiple standards, all based on the 802.11 specification. They vary mostly in speed, range and frequency use. Many modern WiFi capable modules are capable of using IEEE 802.11b/g/n at speeds of 11/54/600Mbps respectively and operate at either 2.4GHz or 5GHz. Typically, these devices connect to a wireless access point within 100m.

2.2.2.2 RFID

Radio-frequency identification (RFID) uses a combination of readers and tags to identify objects. Tags are attached to objects to be identified and can be active or passive. Passive tags do not require a battery as they use some of the energy broadcast from the reader to send back a signal with their identification. As such, the greater the range desired from a passive system, the more powerful the reader needs to be in order to get enough energy to the tag, particularly if the location of the tag relative to the reader is unknown and the reader must broadcast over a wide area. Passive RFID is typically used where assets must pass through choke points, and the tag will not be far from the reader. Passive RFID systems typically have a range of 12m or less.

Active RFID tags use on board power to power their return signal. An RFID system using active RFID tags can be set up with a lower powered reader as the tag does not rely on drawing power from the reader. Active RFID systems offer a range of 100m or more.

2.2.2.3 Bluetooth

Bluetooth uses radio waves from 2.4 GHz to 2.485 GHz to transmit data [8]. Bluetooth is commonly found in consumer devices and is typically used for pairing devices together over a short range – up to 100m outdoors. Bluetooth was designer with the purpose of replacing data cables, for example streaming music from a mobile device to a speaker. Bluetooth allows for 2-way communication between devices.

Bluetooth Low Energy (BLE) operates on the same frequencies as Bluetooth and offers similar range but with significantly less power draw. BLE is only compatible with Bluetooth version 4.0 and onwards, as the same hardware can be used for both technologies. BLE is intended for use in the IoT area.

2.2.2.4 Ultra-Wideband

Ultra-Wideband (UWB) is a wireless technology designed to transmit data over a short range. UWB uses multiple frequency bands which reduces susceptibility to noise. UWB operates in the range between 3.1 GHz and 10.6 GHz. Because UWB uses such a wide frequency band to transmit data, it can transmit through objects more reliably (such as doors) than other radio frequencies. Thus, distance between devices can reportedly be measured within 10cm

2.2.3 GPS Research

The Global Positioning System (GPS) is a navigation system based on satellites that can provide location and time information to a GPS receiver. GPS is operated by the United States (US) government but is not the only geolocation solution available. Other states have completed or are in the process of completing similar solutions to GPS, all of which are classified under the Global Navigation Satellite System (GNSS). Modern GPS receivers are compatible with both GPS and Global Orbiting Navigation Satellite System (GLONASS), the Russian state owned GNSS solution. Some receivers are also compatible with the BeiDou

Navigation Satellite System (BDS), a GNSS owned by the People's Republic of China. GPS requires the receiver to have a view of the sky and is thus best suited for use in outdoor applications. In order to provide 2D location, a GPS receiver must communicate with a minimum of 3 satellites. Communication with 4 or more satellites allows location in 3D i.e. altitude to be calculated. Location accuracy is increased as the number of satellites available is increased. Sources of error include atmospheric interference, signal reflection due to tall buildings or other obstacles and inaccuracies within the receiver's clock or a satellites reported position. GPS is commonly used in mobile phones for navigation, with accuracy said to be within 4.9m [9].

Some microprocessors are available with onboard GPS. Standalone modules designed with the singular purpose of providing GPS are widely available.

2.2.4 Smart Device Research

Smart devices require some basic processing power, along with the capability of operating the needed radios and consuming fairly low power. In order to expedite development of the solution, smart devices that can be built from as off-the-shelf units as possible are desired. The more on-board radios the device has the better, as this will reduce the time needed to configure external modules. These devices must be programmable and have a small footprint. As these devices will not be used in production ..

Some options of off-the-shelf units include those from Raspberry Pi and Arduino.

2.2.5 Dumb Device Research

Dumb devices are simply beacons capable of announcing their presence and identity to some requester. They are required to be low power and totally self contained, with no knowledge other than their own identity. These beacons should be able to communicate over the chosen radio frequency. Some possibilities include beacons using iBeacon or Eddystone technology. The devices should be capable of communicating frequently.

2.3 Cloud Services Research

More and more what would have previously been built on one server/machine is being divided into components that do very specific things on different machines. This allows greater flexibility, redundancy and scalability. Cloud computing is offered in many forms from many different vendors. This section seeks to provide some clarity as to what exactly is on offer.

2.3.1 Requirements

Cloud services must include some database for data persistence, scalable computing and some method of load balancing across resources. For the purposes of this project, it should be possible to implement cloud services using the available free tiers offered from different vendors. The cloud services should all be implemented using a single vendor, so as to reduce the work needed to implement the solutions. Implementing services on a single cloud provider should also reduce latency within the cloud services and thus improve application performance. The location of the cloud services will play an important role in the latency of the application, with cloud services implemented in a location physically closer to the smart device or user requesting services performing faster than those located further away, such as on another continent. The chosen cloud services should be capable of scaling from testing to production level usage without any major changes to the implementation.

The cloud services must be capable of providing a Representational State Transfer (REST) Application Programming Interface (API) for access by both smart devices and users/terminals/3d party services. REST defines an architectural style for transfer of state [10] and is largely used to build web services on Hypertext Transfer Protocol (HTTP) methods e.g. GET, PUT, POST, DELETE etc. This API will form the communication link between the outside world and the computation layer in the cloud services. This API must provide resources that allow devices in the field to deposit and retrieve information, as well as providing resources for administrative back office software to perform queries and administration e.g. addition of a new device.

2.3.2 Vendors

Cloud computing solutions are offered by multiple vendors, with the most popular being AWS (Amazon Web Services), Google Cloud Platform, IBM Cloud and Microsoft Azure. Each of these vendors have different terminology and different offerings. It should be noted that the author has good working experience with AWS. In terms of market share, AWS is the leading cloud offering. The author also has some experience working with Google Cloud Platform. All of these vendors offer cloud services that can be deployed in various locations such as the US and Europe.

2.4 Localization Algorithm

Location awareness is a huge part of this project. We desire the location of all devices in the physical domain, but only some portion of these devices (smart devices) are inherently location aware i.e. have access to onboard GPS radio. The rest of the devices in the field are dumb devices, with no location awareness. The location of these dumb devices must be reasoned from the locations smart devices report 'seeing' the dumb devices. Multiple readings from smart devices reporting 'seeing' a dumb device at some location should allow reasoning of the location of the dumb device. Smart devices must also be time aware, and report the timestamp of their reading, so that readings can be viewed with respect to time. A reading from a smart device reporting the seeing a dumb device at location a at a timestamp more recent than another reading from the smart device reporting the dumb device at location b should be weighted heavier.

Smart devices will also be able to report the locations they have seen other smart devices at, which should allow a more accurate representation of each device and possibly help account for hardware bias and noisy readings from devices.

Smart devices will be uploading readings to some central computing resource, which will aggregate readings from multiple devices and run some location estimation algorithm on the uploaded readings to determine an estimation of the location of a device at some time. The algorithm will have to deal with bursty data, where a smart device for example doesn't have a network connection for some amount of time and uploads multiple readings in rapid

succession, rather than in real-time. The algorithm must also be capable of dealing with the movement of both smart and dumb devices. The change of location of a smart device should be obvious due to it's own location awareness, but the algorithm must also account for the movement of dumb devices. The algorithm must be able to discern between noisy data and a dumb device beginning to move, or being moved a small distance, say a few meters.

Range-based information such as received signal strength indicator (RSSI) could potentially be used to give some indication of the distance to a physical object but this approach has various problems. RSSI is a measure of the power of the received signal. The RSSI value is highly dependent on factors such as the chipset being used and environmental differences. The value reported can vary between chipsets, e.g. chipset A reporting values in the range 1-100 and chipset B reporting values in the range 0-127. As such these values should only be used to indicate if the signal is getting stronger or weaker i.e. is the object getting closer or farther away. Received power level is another possible Bluetooth signal parameter that could be used to indicate distance between objects. Again this may only be useful in indicating the direction of movement – if the average or modal values are increasing or decreasing.

The estimated location of the device should be reported in two-dimensional space i.e. latitude and longitude. This could be extended to three-dimensional space using the altitude reported by the smart device, a value that is available when 4 or more satellites can be reached.

Possible options for an algorithm are algorithms based on either a Kalman filter or a particle filter.

