Multilateration-based localization of nodes within a LoRaWAN network

Daniel Saul
University College London

Supervisor: Dr Miguel Rio

March 21, 2017

Abstract

I wish I were an abstract.

Contents

Chapter 1

Introduction

The Internet of Things (IoT) is one of the most hyped of technology buzzwords at the moment but the term was coined as long ago as 1999 by Kevin Ashton. Back then, all data on the internet was generated or uploaded by humans, whether text, images or information. But Ashton envisioned a future of computer systems generating and collating data on their own, with little to no input from us. Today we are in the situation where

1.1 Low Power Wide Area Networks

1.2 LoRaWAN

1.2.1 Things Connected

1.3 Motivations

1.4 Project Outline

Chapter 2

Localization

2.1 Received Signal Strength

2.2 Angle of Arrival

2.3 Time of Arrival

2.4 Time Difference of Arrival

2.4.1 TDOA Equations

2.4.2 Hyperbolic Solver Algorithms

2.4.3 Problems

Chapter 3

Implementation

This chapter will describe the implemented test setup, ranging from the hardware through to the software and data gathering. The setup consists of a simple custom node, the existing Things Connected LoRaWAN infrastructure across London and custom server-side software. The node transmits its real location as calculated by the on-board GPS which is received by LoRaWAN gateways and uploaded to the server. These real coordinates of the transmission location can then be later compared to the estimated location by the multilateration algorithm.

3.1 Node Hardware

A simple node was designed, built and programmed in C, consisting of an ESP8266 microcontroller, u-blox MAX8C GPS receiver and a Microchip RN2483 LoRa transceiver, as shown in figure 3.1. The RN2483 was chosen as it implements the full LoRaWAN stack on-board, reducing the complexity of the microcontroller code. The GPS receiver used a basic chip antenna whilst the LoRA transceiver used an 868MHz 'rubber ducky' antenna. The specific hardware is not particularly important however, but rather the functionality is.

The GPS receiver was configured to pedestrian mode since the primary method of testing would be a person walking with the node, as opposed to a flight or high velocity mode. The LoRaWAN configuration parameters (device address, network session key and application session key) were hard-coded into the microcontroller firmware.

The microcontroller follows a basic loop. In each iteration, set at roughly every 10 seconds, the microcontroller requests data from the GPS (latitude, longitude, altitude, number of satellites and time)

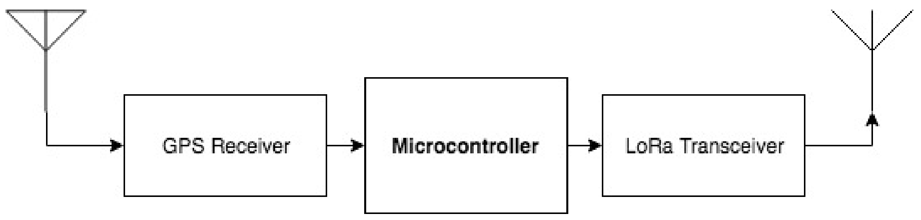


Figure 3.1: Node Diagram

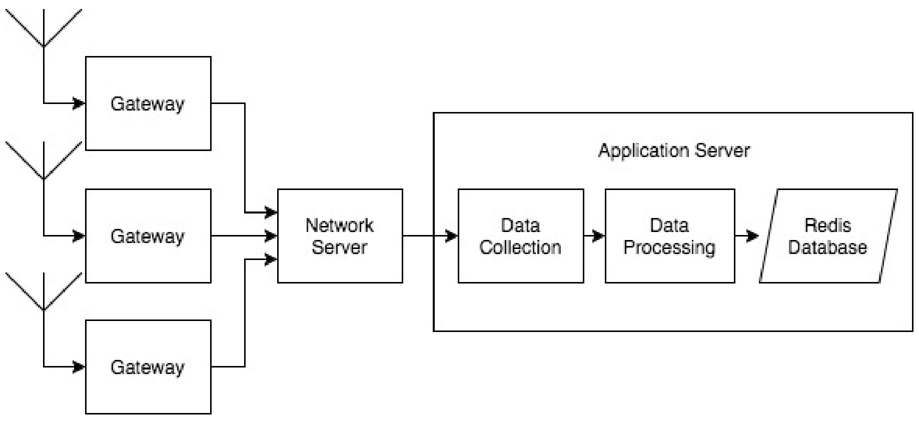


Figure 3.2: LoRaWAN to Server Diagram

and loads it into a struct datatype. A counter in the microcontroller's flash memory, such that it is persistent across power cycling, is incremented and also added to the struct. The entire 20 byte struct is then transmitted as a binary blob payload by the LoRa transceiver. Other functionality allowing transmission on a button press and short bursts of continuous data transmission was also implemented but not ultimately used.

