KING'S College LONDON

University of London

Indoor Localization: Automatic Checkout via Deep Neural Network Based Signatures

Final Project Report

Author: Aaron Burpitt

Supervisor: Dr Osvaldo Simeone

Student ID: 1544167

April 16, 2018

Abstract

The field of indoor localization is at a much different stage of development than seemingly similar, but much larger scale positioning systems. Such global positioning (GPS for example) has become an essential part of everyday life for many in the developed world, yet the thought of precise, accurate indoor positioning systems still seems a sci-fi-esque zenith of engineering. However, recent developments in both indoor-localization research itself, and the current boom in all things machine learning have lead to the emergence of some novel techniques that have brought the aforementioned zenith within our grasp.

This report deals with locating RFID tags, and will build upon an established technique of creating a fingerprint database from the signatures of known-location reference tags, by exploring the use of stacked denoising autoencoders, a class of deep neural networks, to perform the final comparison between the received signal of the desired tag and the reference tag fingerprints in the database.

Originality Avowal

I verify that I am the sole author of this report, except where explicitly stated to the contrary. I grant the right to King's College London to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service. I confirm this report does not exceed 25,000 words.

Aaron Burpitt

April 16, 2018

Acknowledgements

I would like to thank my project supervisor, Dr. Osvaldo Simeone, for always taking the time to answer my questions regarding the project.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

In today's world, more and more are we seeing menial tasks being taken care of by machines rather than people. Automation is driven forward by a myriad of technologies, however discussion of automation is usually accompanied by ominous headlines involving Robotics or Artificial Intelligence, but far more goes into the design of complex automated systems. One such technology is that of localization, specifically small scale, indoor localization.

The following use case may be helpful in grasping the potential of the technology.
For a while now, many supermarkets have had self-checkout systems deployed in their stores. While certainly an improvement over needing one employee for each checkout, this is a somewhat rudimentary way of achieving the goal of cutting down menial labour, as it more so pushes the work onto the customer than alleviating it entirely. With this technology, a single employee can overlook a large number of self-checkouts, allowing many more customers to be "served" at a time, however full automation would require not even a single employee to be present, which of course is unrealistic whith self-checkouts for many reasons including the need to monitor possible theft, and fixing malfunctions in the checkout systems.
The next step in the process of maximizing the efficiency of the shopping process is to remove the need to manually checkout altogether, with or without an employee. The system being envisioned is that of a store that customers can enter by scanning their smartphone to register with the system and associate them with a virtual shopping basket/cart, then just pick up merchandise, and leave without ever having to think about checking out.

1.1 Project Motivation

A recently published paper (Explored in depth in the review section) follows the current trend of using machine learning for absolutely everything, and proposes the use of autoencoders, a class of neural network, to aid in the localization process.
With the aid of test data gathered from an experiment by Gradus Technologies[6], this report will explore this technique and seek to determine it's viability for real-world usage.

1.2 Scope

In any system that interfaces with the physical world, there must be considerations regarding hardware, what do we need to measure, how precisely etc.
However this report is primarily concerned with the software side of things, as we will be using established techniques on the data collection side of things, but pairing that with new developments in data science.

Chapter 2

Background and Review

2.1 Background

This chapter seeks to give the reader the proper context on RFID technology, Localization and the small portion of machine learning relevant to the project.

2.1.1 RFID technology

RFID technology, though relatively new, is fast becoming one of the most impactful developments of the past few decades, as it has uses in almost every industry and context. Many sectors already make use of RFID tags for simple things, but the possibilities are expanding rapidly.

Perhaps the first case of RFID's most common use can be traced back to 1999[22], when professors David Brock and Sanjay Sarma working at the Auto-ID center at MIT suggested the use of low-cost RFID tags to track products via placing just a serial number on the tag, which corresponded to entries into a database accessible via the internet, as opposed to storing the entire database on each tag. This also allowed some automation of previously tedious tasks, for example with the new systems, departing shipments could be automatically identified and the buyer of good notified that their goods are on the way. This decreased cost and increased utility was a major breakthrough for RFID adoption, leading to hundreds of major companies implementing them in their systems in the years following, and resulting in their current ubiquitous presence today.

2.1.2 Indoor Localisation

In terms of positioning in general, a familiar reference point for many will be GPS, which can locate nodes to within a few meters, and so achieves a similar goal to that of RFID localization, but at a scale that renders it useless for most of the use cases discussed here. There is also the factor of microwaves being attenuated and scattered by roofs and walls of buildings that the target is inside.
There are solutions proposed to this problem involving technology very unlike GPS, non-radio methods such as:

- Computer Vision

  Cameras can be used to determine the distance between the camera itself and an object being tracked, so given an accurate 3d model of an interior is provided, object locations in that interior can be determined

However this project is more so concerned with wireless Indoor Localization methods, which are discussed in the review section in greater detail.