2.5 Simulation Research

It is expected that due to the nature of solution needed, manual testing with movement physical devices is not feasible. Testing the system will provide verification of system performance in real life. As such, testing of must be carried out using simulation software. Testing of the constructed system requires a lot of data to be supplied to the system and queried from the system. The following attributes are required to be tested;

- **Load handling of the Cloud Services.** Can the system handle bursty traffic? How does the system respond to peak traffic?
- **Accuracy of location predictions.** How accurately can the system predict the location of an asset, given historical information? How does this accuracy change with noisy data, bursty data, varying ratios of devices?

In order to simulate locations of assets effectively, simulation software will be used. There is simulation software available that will allow modelling of a site/yard and will allow outputting of the resulting data. This simulation output will have to be converted to a format that the system expects, fed into the system and the reporting from the system analysed. The following Fig. 2.1 shows expected simulation architecture. This form of simulation is known as symbiotic simulation.

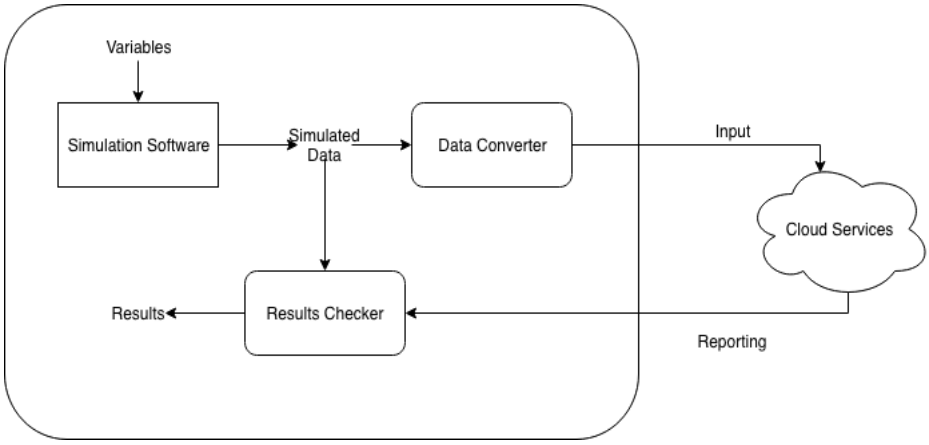

Fig. 2.1. Proposed simulation architecture.

Wrapping the simulation software in a custom piece of software that will control the simulation software, convert the output data the format expected by Cloud Services and compare the results from Cloud Services against the simulated results could provide the desired output. This solution would allow testing to be automated and would provide a platform for more extensive testing, such as testing of accuracy over a time duration of multiple days of data.

Research as to possibilities of using existing simulation software tools has shown that there are multiple possibilities for simulation software. However, this simulation solution will have to meet the following requirements;

- **Programmatic Control.** In order to run multiple simulations, the simulation software must be controllable programmatically to allow adjusting of variables such as data noise, frequency of asset movement, speed of asset movement, frequency of asset scanning etc.
- **GPS Location Simulation.** The software must allow modelling of some yard/site and assigning of geographic information system (GIS) data to the yard/site area.
- **Useable Output.** The output from such simulation software must be in some format so as to allow feeding into the Cloud Services.
- **Validation of Results.** The simulation solution must be capable of outputting the performance of cloud service in terms of localization accuracy.

With the above requirements in mind, it appears that simulation software must be Agent Based, i.e. the software must be centered around the modelling of Agents. Attributes can be assigned to Agents. The software must also have GIS (Geographic Information System) capabilities, so as to allow latitude and longitude simulation. The following simulation software packages have been found that meet the above requirements;

2.5.1 GAMA

GAMA is an open source simulation development environment for building spatially explicit agent-based simulations [11]. GAMA is based on Java and allows instantiation of agents from datasets including GIS data.

2.5.2 AnyLogic

AnyLogic is a company providing simulation software for use in multiple industries, most notably transportation [12]. AnyLogic provides software for a free trial and is Java based, allowing custom integrations in Java to be written. The software allows building of a custom area and complex agent behavior classification.

2.5.3 Brinkhoff Generator and variants

This simulation software was originally built by Thomas Brinkhoff to simulate behavior of moving objects [13]. This software has been extended by more recent efforts, including Hermoupolis, a trajectory generator [14]. Hermoupolis has been released as an open source project, with source code and datasets available on request [15].

Interestingly, Java seems to be a popular language to implement simulation software in. Where the simulation software allows manipulation of the code underlying the model, the simulation software could possibly communicate directly with cloud services, without the need for external software to act as a broker. Such an architecture can be seen in Fig. 2.2. Such an implementation would reduce the amount of moving parts in the solution and reduce solution complexity. This type of architecture would require that the simulation software have network access, and be capable of communicating with the cloud services. If the underlying code can be manipulated, this should not be a problem as common Java/other language libraries are available to provide this functionality.

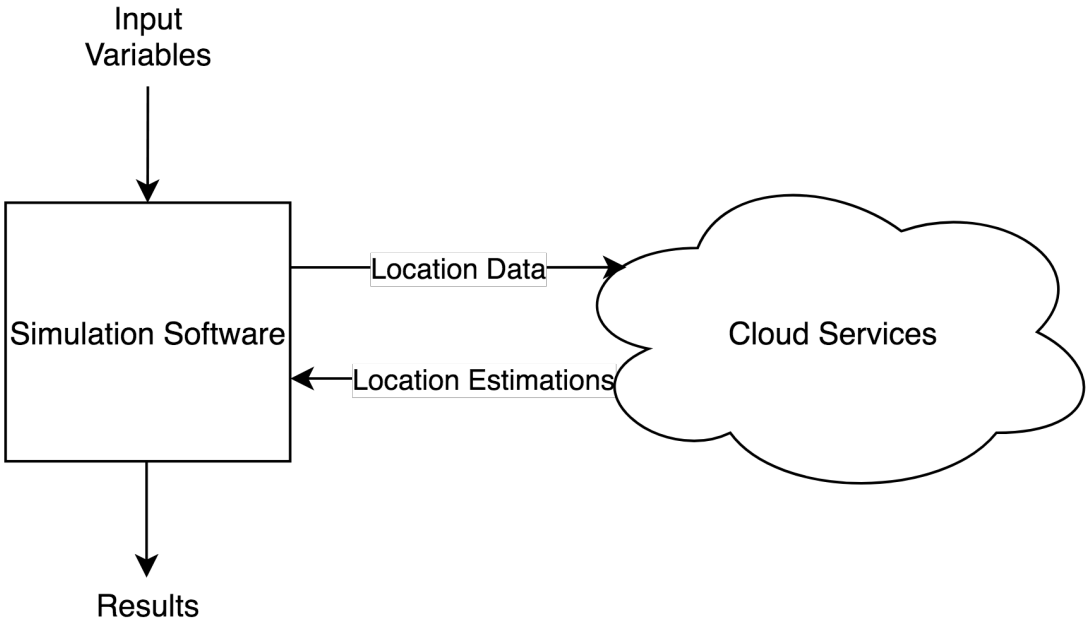

Fig. 2.2. Simulation Architecture

In summary this chapter focused on the research behind the project. State of the Art solutions from industry and literature were examined and options for the components of the project were broken down.

3 Design and Implementation

This chapter will focus on the design of the chosen solution with a top-down approach. As the solution is made up of separate components, a top-down approach seems appropriate. First an overall view of the system will be given in System Architecture. The following sections will then divide the system into components and examine their design and implementation.

3.1 System Architecture

As mentioned previously, the solution consists of two main components; Hardware, Cloud Services. Overall system architecture is shown in Fig. 3.1.