The main file of the node firmware can be seen in appendix A.

3.2 Server-side Software

When a gateway receives a packet from a node, it uploads it to the network server. The data can then be passed by the network server over an HTTP connection to the relevant application server, if the application server has implemented the necessary JSON-RPC methods [1]. A python script to be run on a public server with open port 8080 is found in appendix B.1 and implements the 'uplink' and 'post_uplink' methods. The 'uplink' method provides data as soon as one gateway has received a packet whilst the 'post_uplink' method should provide data on all receiving gateways a few seconds later. Unfortunately after testing, it was found that this latter method is not functional and that therefore obtaining details on all gateways is not possible this way. Hopefully it will be in the future.

Therefore, a workaround was found by using the unofficial API running the Things Connected dashboard. The last 20 received packets from each gateway are available from a 'messages' endpoint. By continuously polling this endpoint at a rate faster than that at which the node is transmitting, all data from all gateways can be collected. Figure 3.2 shows a simplified system diagram for the LoRaWAN and server-side systems. A python script to perform an HTTP POST request to login and continually fetch data from the unofficial API through HTTP GET requests is found in appendix B.2. The fetched data is published to a Redis Pub/Sub channel which is subscribed to by another script, found in appendix B.3, where the data is processed. The payload of each packet is decoded and the binary blob unpacked into the constituent pieces of data. If the packet payload already exists in the Redis database, the details of the new gateway receiving the packet is appended to the gateway list. If it doesn't already exist, the entire packet data is freshly inserted into the database.

It is at this point in the system that any final positioning algorithm would ultimately be included. Instead, another python script, in appendix B.4, allows all the received data to be exported from the database to a JSON file for analysis and testing.

Chapter 4

Data Analysis & Results

In this chapter, the gathered data and the various transformations, algorithms and analytics carried out will be presented alongside the final positioning errors obtained. All investigations were programmed in Python and carried out in an iPython Jupyter notebook.

4.1 Gathered Data

The GPS node was taken for a walk through central London, through both areas with a high number and low number of gateways. Table 4.1 shows the number of gateways that each packet was received by, with 648 packets received in total by 8 different unique gateways. It can be seen that only about 25% of packets were received by 3 or more gateways, with three gateways being the minimum requirement to estimate a 2-dimensional position.

Figure 4.1 provides a geographical visualization of the data. The location of each gateway that received a packet is shown in blue, the true location of packets received by less than 3 gateways are shown in red and the true location of packets received by 3 or more gateways are shown in green. With a cursory glance, it can be seen that generally more packets are received by higher numbers of gateways in areas of high gateway density.

Taking a closer look at the received data, each gateway provides the latitude, longitude and altitude of its position and a timestamp with microsecond precision. Since the position coordinates for each gateway vary between packets, it would seem that for every packet received, a new location is retrieved from the GPS. It is unknown whether the timestamp is also retrieved live from the GPS, or from an internal clock. Either way, microsecond precision is not ideal bearing in mind that 1 µs is equivalent to 300 metres in distance. This also means it is unlikely the timestamping occurs at the instant a packet is first received.