2.1.3 Machine Learning

Machine learning, broadly, is the study and construction of algorithms that can learn from and make predictions on data.

The dataset in our case consists of readings from a ray tracing simulator, and we are attempting to compare this data with previous readings to determine which reference tag's signature is most similar to that of the tag we are trying to locate, so why can't we do this directly via something like the sum of absolute differences? The answer lies in the nature of the method being used to communicate with the tags, regardless of the specific technology being used, they all employ electromagnetic waves, which are subject to a large amount of interference.

Autoencoders are a class of neural network concerned with **reconstructing** whatever data is input, as accurately as possible. This is of use to us as it provides a way to offset the effects of data corruption, with a specialized form of autoencoder known as a **Denoising Autoencoder**, which is trained on corrupted data to simulate real world conditions, and so if trained correctly can construct a very accurate estimation of what the original data was when given corrupted input.

2.2 Review of Relevant Literature

2.2.1 Alternative Localization Methods

Range-Free:

- ToF - Time of Flight[20]:

  There are 2 classes of ToF localization systems, possible under differing hardware situations, the first requiring accurate, synchronized clocks. A signal is sent from a device of known location to the desired device, both having an accurate clock, and the departure time of the signal is compared to the actual time of arrival.

  The second type is useful in situations in which only loose absolute time synchronization is possible, Ad hoc wireless networks are an example of this. Pair-wise roundtrip TOF measurements do not require absolute clock synchronization. By sending a ranging signal and waiting for a reply, the individual clock biases are subtracted away.

Range-Based:

- RSSI - Relative Signal Strength Index[8]:

  The difference between the RSSIs of a pair of tags is used as a metric for spatial distance. Implementations are generally based on the least squares algorithm, and can improved by including things such as trilateral localization algorithms.

  In terms of actual use, the vast majority of RSSI bases systems use WIFI signals, convenient due to the low cost, and in most cases no additional hardware requirements.

  The main issue is that the RSSI value is relatively vulnerable to the influence of the physical environment, the accuracy of the technique rapidly falling off in non-line of sight situations.

- AoA - Angle of Arrival[16]:

  Difference between the angles of arrival used as metric for spatial distance between tags. Comparisons [12] between AoA and RSSI have shown AoA to be slightly more accurate, however they share many of the same drawbacks, non-line of sight etc.

  A more in-depth look at AoA can be seen in the paper cited in the title of this sub-section, where it is suggested that AoA be used in an outdoor setting instead of our use case.

2.2.2 Dude, Where's my Card?[24]

The RFID positioning system outlined in the paper differs from many of the options discussed previously in its relationship to non-line of sight situations, as it in no way detracts from the system's effectiveness, in fact it potentially increases it. The technique relies on computing the multipath profile of desired tags from data collected by an antenna array, and comparing to a set of reference tags, so we can adapt some of these aspects into this project.

The paper provides the starting point for this project via the technique outlined in section 4, which explains in depth how to acquire the multipath profile from their antenna array, this is where our use cases diverge slightly, as they use a single moving antenna to simulate a 1-dimensional array of antennas, whereas we have a 2 by 2 array of stationary antennas and will require some altering to the specified equations.

A key difference between the method discussed in this paper and that being explored in this report is the way in which multipath profiles are compared. Here, it is suggested that Dynamic Time Warping (DTW)[2] be used. DTW is a technique that was first devised for speech recognition but has seen widespread use in comparing all manner of time-series data.
The algorithm allows the measurement of similarity between temporal sequences, by calculating the optimal match between each time in the 2 sequences. This is important under the circumstances of the experiment in the paper due to the prominence of the multipath signatures being recieved potentially at different times from the same source, but much less impactful in the Gradus experiment, as the testing took place in a much emptier room.
This report seeks to establish the merit of using an autoencoder based comparison, which we will compare with the DTW based method in this paper in the conclusion.

The final location estimation utilizes a hierarchical nearest neighbours approach.
Initially, the system seeks to locate the tag using reference locations that are spaced relatively far from each other, so as to reduce the search space of tag comparisons instead of comparing every reference location. Once these widely spaced neighbours are found, the system searches the area around each of them, covering more closely spaced tags. This is repeated until no more closely spaced tags exist around the most recently found neighbour.
Whichever of these reference locations at the lowest level is found to be most similar to the desired tag's profile is taken as the location.

2.2.3 Bluetooth Low Energy Localization[4]

This paper proposes a localization system that makes use of RSSI data gathered from low energy bluetooth tags, and works similarly to the previous paper apart from one key aspect, the use of machine learning.
From the paper:

> RSSIs are unstable and have strong signal fluctuations due to fading and multi-path effects. Besides this, there could also be beacon lost during the measurement of RSSIs.