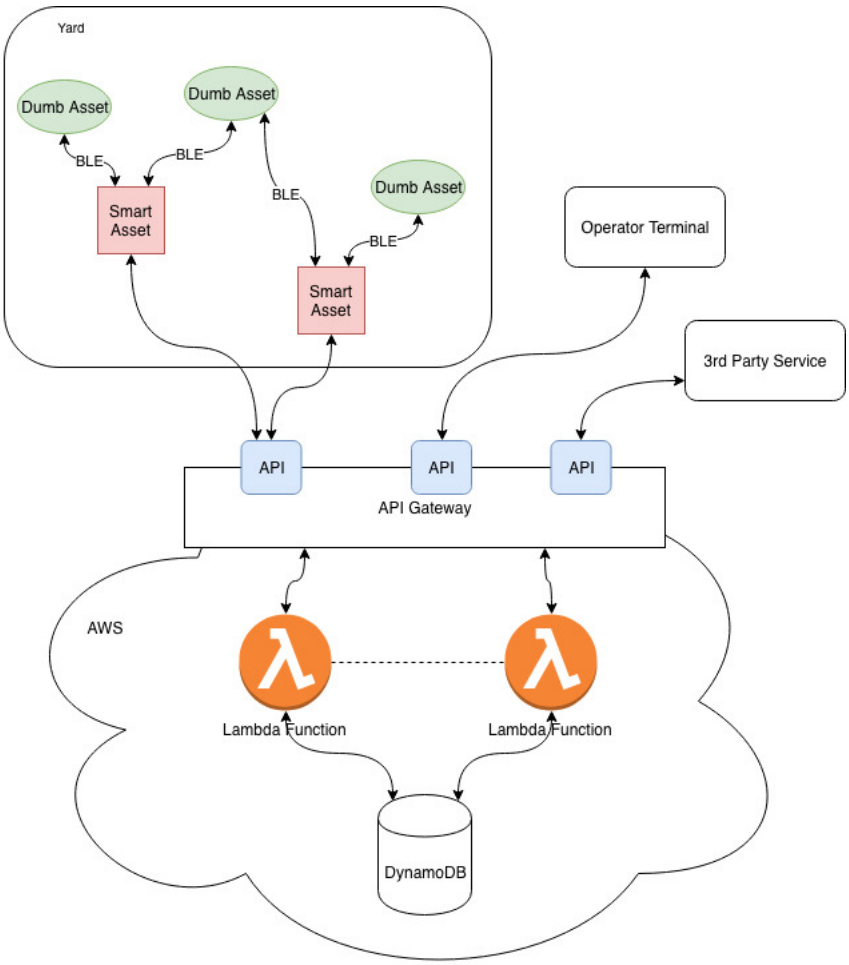

Fig. 3.1. System architecture.

This architecture shows the implementation of the desired features as shown in Fig. 1.1. The 'Yard' is the physical domain in which the assets are located. Here we have many Dumb Assets

(beacons) to a few Smart Assets. These devices communicate with each other using BLE. Cloud Services are deployed on AWS. Device to AWS communication is over WiFi.

The AWS portion of the system consists of three main components; API Gateway, Lambda Functions and a DynamoDB instance. The API Gateway exposes a public facing API and directs API requests to Lambda Functions. Lambda Functions perform computation based on requests and data in the database. DynamoDB is the database used.

Operator Terminal and 3rd Party Service are the two main envisioned use cases for user-facing interaction with the system, providing access to information stored in the system. They are simply examples of requests from the system for information.

3.2 Hardware

Hardware is separated into Smart Devices and Dumb Devices, individually discussed in Smart Devices and

Dumb Devices respectively. Communication between these devices is over BLE.

3.2.1 Smart Devices

Smart device capabilities include:

- Unique ID (Bluetooth MAC address)
- BLE radio
- WiFi radio
- GPS radio
- Knowledge of `/readings` endpoint

Smart devices continually scan for nearby devices using BLE. Nearby devices can include both other smart devices and dumb devices. Smart devices are continually uploading situational information to cloud services over WiFi. The upload/scan frequency can be adjusted. The flowchart of smart device activity is shown in Fig. 3.2.

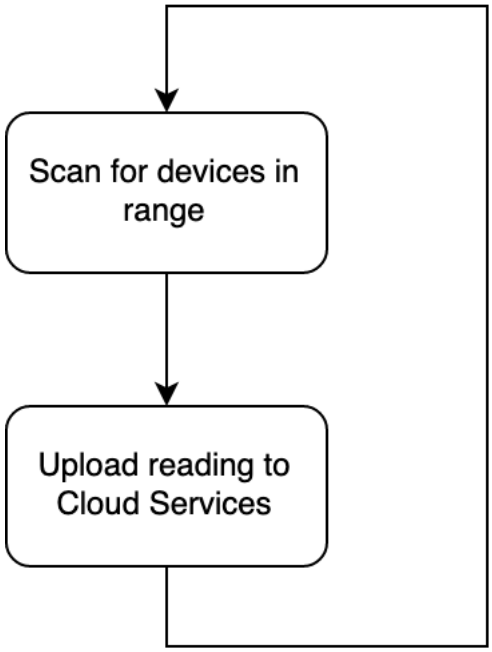

Fig. 3.2. Smart Device Operation Flowchart

The Smart Devices information collected about nearby devices (their `deviceId` and name), their GPS location at the time of scanning, their own `deviceId` and the `timestamp` of the scan to AWS for processing. The reading is uploaded to the `/readings` endpoint using an HTTP POST request as a JSON document. The device aggregates data, no processing of the data is done on the device. Fig. 3.3 shows the data format uploaded from the device. Note that `devices` is an array of device objects – each of which has an address and a name. This array contains every device "seen" at the given location, by the device specified in the `deviceId` field, at the time specified in the `timestamp` field. The `timestamp` field contains the time the scan was run in ISO 8601 i.e. time and date in UTC (Coordinated Universal Time).

```
{
    "deviceId": "78:10:A7:B4:EF:50",
    "timestamp": "2019-02-25T10:30:40.762Z",
    "location": {
        "latitude": 53.2836066,
        "longitude": -9.0649583
    },
    "devices": [
      {
        "deviceId": "98:01:A7:B4:EF:50",
        "name": "trailer_001"
      },
      {
        "deviceId": "98:01:A7:B4:EF:8A",
        "name": "trailer_005"
      }
    ]
}
```


Fig. 3.3. Smart device example upload

For the purposes of the project, a Raspberry Pi 3 B+ [16] was chosen as the 'smart' device. As stated previously, Bluetooth and WiFi are needed capabilities and this device has both. The device has a Bluetooth 4.2 capable radio, which meets the requirements of supporting BLE. The device has a 1.4GHz quad-core processor and is, if anything, over-powered for the purposes of this project. Importantly, the Raspberry Pi does not have on-board GPS, but has 40 GPIO (General-purpose input/output) pins which give it the potential to utilise an external GPS module. The Raspberry Pi has a large community surrounding projects on the platform, which is an advantage when compared to other, less well-known devices.

The external GPS module selected is the Adafruit Ultimate GPS Breakout board [17]. This module can track up to 22 satellites on 66 channels with a -165dBm receiver. In practice this should yield very accurate location data. Importantly, the module can provide updates at a maximum rate of 10Hz, which will allow the Raspberry Pi to get near real-time updates if needed. There is a GPS antenna on-board (-165dBm), but with a uFL connector allowing the use of an external antenna if needed. The Raspberry Pi communicates with this module over SPI on its GPIO ports. The circuit diagram for the connection of the GPS module and the Raspberry Pi is shown in Fig. 3.4. This diagram is originally from the adafruit website.

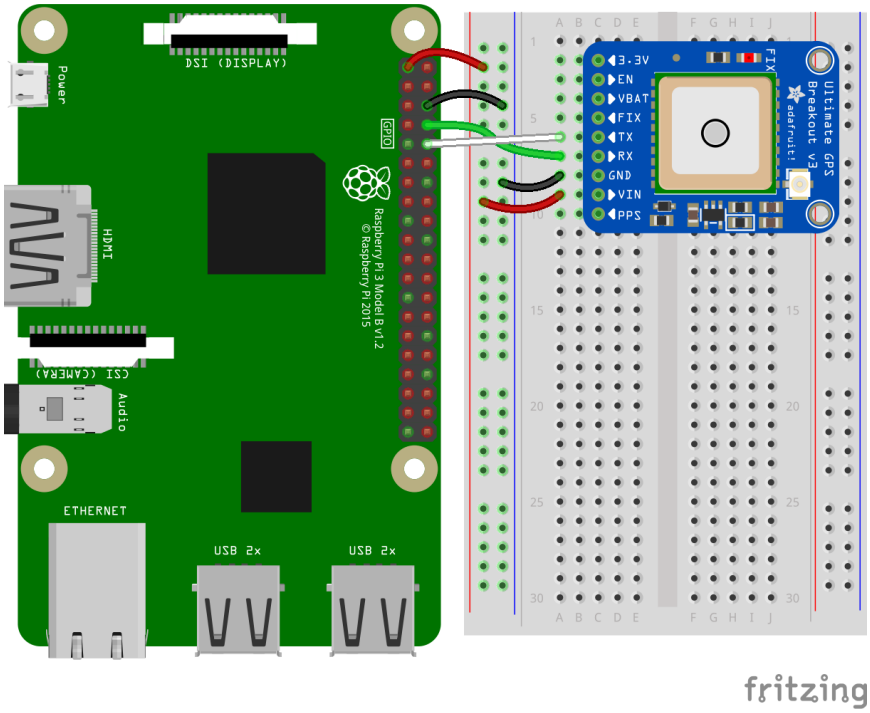

Fig. 3.4. Raspberry Pi and Ultimate GPS Circuit Diagram

The Raspberry Pi runs a Python script to poll for nearby Bluetooth devices, get updated latitude and longitude and upload the resulting data as a JSON (JavaScript Object Notation) document to the API Gateway over WiFi. The python script can be found in some appendix.