Table 4.1: Number of packets received by n gateways

Number of Gateways	1	2	3	4	5	Total
Number of Packets	310	168	111	46	13	648

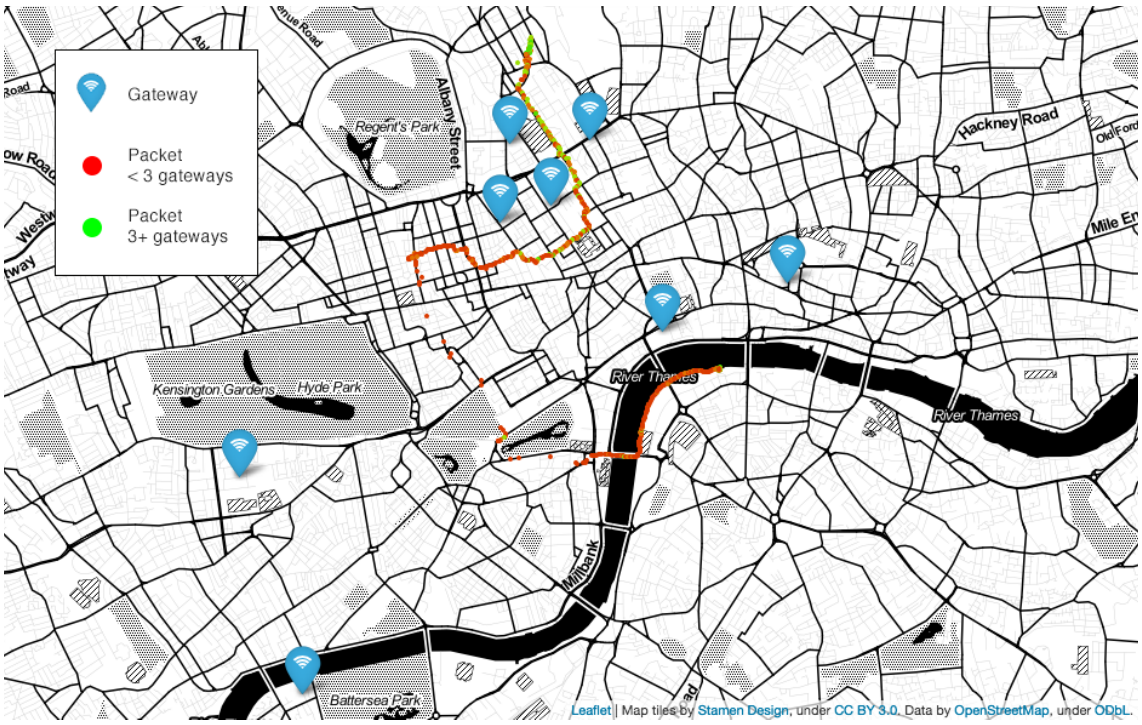


Figure 4.1: Geographical visualization of gathered data

4.2 Cartesian Coordinates

When working with the TDOA algorithms, it is important that a linear cartesian coordinate system is used, as opposed to the WGS84 latitude and longitude used by the GPS.

If 3-dimensional positioning was required, latitude, longitude and altitude (LLA) could be converted to the Earth Centred, Earth Fixed (ECEF) frame and vice versa. In this reference frame, the origin is at the centre of the Earth with the x-axis intersecting the Greenwich meridian and the equator, the z-axis being along the spin axis of the Earth and the y-axis being perpendicular to both of these. However, for 3-dimensional position estimates, a minimum of 4 gateways would be required and the effect of altitude is likely to be negligible compared to other sources of error.

For 2-dimensional positioning, a local grid reference frame can be used. Latitude and longitude can be converted to OSGB36, the British Ordnance Survey National Grid reference system, and vice versa, through a number of calculations and adjustments detailed in OSTN02 [2]. The resulting cartesian coordinates are in metres and also allow for easy plotting of the hyperbolic equations and position estimates. When positioning errors and distances are given, it is simply the euclidean distance between two sets of (x,y) coordinates in metres.

4.3 Received Signal Strength

The Received Signal Strength Indicator (RSSI) has already been discounted as a viable method of positioning due to the broad number of factors involved of which distance is only one. Figure 4.2 plots the RSSI of each packet received by each gateway against the distance from the transmission to the gateway. Whilst it can be seen that there is a relationship, the large variance of RSSI for any distance, particularly at the lower end, makes it unusable for positioning on its own.