So as one of the methods of combating this, the paper introduces the idea of using autoencoders, a class of neural networks to perform the comparison of target and reference tags, while incorporating a denoising criterion so that the autoencoder is capable of making accurate reconstructions from corrupted data.
The implementation of this being the goal of this project. Testing will be done with the data provided by Gradus Technologies. The concept of having 2 discrete phases in the localization process is also presented, defined as:

- Offline phase

  Training phase, fingerprint database is constructed with position dependent parameters extracted from the measured RSSIs of reference locations

- Online phase

  Known as the localization phase. It maps the current RSSI measurements to a reference location by finding the most relevant RSSI fingerprint from the database.

The need for the autoencoder comes from the inherent instability in wireless signals in general, making comparison more complicated than some may presume, and while the data we will be using for the project isn't gathered from Bluetooth beacons so some implementation specific details will be different, the autoencoder, if it can be trained effectively, will be a greatly improved method of comparison compared to older techniques.

2.2.4 Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion[15]

This paper explores many aspects of autoencoders, ways in which to optimize them, and how they compare to models with similar uses such as support vector machines and deep belief networks.

The paper is highly concerned with exactly what constitutes a "good representation" of any particular dataset.

The Denoising Criterion

Introduces the use of a denoising criterion, extending the "good representation" concept to:

"a good representation is one that can be obtained robustly from a corrupted input and that will be useful for recovering the corresponding clean input"

A number of corruption techniques are defined, and are discussed more in-depth later in this report.
It is stated that the denoising approach makes little sense when working with data of low dimensionality; our data will have 441 input features and so is a good candidate for the technique.

Stacked Autoencoders

Establishes the use and merit of stacking autoencoders, and ways in which to do so, including pretraining which will be used for this project.
The stacked denoising autoencoder is shown to perform similary to or better than over techniques when working with common machine learning datasets such as MNIST (A large collection of handwritten digits).

Chapter 3

Requirements

3.0.1 Software

Functionality

The following are basic requirements, with which we can test the viability of the autoencoder for localization purposes.

1. Compute Multipath profile from tag data

   We receive the data as a complex number vector, and must transform it into the multipath profile of the signal so that it can be compared.

2. Autoencoder

   (a) Single Layer Autoencoder

   The first step is to build a single-layer autoencoder capable of producing an output X' from an input X.

   (b) Corrupting Input

   Next, the input X will be corrupted to simulate transmission noise, while the output should remain relatively unchanged.

   (c) Stacking Autoencoders

   More layers will be added to the neural network, which in theory will allow a more precise representation of the data to be built, allowing more accurate reconstructions.

   (d) Pretraining Procedure The technique of training a succession of multiple autoencoders with fewer and fewer hidden units, from which the weights of each layer will be taken so as to build a new multi-layer neural net.

3. Construct and save fingerprint database

   Given a set of reference tags and their signal data, the software will construct the specified type of autoencoder for each of them, and save the weights so that they can be rebuilt easily.

4. Location estimation

   To estimate the location of a tag, the software will:

   (a) Compute the multipath profile of the desired tag

   (b) Iterate over the database of reference location data

   (c) Construct a stacked autoencoder from the data saved for each

   (d) Use each autoencoder to reconstruct the desired tag's profile

   (e) Computer location estimate using K-nearest reference locations.

Accuracy

As there are many proposed RFID localization techniques, it is natural that their accuracy varies quite a bit. Most proposals including AoA and RSSI are accurate to within 1m, while the chosen method for this project, PinIt, achieves 16 cm 90th and 25 cm 99th percentile accuracy respectively[24].

This report aims to establish viability of a new method, so no concrete goal will be set for the accuracy, though a reasonable target would be similar to the current techniques as above.

3.0.2 Hardware Considerations

While the hardware portion of the system isn't the main concern of the report, some brief research into what the real-world cost of implementation would be has been done, as this would likely be the largest hurdle to adoption.

- Energy
  - RFID Tags

    There are no operational costs associated with the tags in this case, as passive UHF RFID tags operate all of their circuitry by capturing RF energy transmitted from the reader.
  - Antenna

    Studies[11] have shown the power consumption to be minimal, no more than 2.5watts, and depending on make/model, potentially much less.
- Deployment
  - RFID Tags The cost of RFID tags can vary dramatically depending on the use case, some tags that carry out multiple functions can be quite expensive, reaching prices of $20 per tag[7]. However, such complicated tags are far from necessary for the purposes of localization, in fact it is possible to simple passive UHF tags to perform what is necessary, which price in the range of a few cents.
  - Antenna

    Again, the cost here can vary, but all we need from the antenna is a signal reading, so cheap omni-directional antennas are adequate, for example the antenna used in the first paper discussed above can be found for $34.00[1].