The GPS unit used to provide the Raspberry Pi with location awareness is the Adafruit Ultimate GPS module.

3.2.2 Dumb Devices

Dumb device capabilities include:

- Unique ID (potentially Bluetooth MAC address)
- BLE radio

Dumb devices are simple beacons that can be attached to any asset on which telematics information is desired. These beacons are expected to be low-cost, such as insert some research about low cost off-the-shelf beacons here . These beacons are set up with some unique ID, which could be either user specified or simply the Bluetooth Address of the device, which should be unique. These `deviceIds` are recorded in the DB and stored with metadata

such as device name for future use. Storing device names allows a user or customer to assign meaningful identifiers to each device, such as trailer_001 or some such. A dumb devices must be discoverable by other Bluetooth devices. As long as the device is discoverable it will respond to queries from other devices and respond with it's Bluetooth address. If user-defined IDs were required, the device would have to be capable of responding to queries about it's `deviceId`. This would require more complex devices, reduce battery life and increase latency from device discovery to saving the device ID. This is why using the device's Bluetooth MAC address is recommended. Storing only the device's Bluetooth MAC address on the device also minimises the information a foreign agent could glean from the device. If more data was stored on the device such as the system's internal identifier of the device, this information could be used maliciously.

3.3 Cloud Services

As previously stated, cloud services are built using services provided by AWS. AWS divides it's offerings into units called services. The primary services used in this project consist of an API Gateway, AWS Lambda Functions and a DynamoDB instance [18]. A functional diagram of cloud services architecture is provided in Fig. 3.5 below. Note that each component of the cloud services solution is a stand-alone service, with little specific dependencies on the other components. This is deliberate, so that components can be changed and updated with as little impact on other components as possible. For example, the API Gateway could instead point requests at a more traditional server such as an Elastic Compute Cloud (EC2) instance, which could for example host a relational database itself, all without any externally noticeable changes. This loose coupling of components is what will provide the desired adaptability to change. Various other AWS components are used for different utilities such as Identity and Access Management (IAM) for role and user administration. IAM manages the permissions components have to communicate and control other components, for example, a Lambda Function cannot create or delete tables in DynamoDB, but can add and remove information from different tables. CloudWatch is the service used to store and retrieve logs produced by each component. These other services are necessary but not unique to this project.

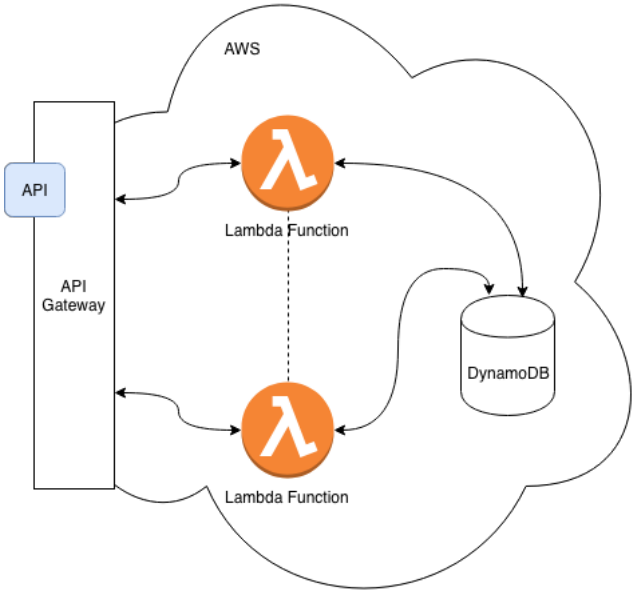

Fig. 3.5. Cloud Services Architecture..

Cloud services are hosted in AWS region US-East-2. This means each component discussed above was hosted in Ohio, in the United States. This is worthy of note as in order to avail of the cheapest rates between AWS services, they must be hosted in the same region. This project should be staying within the free tier allowances for service usage, but US-East-2 was chosen because it is one of the cheapest regions to use for larger volumes of data [19].

3.3.1 API Gateway

The API Gateway is the only public facing component and exposes a HTTP RESTful API. It acts as the 'Front Door' to the cloud services. It exposes HTTP resources e.g. `/api/readings` for use with REST (Representational State Transfer) methods. API Gateway only exposes resources that relate to functions the Lambda functions can handle. The resources API Gateway exposes are shown in **some fig.** API Gateway triggers Lambda Functions based on the resource requested, and acts as a load-balancer. API Gateway responds to requests and triggers Lambda Functions based on these requests. As Lambda Functions only exist as long as they are doing work, this allows the system to scale up and down to meet demand, which is important with respect to the amount of bursty data the system is expected to receive.

The following HTTP endpoints are exposed by the API Gateway.

POST /readings

Description: Submit a new reading

Parameters: None

Body: reading to be submitted.

Responses:

- 200 OK
- 500 Failure

GET /devices

Description: Get a list of devices active in the system

Parameters: None

Body: None

Responses:

- 200 OK. Example Operation:

```
{
    "devices": [
      {
        "deviceId": "<scanned deviceId 1>",
        "name": "<scanned device name 2>"
      },
      {
        "deviceId": "<scanned deviceId 2>",
        "name": "<scanned device name 2>"
      },
    ]
}
```

- 500 Failure

Response Type: JSON

GET /devices/:deviceId

Description: Get information about a specific devices

Parameters: None

Body: None

Responses:

- 200 OK. Example Operation:

```
{
    "devices": {
        "deviceId": "<scanned deviceId 1>",
        "name": "<scanned device name 2>",
        “lastKnownLocation”: {
            “latitude”: “53.29105322965723”,
            “longitude”: “-9.070886204719358”
        },
        “locationUpdateTimestamp”: “2019-04-01T10:52:006Z”
    }
}
```

- 500 Failure.

Response Type: JSON

GET /devices/:deviceId/location

Description: Get the location of a specific device

Parameters: None

Body: None

Responses:

- 200 OK
- 500 Failure

Response Type: JSON

DELETE /devices/:deviceId

Description: Delete the device entry from the devices table

Parameters: None

Body: None

Responses:

- 200 OK
- 500 Failure

Response Type: JSON

3.3.2 Lambda

AWS Lambda provides high availability compute infrastructure, without the need to provision, scale or manage servers [18]. In essence, a Lambda Function is a script that is only run when an event is triggered. In this application, these events are triggered by API Gateway. This design allows only the computational resources needed at any instant in time to be working, and computational resources to scale dynamically in response to load. Lambda Functions can be deployed in many languages, but for this project Node.js was used. Node.js is an open source project providing an asynchronous, event driven JavaScript runtime [20] which is ideal for implementation of a server-side application. As we will see, the implementation of the Node.js application can be run locally on a test machine to simulate behaviour of the Lambda and API Gateway components.

Lambda Functions perform read and write operations to the DynamoDB instance. They validate data uploaded from devices and upload the DB (Database). Lambda Functions also perform computation based on data in the DB. Lambda functions act as computation on demand functions, effectively the same as running a script on a standard machine when there is some requirement for work to be completed. The difference however is that nothing is running until there is work to be completed. API Gateway triggers Lambda functions to be run when it receives a request it deems valid. Lambda functions can also be triggered to run on some event, such as a new item being written to the DynamoDB instance.

To deploy Node.js application code to AWS Lambda and API Gateway, Claudia.js was used. Claudia.js is an open source project that provides an automated deployment and configuration platform that increases the deployment speed and reduces configuration error when deploying application code to AWS [21]. Deployment using Claudia.js also allows a standard express app to be deployed as an API Gateway and a Lambda function. This allowed for the app to be run locally, which provided an opportunity for debugging. This is further discussed in the Testing and Validation chapter.

The Lambda functions run an Express app. The Express app has the following HTTP resources. Note these resources align with the methods exposed by the API Gateway.

POST /readings

Submission endpoint for a reading taken by a smart device. The reading must be of the format shown in Fig. 3.6.

```
{
    "deviceId": "<deviceId of uploading device>",
    "timestamp": "<timestamp of scan completion, ISO 8601>",
    "location": {
        "latitude": <latitude as a number>,
        "longitude": <longitude as a number>
    },
    "devices": [
      {
        "deviceId": "<scanned deviceId 1>",
        "name": "<scanned device name>"
      }
    ]
}
```

Fig. 3.6. /reading Expected Format

Where deviceId is the deviceId of the uploading device, timestamp is the time that the scan was completed, in ISO 8601 format. Location is the GPS location the scan was completed at, an object with latitude and longitude as parameters. Devices is a list of device objects scanned at the given location, where each device consists of a deviceId and a name.