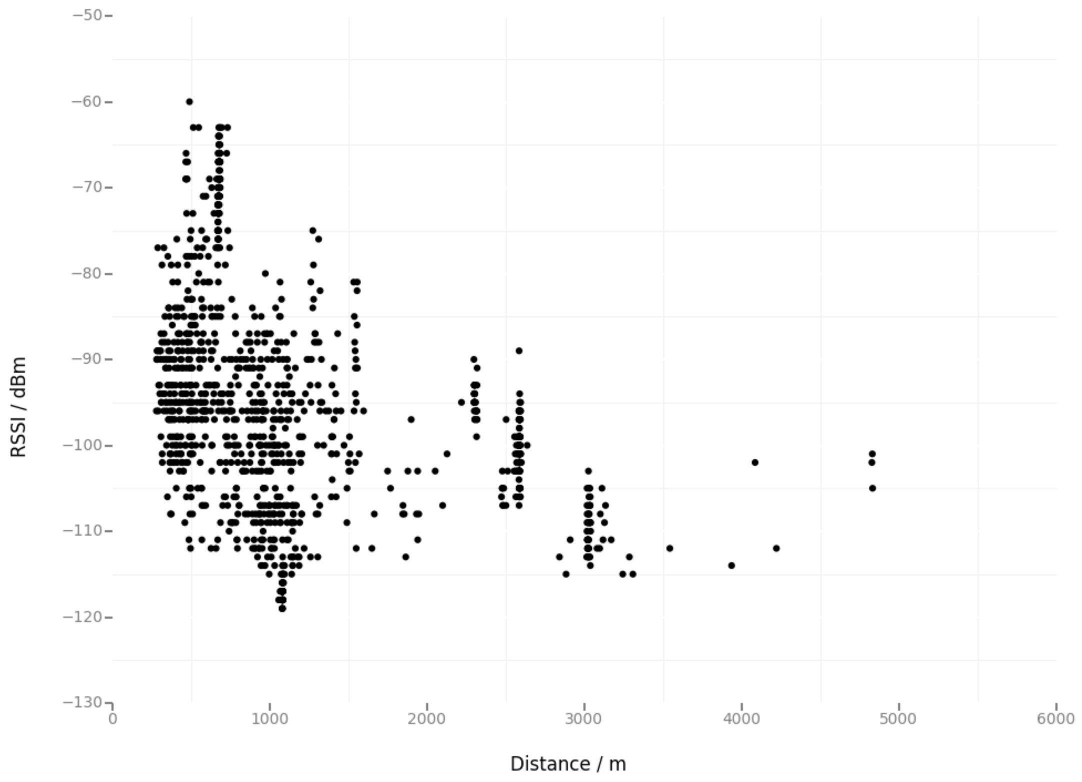


Figure 4.2: Relationship between RSSI and distance from gateway

4.4 Chan's Algorithm

Initial investigations were carried out using an implementation of Chan's algorithm [3].

The results of applying Chan's algorithm to the raw data directly, choosing the latter 3 gateways to have received each packet, were disappointing and can be seen in table 4.2. Out of 170 possible packets that were received by 3 or more gateways, the algorithm only returned position estimates for 18, or about 11%. The mean positional error of these when compared to the true locations was 1112 metres.

When the hyperbolic equations were plotted, it was found that the majority of packets did not have hyperbola that ever intersected or did not have hyperbola that were even valid due to the limited precision of the gateway timestamps. One such instance can seen in figure 4.3. A limitation of Chan's algorithm is that it calculates exact solutions to the hyperbolic equations, so in all these instances where no exact solutions exist it was unable to provide a result.