This endpoint verifies that the submitted reading is of a valid format. It then uploads the raw reading to the readings table. After this has been completed, it loops through each device reported in the reading and queries the devices table for the device. If there is no entry for the device in the table, the device is added and it's lastKnownLocation and locationUpdateTimestamp updated to the location and timestamp value reported in the reading. If the device is in the devices table, the lastKnownLocation and locationUpdateTimestamp are updated using the localization algorithm. The lastKnownLocation value is updated by passing the location value in the reading and the current lastKnownLocation to the localization algorithm. Once all the above steps have been completed a 201 (Created) status code is returned in the HTTP response. If any of the steps fail, a 500 status is returned in the HTTP response.

GET /devices

Returns a list of devices from the devices table. Each device is returned as a device object, where a device has a deviceId and a name. On successful compilation of the list, the list is returned as an object entitled devices with a 200 status.

GET /devices/<deviceId>

Returns information about the device specified in the request by deviceId. If the deviceId is invalid a 500 is returned in the HTTP response. Otherwise all the information contained in the devices table for the selected device is returned as JSON. This will take the form of a device object, with deviceId, name, lastKnownLocation and locationUpdateTime parameters. lastKnownLocation is an object with latitude and longitude as parameters. On success the device will be returned with a status 200.

GET /devices/<deviceId>/location

If the device specified with deviceId is valid, the lastKnownLocation is returned with a status 200 OK. If a device with the specified deviceId does not exist the resource returns a 500 with the error message. Response JSON consists of a location object with latitude and longitude parameters (both represented as strings).

DELETE /devices/<deviceId>

If the device specified by deviceId is valid, the item corresponding to this device will be deleted from the devices table in the DB. This deletion is irreversible. Deletion success returns a 200 OK. If device with the specified deviceId doesn't exist or the delete fails, the resource returns a 500 with the error message. There is no response body.

The localization algorithm is an implementation of a Kalman filter, and is run in the POST /readings method. This algorithm takes the current sensor reading i.e. the location object containing latitude and longitude and computes an uncertainty for this reading. It then combines the current reading and uncertainty with the previous location and uncertainty to generate an estimation of the current reading. In this manner it should filter out location reading noise as well as respond to trends in change of location that might be hard to recognise otherwise.

3.3.3 DynamoDB

Amazon DynamoDB is a key-value and document database that provides automatic scaling [18]. DynamoDB is serverless, meaning it runs as a stand-alone instance and doesn't require server management or provisioning. It automatically scales tables depending on capacity and currently needed performance. DynamoDB stores system data for the entire application. The database consists of two tables; devices and readings, the structure of which are shown in Table 1 and Table 2 respectively.

devices	Primary Key	Attributes			
	PK				
	deviceId	name	lastKnownLocation		locationUpdateTimestamp
	deviceId1	deviceName1	latitude	longitude	ISO 8601

Table 1. DynamoDB devices Table

readings	Primary Key		Attributes		
	PK	SK			
	deviceId	timestamp	location		devices
	deviceId1	ISO 8601	latitude	longitude	list

Table 2. DynamoDB readings Table

DynamoDB stores key-value pairs and documents, without defining explicit relationships between tables. Here this means devices.deviceId and readings.deviceId are not related in the database schema, and it is up to the application consuming the database to maintain this relationship. This allows a very flexible schema that can be updated and adapted continually, but care must be taken to maintain relationships between attributes. Document storage allows readings.devices to contain a list of device objects, each of which have a deviceId and a name. location and lastKnownLocation are also stored as objects, which makes programmatic representation easier. Another advantage of DynamoDB is that each item is stored as JSON, removing the need for extra data conversion in the Lambda functions.

Note that readings contains two entries under Primary Key, PK and SK. These correspond to a partition key and a sort key entry. Due to the nature of readings, where a device can upload multiple readings and multiple devices can conceivably upload readings at the same time, a combination of the deviceId and the timestamp must be used to define a composite primary

key in the table. DynamoDB uses the partition key value as an input to an internal hash function [18].

In summary, this chapter focused on the hard implementation and design of the solution. First the overall architecture was broached, with a focus on the function of each component in the system, then the design of each component was explored in a top-down approach.

4 Experiments and Results

This chapter highlights the experiments carried out in development and validation of the project. As the project consists of multiple, separate components working together, the functionality of each component must be validated individually, as well as in the context of the whole system. Unit Testing was a tool used to assist in development of the system, in order to manage the complex behaviour of the cloud components. Testing was carried out at component level on the GPS module.

4.1 Testing and Validation

4.1.1 Unit Testing

In order to validate the behaviour of the cloud services, unit tests were written. These tests can be configured to run against an API running on the local machine, or against a live API Gateway endpoint. Utilisation of these tests served as software contracts, so that software components could not be accidentally be broken during development of new features.

The software development process used to develop cloud services Test Driven Development (TDD), where the process of implementing a feature is as follows; first, a failing test is written to specify the behaviour of the feature being developed, then the feature is developed to a standard that allows the test to pass. This process forces the behaviour of a feature to be ironed out before the feature is implemented, thus enforcing good design practices.

As the cloud services all run in the cloud, in an environment that we don't have complete visibility over, debugging was a challenge. The Lambda functions output log files to AWS CloudWatch, but online debugging is not available. This is part of the reason that the API Gateway and Lambda code was written as an Express application, to allow the Express app to be run locally on a development machine, in an environment that allowed debugging. This was beneficial in debugging errors, both within the cloud services coded and within the simulation. The simulation software could be directed to send requests to the local machine, allowing visibility of the requests the simulation software was outputting.

Unit testing could be performed against code running on the local machine or against code running in the cloud with the toggling of an environmental variable. Uploading code from the development machine to production was time consuming, and validation that the code would work locally before pushing it to production was helpful.

Validation of the code running on the Raspberry Pi was completed in **x** steps. The function of the code was broken down into reading and submission of data. This allowed the code to be run on a development machine to verify submission of mock data in the correct format. Once this was completed the script running on the Raspberry Pi was incrementally developed to add the different components of the reading. These included the timestamp, nearby Bluetooth devices and the location reading from the GPS module. Incremental validation of these components prevented major breakages in the process.

4.1.2 GPS Accuracy Testing

Purpose

To get real world numbers for accuracy and an indication of the noise of the Adafruit Ultimate GPS module by leaving the GPS module in a known position for an extended duration of time. Noise and inaccuracy of the GPS module will provide input to the simulation software, in order to simulate more realistic data.

Method

1. The Raspberry Pi and the GPS module were connected to each other. An external antenna was connected to the GPS module and the antenna placed on the outside of a skylight, so as to have an uninterrupted view of the sky.
2. A Python script was loaded onto the Raspberry Pi to query the GPS module for a new GPS reading at a frequency of 1Hz. The script saved the reading to file. This script can be found in the appendix.
3. The script was left running for 90 minutes.

Results

The experiment collected 5779 readings, equating to 96 minutes of runtime. Another Python script was used to read from the file and compute the mean and variance of the data from the readings. The results of this script are shown in Table 3.

mean latitude	53.27867288977332
mean longitude	-6.144917466401338
mean number of satellites	7
Actual latitude	53.278558
Actual longitude	-6.145032

Table 3. GPS Analysis Results

The values for 'Actual latitude' and 'Actual longitude' were generated using Google Maps to generate latitude and longitude for the location the GPS module was placed.

The average difference between the 'actual' location and the location as reported by the GPS module i.e. the 'reported location' works out as 14.87 meters.

Table 4 shows a table of the distance in meters from the mean estimation of each reading. Table 5 shows the distribution of distances from the mean. 11.94 meters is the standard deviation of distances from the mean.

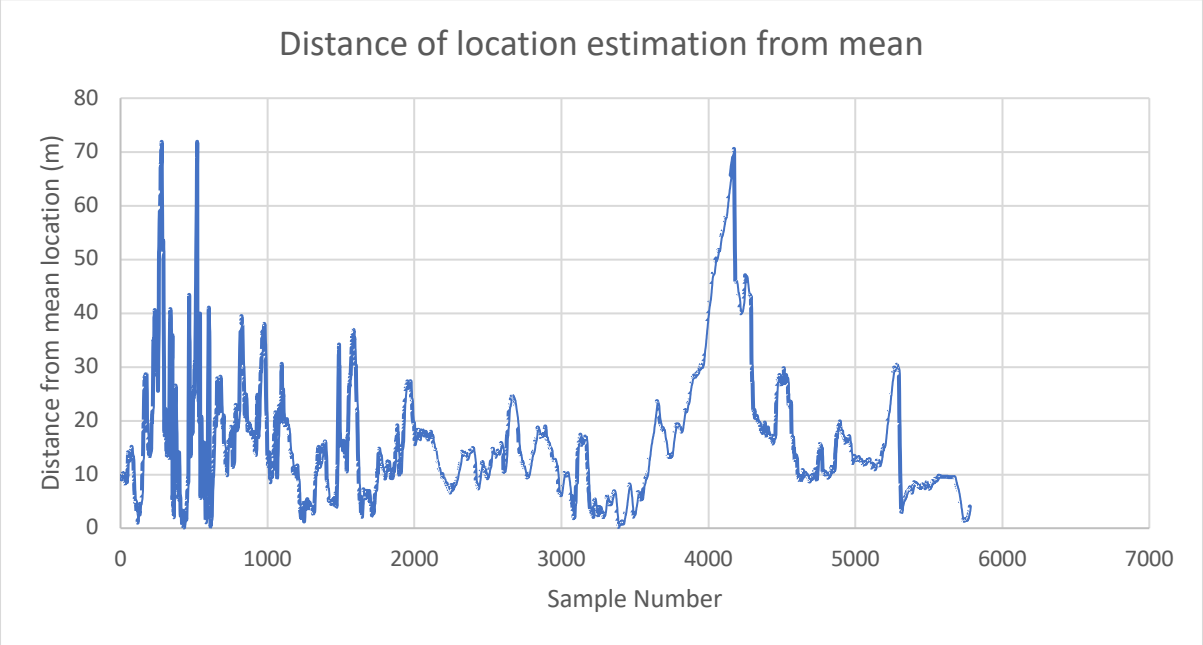

Table 4. Distance from Location Estimation to Mean

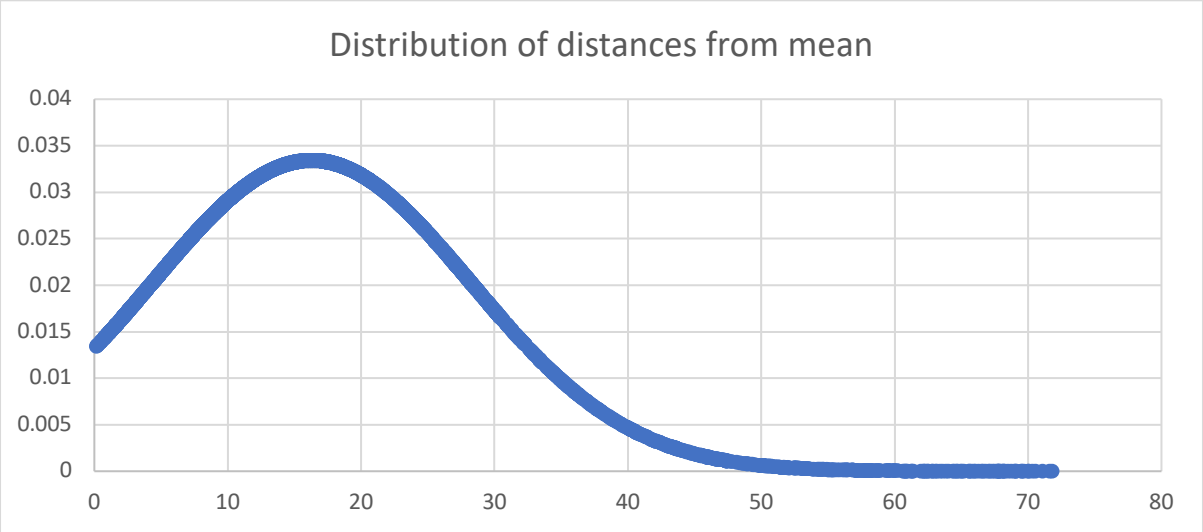

Table 5. Distribution of Distances from the Mean

Conclusion

The GPS readings vary significantly, with some readings as far as 70 meters from the mean. The GPS readings could be calibrated to improve accuracy and filtering of the reading could also improve precision. From the standard deviation of the distances of locations from the mean we can see that 68% of readings are within a radius of 11.94 meters.

In order to get a latitude and longitude reading, a GPS module must be in contact with a minimum of 3 satellites. The more satellites that can be contacted, the more accurate the reading should be. The mean number of satellites the GPS module was in communication with was 7, which would normally lead one to assume an accurate reading was provided. Looking at the distribution of distances from the mean, one can safely say that the readings are not accurate.

An average distance error of 14.87 meters is quite significant in the context of localisation of a device within a shipping yard, where ISO standard shipping containers are 2.44m wide and either 6.06m or 12.2m long [22].

The accuracy and precision of the GPS module is disappointing.

4.2 Simulation

As discussed above, AnyLogic simulation software is used to simulate the movement of devices in the real-world. AnyLogic provides the mock data for AWS to ingest, and compares AWS data on device locations to actual locations, as in simulation. The simulation is run with agents on a GIS map so that real latitudes and longitudes can be provided to AWS. As far as AWS is concerned, the data coming from the simulation is no different from real world data it would receive. Fig. 4.1 shows how the simulation software communicates with AWS. The simulation software is Java based, allowing programmatic HTTP requests to be sent to AWS using Apache Web Components. The simulation user specifies input variables such as the root endpoint the simulation software is targeting (localhost used for testing), the number of smart and dumb devices to simulate, the Bluetooth radio range of devices.

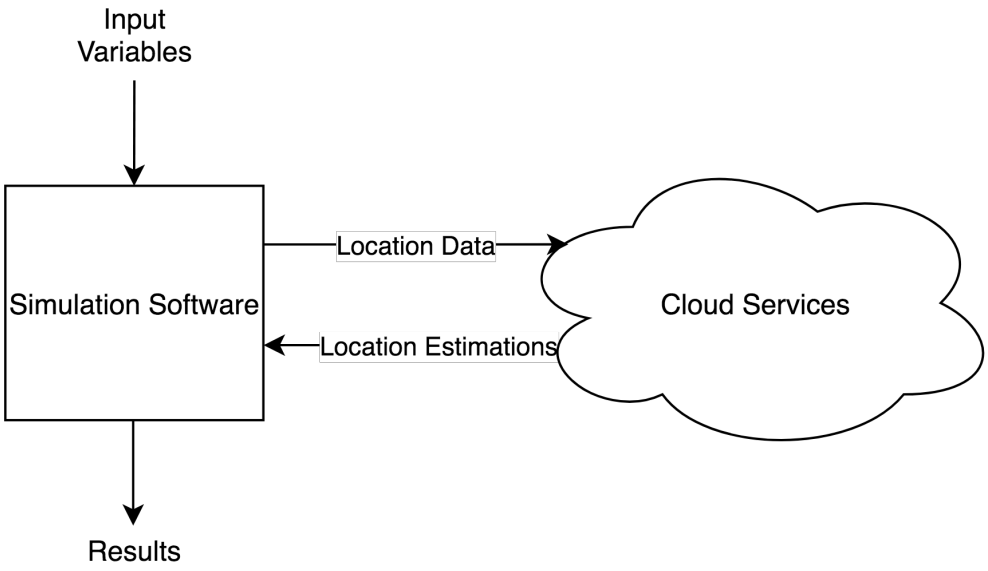

Fig. 4.1. Simulation Architecture.

The simulation software runs the simulation with based on the set of user inputs provided, supplying AWS with 'real-time' information during simulation. This real-time data comes from the smart devices being simulated, as they move around the simulated yard on predetermined routes they are continually scanning for other nearby devices and uploading data to AWS, just like real world devices would do.

AnyLogic allows custom agents to be built using their drag and drop methodology. This drag and drop functionality is supplemented by allowing the input of custom Java code into the agent. This allows agents to extend other agents. Thus, we have the following hierarchy of agents as shown in Fig. 4.2, where each arrow depicts extension of the object above. i.e. Forklift extends Smart Device, which extends Device. Only the bottom objects (Forklift, Tractor and Trailer) are instantiated.