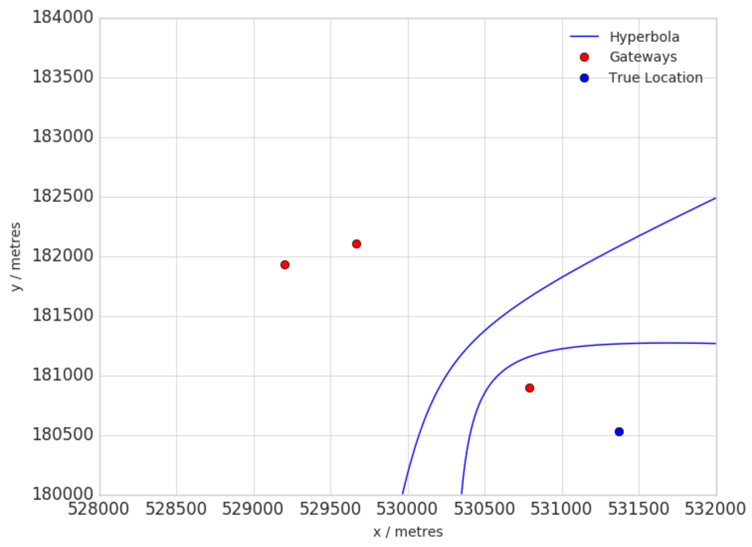


Figure 4.3: Geographical plot of packet with non-intersecting hyperbola

Table 4.2: Results of Chan's algorithm

	Resolvable Packets	Mean Error	Median Error	Standard Deviation
Raw Data	10.6%	1112m	882m	671m
Offsets	17.0%	38,810m	903m	110,200m
Offsets (outliers removed)	14.1%	866m	642m	767m

4.5 Offsets

In an attempt to improve the initial results achieved with Chan's algorithm, both in terms of the number of packets with a solution and in terms of the positional error, it was theorized that each gateway may have a timing offset associated with it which could be corrected for. Offsets cannot be calculated for each gateway individually since we are using TDOA and do not have an exact transmit time, but offsets for pairs of gateways can be found.

The offset of a pair of gateways, i and j, is shown in equation 4.1. The result of the difference of the euclidean distances from the true position of the packet, $d_i - d_j$, divided by the speed of light, C, is what the time difference of arrival should be. Therefore the difference of this value and the TDOA, $t_i - t_j$, provides a time offset for that pair of gateways.

$$offset_{ij} = (\frac{d_i - d_j}{C}) - (t_i - t_j) \tag{4.1}$$

The offset is calculated for every possible pair of gateways for every packet. The probability density function (PDF) for these offsets is shown in figure 4.4 and it can be seen that they are quite varied, with the majority of values within 10 µs and the mean at 3.58 µs. In terms of distance, that is a mean offset of over 1km.

Of greater significance are the spread of values for each individual pairing, shown in figures 4.5a and 4.5b. 4.5a shows that some pairings have mean offsets of as high as 0.06s, which would cause significant

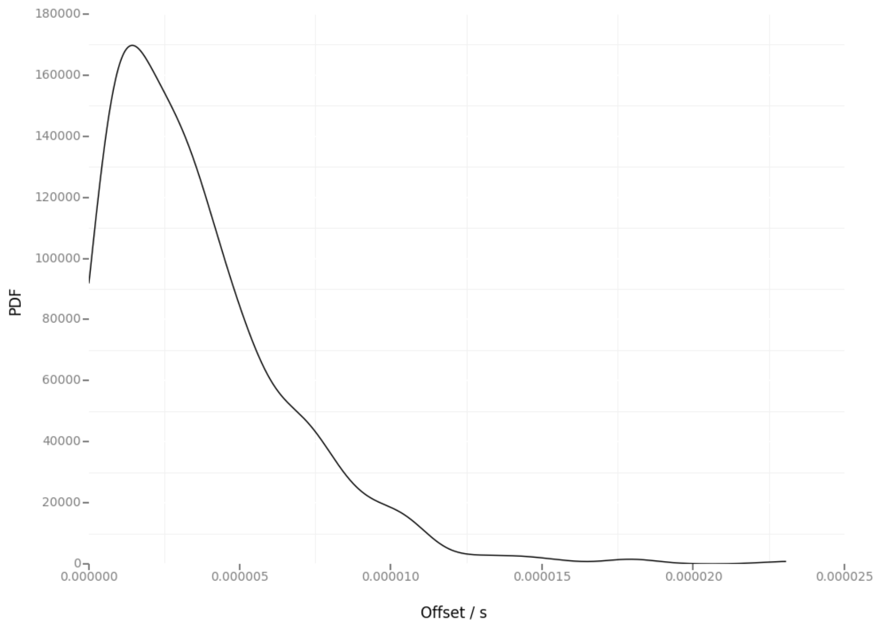


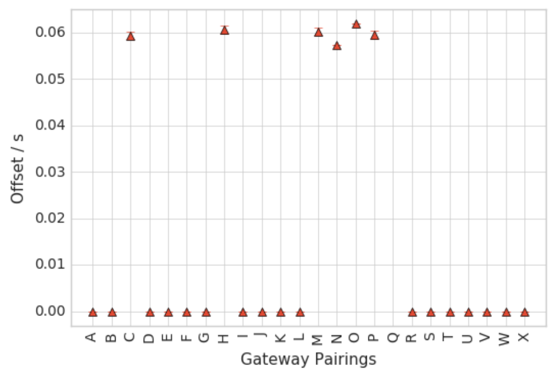
Figure 4.4: Probability Density Function for all calculated offsets

errors in positioning and likely cause no solution to be found at all. 4.5b narrows down to those pairings with offsets in the microsecond range and allows the standard deviation bars to be seen. These show that the offsets are not particularly constant for most pairings, perhaps suggesting that other factors are involved such as multipath and non-line-of-sight effects or just that the gateways do not assign a timestamp in a particularly controlled way.

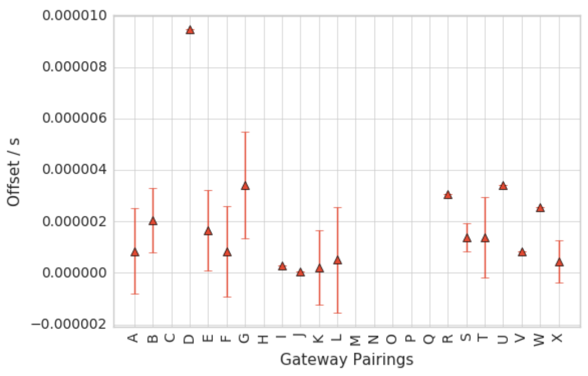
Applying these mean offsets to the TDOAs before using Chan's algorithm results in slight improvements with a solution for 17% of packets, shown in row 2 of table 4.2. Due to clear outliers however, resulting in solutions tens of kilometres from the gateways, the mean positional error is almost 40km but with a median of only 903m. Removing these outliers gives solutions for about 14% of packets with a mean positional error of 866m, shown in row 3 of table 4.2, down from an error of 1112m without applying offsets.

4.6 Iterative Algorithm

Even with the addition of offsets, Chan's algorithm severely limits the number of resolvable packets. Ideally an approximate solution is needed in the cases when the hyperbolic equations do not intersect and this is provided by utilizing an iterative algorithm rather than a closed. With the implementation of a least squares based iterative algorithm, an exact solution does not need to exist. In addition, all the data in over-determined situations can more easily be used, potentially improving the estimation accuracy - i.e. when there are more than 3 gateways that have received a packet and thus more TDOA equations than there are unknown coordinates. Being an iterative algorithm, an initial estimate is required and the further the estimate from the true solution, the more iterations it will take to converge on a solution. To keep processing times down, a limit on the number of iterations will normally be set and so ideally the initial estimate will be set to be as close to the solution as possible. With trial and error of several potential initial estimates, including using the origin (0,0) and the result of the previous estimate, it was found that using the position of the first receiving gateway minimized the number of packets which



(a) All gateway pairs



(b) Gateway pairs with µs offsets

Figure 4.5: Mean and Standard Deviations of Calculated Offsets

did not converge. The only other possible disadvantage of an iterative technique compared to a closed form algorithm such as Chan is the processing time - however, the processing is being performed on the server-side rather than an embedded device and so this an inconsequential issue.

When the iterative algorithm was run on the raw data, the results were immediately an improvement with a solution returned for about 44% of packets that were received by three or more gateways, compared to 14% with reasonable error for Chan's algorithm. As shown in row 1 of table 4.3, the mean positional error achieved was 972m, a similar figure to Chan's.

Since all receiving gateways are taken into account using the iterative algorithm, improvements were found by rejecting outliers in the receiving gateway timestamps. Erroneous times that would clearly put the node beyond the reach of the network were rejected, resulting in solutions for over 58% of packets with a mean positional error of just over 1km, shown in row 2 of table 4.3. By also applying the offsets for gateway pairs before using the iterative algorithm, solutions were found for about 55% of packets with a mean positional error of 930m and a lower standard deviation of 557m, shown in row 3 of table 4.3.