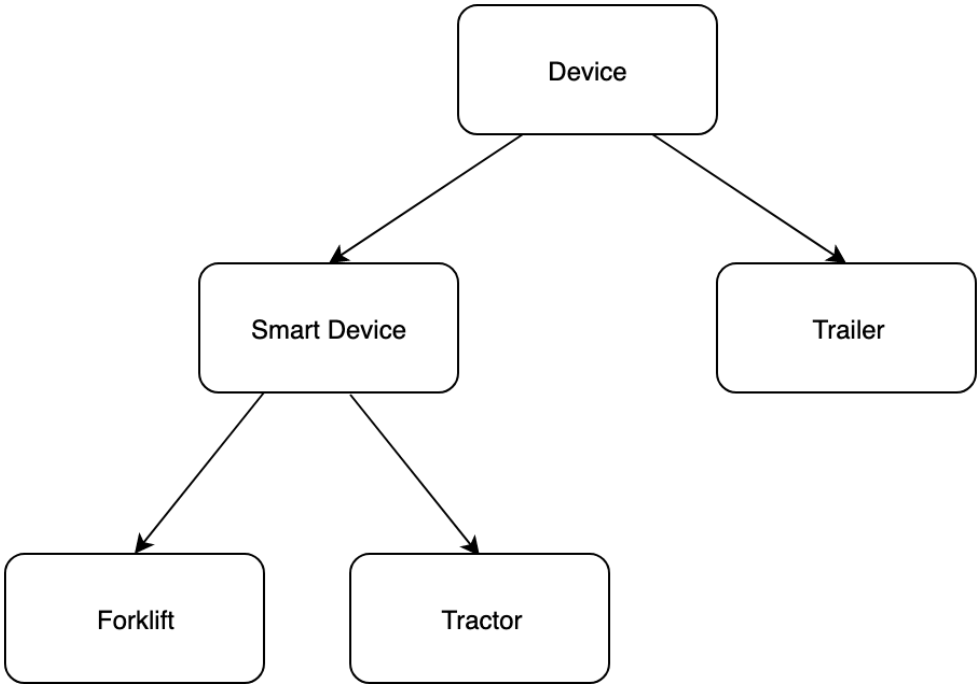

Fig. 4.2. AnyLogic Device Hierarchy

When the simulation starts, it instantiates a number of devices, both smart and dumb, in various locations on the GIS map. Each device is assigned an autogenerated deviceId of the form sim<id_number> where id_number is an integer incrementing from 1. Requests are sent to the cloud services to delete any devices that share the deviceId. This prevents data from previous simulations affecting the current simulation.

The simulation contains a GIS map on which objects are placed, which provides objects with valid real-world locations. Objects are placed at specific locations defined by latitude and longitude. The smart objects are set up to follow certain routes on the map, and the dumb objects remain stationary. As each smart objects moves along it's pre-determined route, it

'scans' for objects within range. In practice, each object has access to every other object on the map, but it only 'reads' objects that are within range. Each device extending Device has an inherited method called isWithinRange(). This method accepts a Device as the single parameter and verifies that the Device passed is within the range of the callee's bluetoothRange parameter. Thus, smart devices can only read from devices that are close enough that the distance between them is less than both of their bluetoothRange parameters. Range is an adjustable parameter, as is speed and the scan interval. With every scan, smart devices upload the reading they have taken to the cloud services and verify the reading is accepted. Thus the cloud services receiving real -time data.

When the simulation has ended, the simulation queries AWS for device location estimations, for every device in simulation. The location data returned from AWS is compared with actual location data the simulation has of every device and the resulting accuracy (i.e. distance from estimation to actual location) is displayed to the user.

The simulation can also be directed to query localhost as the root endpoint, instead of the active API Gateway.

4.2.1 Kalman Filter Measurement Error Test

Purpose

This experiment seeks to find optimum values for the measurement error in use in the Kalman filter, in order to minimise the location estimation error. This is done through trial and error, using a simple simulation and the cloud services code running on the local machine.

Setup

A simple simulation consisting of a single stationary trailer and a moving forklift was set up using the simulation software. A screenshot of the simulation setup is shown in **Error! Reference source not found.**. The forklift moves from the position shown in the figure along the dotted line to the red dot at the end of the line, at a speed of 10kmph. The trailer is shown as a large red rectangle, and remains stationary throughout the simulation. The forklift is a smart device and the trailer is a dumb device. The simulation simulates a drive by interaction between the forklift and trailer.

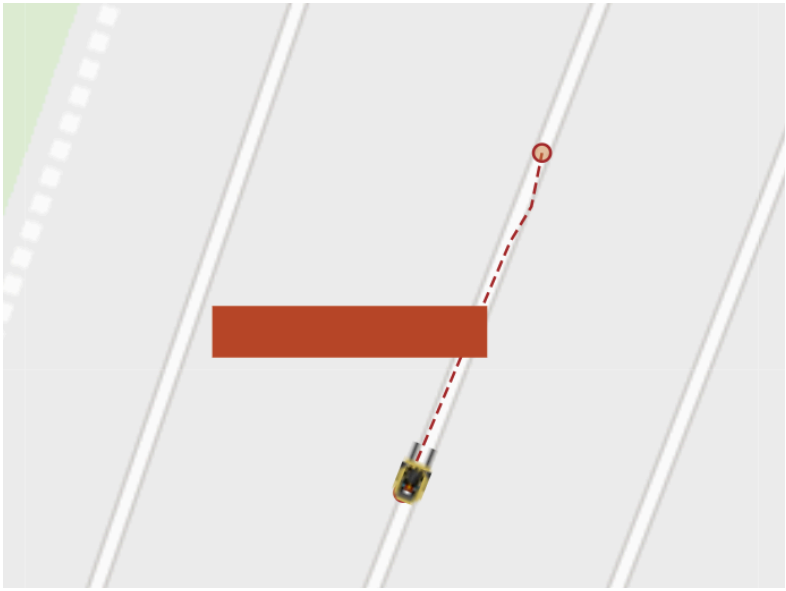

Fig. 4.3. Kalman Measurement Error Experiment Simulation Setup

When the simulation begins, the simulation places the forklift and trailer on the GIS map in the positions shown. The simulation autogenerates deviceIds for each device, and sends a request to the cloud services to delete the devices. Deleting the devices erases previous lastKnownLocation information from the database, allowing the Kalman filter to start from a blank slate with no previous assumptions or estimated location available. The simulation software is pointed to query localhost for cloud services.

The cloud services code contains constants which define the values of the measurement error for use in the Kalman filter. As the code is running locally, these values can be changed and the program simply restarted to force the new values to be used in calculations.

Method

1. The measurement error values under test are written to the cloud services code and the cloud services restarted locally. For simplicity, the same value is written to both the parameter used for location estimation of the smart devices and the parameter used for dumb device location estimation.
2. The simulation was run and the reported distance error was recorded.
3. The experiment was repeated with various values.

Results

Table 1 shows the results from the experiment. Fig. 4.4 graphs measurement error with estimated location error for both the smart device and the dumb device.

Measurement Error	Smart Device Error (m)	Dumb Device Error (m)
10	19.72527886	8.865339992
5	19.57605335	8.709649509
1	18.51419621	7.863447568
0.5	17.44819143	7.386517479
0.2	15.3036416	7.146114764
0.1	13.39737652	7.352183359
0.01	9.508603216	8.121576837
0.001	8.846686051	8.259345813
0.0001	8.77581702	8.27399154
0.00001	8.768680006	8.275465148
0.0000001	8.767894376	8.275627345
0.000000001	8.767886519	8.275628965
1E-11	8.76788634	8.275629032

Table 6. Kalman Measurement Error Results

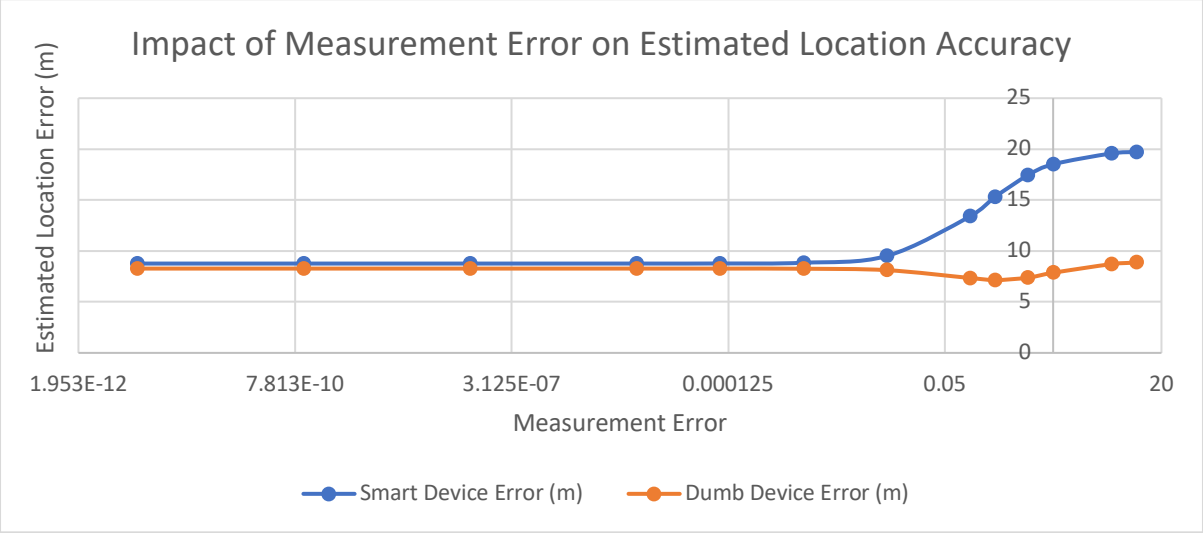

Fig. 4.4. Graph of Measurement Error vs Estimated Location Error

Conclusions

The discrepancy between the error experienced in estimating the location of the forklift (smart device) and that of the trailer (dumb device) is interesting to note, in that the Kalman

filter estimates the location of the trailer more accurately than that of the forklift, even though the forklift is reporting it's exact GPS position every second to the cloud services. The lowest error achieved in predicting the location of the forklift was 8.76m with a measurement error of 0.00001 and below. Fig. 4.4 shows decreases beyond 0.00001 cease to improve the prediction error. The lowest error achieved in predicting the location of the trailer was 7.15m with a measurement error of 0.2, which is a markedly improved result over best effort error achieved in predicting the location of the forklift.