Table 4.3: Results of Iterative algorithm

	Resolvable Packets	Mean Error	Median Error	Standard Deviation
Raw Data	44.1%	972m	850m	614m
Raw Data (reject outliers)	58.2%	1080m	929m	716m
Offsets	55.3%	930m	863m	557m

To better visualize the positional error, 4.6 plots the probability density function for the iterative algorithm with offsets applied. Whilst the mean error is just under 1km, the variability of the positional error is large with errors as great as 3km occurring. Precise and reliable positioning would therefore appear to be beyond the scope of the current network, but a rough position over a broad city scenario can certainly be achieved.

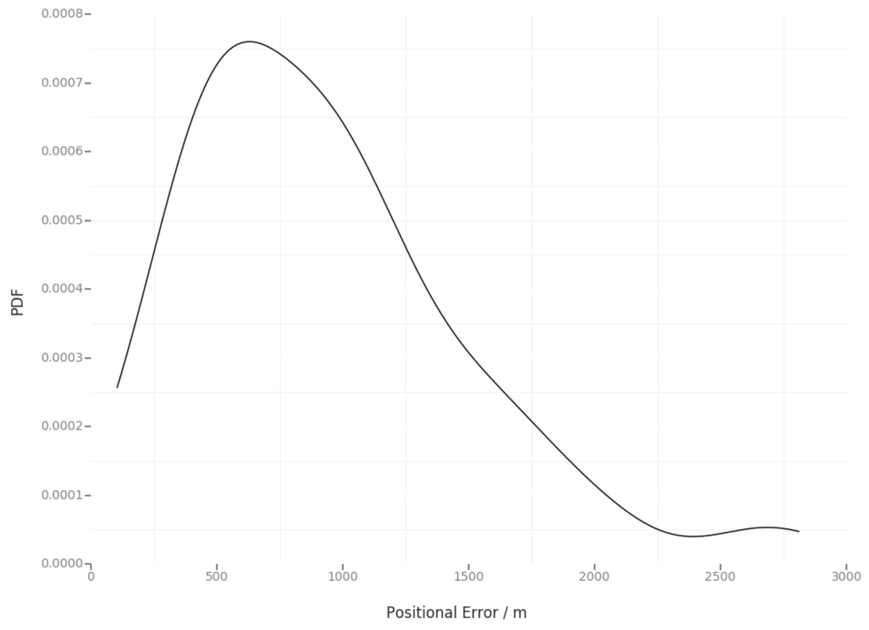


Figure 4.6: Probability Density Function for Positional Error using Iterative Algorithm with Offsets

4.7 Oversampling

Lastly, oversampling was employed in an attempt to improve the positional accuracy and reduce noise. With the test node stationary at one location known to be picked up by multiple gateways, a large number of transmissions were made. Two methods were considered to test the viability of oversampling, the first being to calculate every possible TDOA and find the mean for each gateway pair to achieve a higher resolution time difference. The second was to find an estimated location for every packet and take the mean of the coordinates as the final estimate. This latter method was chosen as simpler and likely to produce similar results.

Table 4.4: Results of Oversampling with Iterative algorithm

	Resolvable Packets	Mean Error	**Final Estimate Error**
Data	29%	1196m	**467m**
Data (reject outliers)	64%	1198m	**392m**

Running the iterative algorithm on each individual packet received from the location resulted in a mean positional error of almost 1.2km with only 29% of packets achieving a solution. However, the mean location calculated from the set of estimated coordinates achieved a far lower positional error of just 467m, shown in row 1 of table 4.4.

Running the algorithm again whilst rejecting outliers resulted in a mean positional error of almost 1.2km but with 64% of packets achieving a solution. With more packets therefore being using to estimate the final location, a positional error of 392m is found, shown in row 2 of table 4.4. These results are visualized in figure 4.7, where the individual location estimates can be seen with the final estimated location in green and true location in blue.

Whilst these results are promising and more accurate than estimating locations based on an individual transmission, this method is likely only suitable for use on stationary nodes. Whilst transmitting multiple packets in bursts is possible, it is undesirable both due to power constraints and sharing use of the network amongst potentially large numbers of other nodes. In reality, any oversampling of location will happen over longer periods of time, reducing usefulness for moving nodes which are the primary use case for positioning.

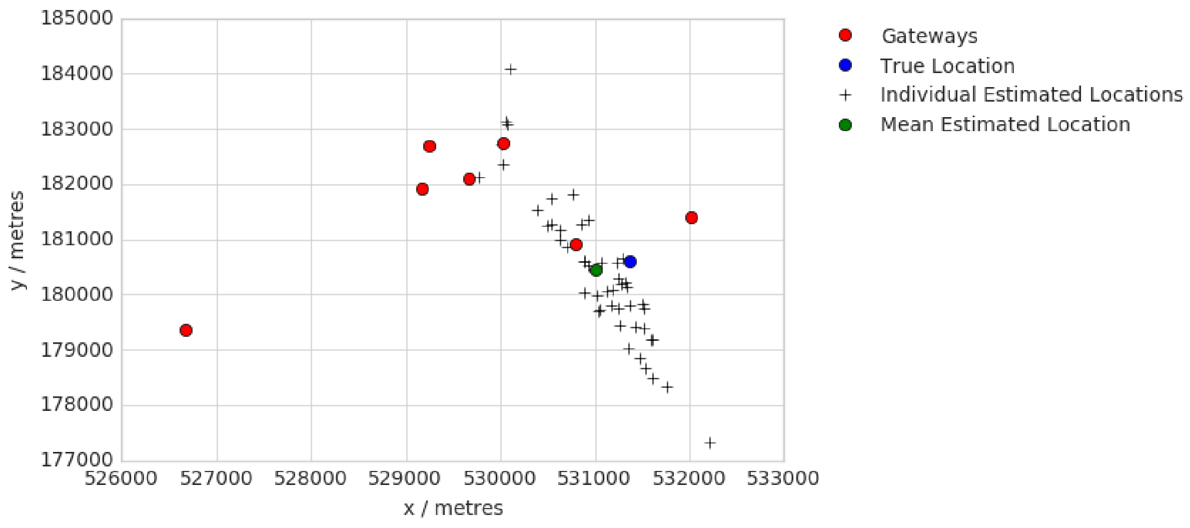


Figure 4.7: Geographical plot of oversampling with iterative algorithm

Chapter 5

Conclusion

Since only 50% of packets were received by 3 or more gateways and a further x% were rejected, leaving only y% of packets with location estimates, it is clear that the LoRaWAN network used is not ideal for positioning. However, it is important to note this is a relatively new network and that with a broader spread of gateways, these numbers are likely to be greatly improved. It should also be noted that investigating the number of receiving gateways and distance from gateways was not a primary aim and ideally more data representing a greater geographical spread around the network's gateways would be collected and thus the reliability of these statistics improved.

Ideally offsets need to be calculated on LOS only transmissions, in a large open space. Quantify the effects of NLOS/multipath. Also, cross validation should be performed with a larger dataset, else overfitting of offsets to dataset - particularly in case of gateway pairings with only a few transmissions, or even just one transmission, *optimistically biased*.

Iterative algorithm better than closed form for such unreliable, unprecise data. Might be better algorithms than least squares though? Kalman?

Indoor, short range, precise not possible with current hardware.

Future: design own gateways with proper, accurate, precise timestamping! Gateways calculate offset from one another Combine RSSI and TDOA??? Machine learning due to noisy NLOS environment??

MORE GATEWAYS, MORE WIDELY SPREAD - more chance of LOS / closer to LOS, and more 3+ gateways

Chapter 6

Example Chapter

6.1 Aims

I aim to do things, as shown in equation (6.1). I will show how to do this in Section 6.2.

$$f(x) = \dot{x} + Bu(t) + Cx(x) \tag{6.1}$$

6.2 Apparatus

50% of my study.

A diagram of how to set up the apparatus is shown in figure 6.1.

6.3 Observations

6.4 Discussion

These results can be seen in table 6.1.

Figure 6.1: Apparatus setup

Table 6.1: Predicted and measured beam deflections

Thing	thing	thing	thing
Another	another	another	another

Bibliography

[1] Everynet Core API v1.0, 2017.

[2] OSTN02, 2002.

[3] Y.T. Chan and K.C. Ho. A simple and efficient estimator for hyperbolic location. *IEEE Transactions on Signal Processing*, 42(8):1905–1915, 1994.

Appendix A

Node Firmware

Appendix B

Server-Side Software

B.1 Python Everynet API Script

B.2 Python Fetch Script

B.3 Python Processing Script

B.4 Python Export Script