The location prediction accuracy of the smart device was expected to be significantly greater than that of the dumb device, based on the smart device providing cloud services with accurate GPS data on it's own location. However, the Kalman filter actually performed better when predicting the location of the dumb device. It should be noted that the simulation did not add artificial noise, thus cloud services were receiving the actual location of the smart device. Even when the measurement error for the smart device was reduced to arbitrarily small values, the accuracy ceased to increase. This would suggest limitations in the ability of the Kalman filter implementation to accurately predict locations.

The simulation run is quite short, the forklift takes a total of 8 readings spaced at 1 second intervals, which provides the Kalman filter with a total of 8 individual readings, 1 of which reads the trailer as being in range. This lack of information is likely a source of error for the filter.

In summary, this chapter detailed the use of experiments and tests to develop and validate the solution. The use of TDD to drive development of cloud services is of particular note, with detailed reporting on the testing of real-world characteristics of the GPS module provided. Finally the simulation software and process was detailed.

5 Discussion and Conclusions

This chapter will discuss the results of the project and the success of the project in terms of meeting goals. Conclusions are drawn from the project and recommendations for future work are provided.

5.1 Results and Discussion

Accuracy generated from simulation and localization algorithm.

Latency of cloud services from reading submission to submission being loaded into database.

5.2 Achievement of Goals

The project achieved the goal of building an entire system that can be used to estimate the location of a fleet of devices, without each device being location-aware. The goals were to implement this using scalable cloud computing, which was achieved. However, the degree to which the project met these goals was not as high as hoped. It was hoped the solution would be more effective at estimating the location of devices, but as it stands the solution is limited in the situations it performs well.

It could be argued that the goals set out at the start of this project were possibly too ambitious. The scope of the project was very broad, and rather than answering a single specific question, such as an implementation of a cloud-based localization algorithm, it sought to provide an entire solution to the problem of asset location.

5.3 Challenges

Test Driven Development proved to be an effective process for writing software and in particular managing the complex relationships between components of the project. A large part of the project was management of all the different components and managing communication between them.

Time management could have been better, setting deliverable goals besides the deliverables required for completion of the project from the beginning of the project would have been helpful. Initially, large amounts of time were spent simply researching the project, and as such

it was difficult to plan in advance what deliverables would be needed and when. Regardless, the pressure increased as the deadline grew near. A large amount of time was spend on physical hardware, and this turned out to not be as important of a component to the project as was initially expected, due to the ability of the simulation software to simulate real-life hardware. Time could possibly have been better spent working with the simulation software to develop more complex simulations to use as the basis for refinement of the location estimation algorithm in the cloud services.

The cloud services worked quite well, for a basic implementation of such a solution. The time investment needed to get all the components up and running was underestimated, and as such assembling the basic components took more time than expected.

The time needed to implement a simulation in simulation software, once selected, was also underestimated. Initially it was expected that the simulation software would need to be wrapped in custom software, which would have consumed even more time but thankfully AnyLogic was powerful enough to allow the simulation software to manage itself.

5.4 Recommendations for Future Work

Future work should include development of a more capable location estimation algorithm. The algorithm implemented is quite basic in nature, and has not been tested with bursty or noisy data. Development of more complex simulations would go hand in hand with the development of this algorithm and allow testing of a greater variety of situations against the algorithm.

Part of the goal of this project was to build a solution that would work in a real-world implementation, but some aspects would need to be developed to allow this to happen, such as some system for device authentication and a greater set of API resources to allow management of devices.

This project aimed to be scalable up to large numbers of devices and much higher traffic, but in order to do this some changes should probably be made, such as the time that the location

estimation algorithm is called. At the moment, when a new reading is submitted to /readings, the same code that submitted this reading then calculates a new estimated location and ensures all devices uploaded in the reading are accounted for in /devices. This functionality could be moved to another Lambda function that is called when the /readings table is updated, thus separating responsibility of updating /devices to another piece of code, that is a device in the field is not waiting on a response from.

References

[1] Federal Motor Carrier Safety Administration, “Electronic Logging Devices Implementation Timeline,” U.S Department of Transportation, 2018.

[2] F. S. A. o. Ireland, “Storage and Transport of meat,” 21 2 2017. [Online]. Available: https://www.fsai.ie/legislation/food_legislation/fresh_meat/storage_and_transport.html. [Accessed 3 4 2019].

[3] Orbcomm, “Trailer Tracking Improves Turn Times and Revs Up Revenue for Christenson Transportation,” 1 November 2017. [Online]. Available: https://blog.orbcomm.com/trailer-tracking-improves-turn-times-revs-revenue-christenson-transportation/. [Accessed 4 4 2019].

[4] FMCSA, “Driver Detention Final Report ST2018019,” 2018.

[5] S. Son, B. Kim, H. Park and Y. Baek, “Hierarchical asset tracking system using IEEE 802.15.4a radio in container terminals,” in *IEEE*, Seoul, South Korea, 2016.

[6] Orbcomm. [Online]. Available: https://www.orbcomm.com/. [Accessed 4 4 2019].

[7] IEEE, “IEEE std 802.11-2012 (Revision of IEEE std 802.11-20017),” 2012.

[8] Bluetooth, “Core Specification v5.0,” 2016.

[9] F. E. P. van Diggelen, “The World’s first GPS MOOC and Worldwide Laboratory using Smartphones,” *Proceedings of the 28th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2015), Tampa, Florida, September 2015,* pp. 361-369.

[10] R. T. Fielding, *Architectural Styles and the Design of Network-based Software Architectures,* Irvine: University of California, 2000.

[11] UMNISCO, “GAMA Platform,” [Online]. Available: https://gama-platform.github.io/. [Accessed 6 January 2019].

[12] “Anylogic,” The AnyLogic Company, [Online]. Available: https://www.anylogic.com/.

[13] T. Brinkhoff, “Generating Network-Based Moving Objects,” GeoInformatica, Vol. 6, No.2, 2002.

[14] N. Pelekis, S. Sideridis, P. Tampakis and Y. Theodoridis, “Hermoupolis: A Semantic Trajectory Generator in the Data Science era,” he SIGSPATIAL Special Newsletter of the Association for Computing Machinery, Special Interest Group on Spatial Information, Vol. 7, Number 1, March 2015.

[15] “Hermoupolis Download Request,” [Online]. Available: http://infolab.cs.unipi.gr/?page_id=2255. [Accessed 6 January 2019].

[16] “Raspberry Pi 3 B+,” Raspberry Pi Foundation, [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/. [Accessed 6 January 2019].

[17] “Adafruit,” Adafruit, [Online]. Available: https://www.adafruit.com/. [Accessed 6 January 2019].

[18] “AWS Documentation,” Amazon Web Services, 2019. [Online]. Available: https://docs.aws.amazon.com/. [Accessed 6 January 2019].

[19] E. Marquez, “Save yourself a lot of pain (and money) by choosing your AWS Region wisely,” [Online]. Available: https://www.concurrencylabs.com/blog/choose-your-aws-region-wisely/. [Accessed 10 November 2018].

[20] “Node.js,” [Online]. Available: https://nodejs.org/. [Accessed 6 January 2019].

[21] “Claudia.js,” [Online]. Available: https://claudiajs.com/. [Accessed 6 Jan 2019].

[22] *Series 1 freight containers - Classification, dimensions and radings,* IEEE Standard 668.

Appendix

Repository

All the software used for this project can be found at https://github.com/monkfungus/FYP. Git and GitHub were used throughout this project to manage and store versions of the project, thus this repository contains a historical record of project development and all the files, results, diagrams and scripts used in development of this project. This repository contains a readme file that explains the layout of the repository and the process to recreate the solution presented in this report. The readme is located at https://github.com/monkfungus/FYP/readme.md. A history of commits can be found at https://github.com/monkfungus/FYP/commits/master.

Gantt Chart