Chapter 4

Specification

This section will give details of exactly **what** is used to implement the code that fulfills the requirements laid out previously. **How** things are implemented is detailed in the sections to come.

Considering the fact that this project revolves around machine learning, it was the main influence for language and library choices. A conventional choice would be matlab, with its indisputable track record and support for scientific tasks. However for much more than proof of concept and testing, it falls short when compared to a general purpose programming language like python. As the project is partially motivated by a real-world use case, it seems appropriate to produce software that is easily **deployable**. It would be relatively trivial the hook this project's code into any number of existing infrastructures, such as a web server, so any number of systems could be developed utilizing the localization technology.

The entirety of the code for the project is written in Python 3.6, described as the "swiss army knife of machine learning", a moniker that rang true for this project, as certain libraries allowed more time to be used planning and optimising, and less on figuring out the intricacies of the code.

All external libraries are free and open-source.
The following is a list of python libraries, how they are to be utilized (Referencing requirement numbers from the previous section), and a little background on each of them.

Python Libraries Used:

- Numpy[13]

  Considered the "fundamental package for scientific computing with Python", makes working with n-dimensional arrays easy, and provides quick access to less common mathematical functionality.

  Used whenever complex mathematical functions are required.

- Pandas[14]

  Provides data structures and analysis tools, somewhat of a programmatic Excel.

  Used to help work with some of the datasets.

- Plotly[17]

  Data visualization library, allowing many types of plots to be produced quickly with multiple common data-science languages (Python, R, matlab), and a javascript library to allow them to be displayed online.

  Used to visualize the singal reconstructions.

- TensorFlow[21]

  Quickly becoming the premier neural-network library, due to the ease at which it allows construction of "graphs" that map out whatever computation is required.

  Used for the construction of the autoencoder models.

- TensorBoard

  A visualization additon to Tensorflow, automatically creates interactive graphs to keep track on the inner workings of whatever machine learning model is being trained.

- Scikit-learn[19]

  A multi-purpose toolkit for various data-science related tasks in python.

  This project makes use of the preprocessor to normalize data.

- TQDM

  Progress Bars.

- Pickle

  Implements binary protocols for serializing and de-serializing a Python object structure, in other words allows the saving/restoring of python objects to files.

Chapter 5

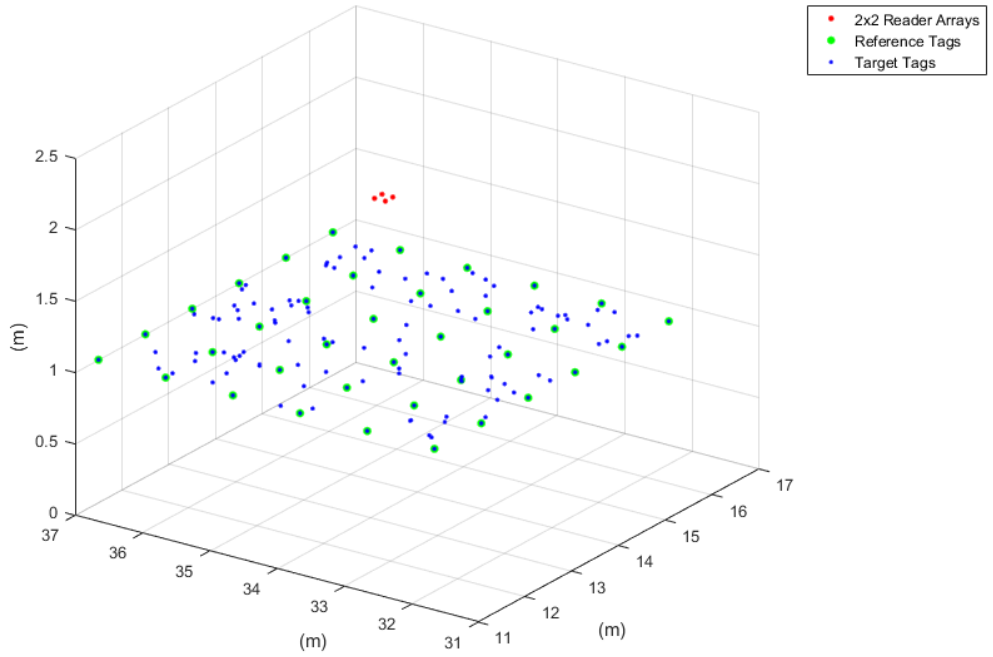
Design

5.1 Multipath Profile Computation

This section details the calculations laid out in the "Dude, where's my card?" paper that relate to the project, and how they have been adapted to our circumstances.

Conveniently, multipath profile computation is flexible and can be adapted to various permutations of environments, based on antenna set-up.

The tag and antenna set up is as depicted in the following diagram:



The paper includes details for 2 possibilities:

- 1-Dimensional antenna array

$$B(\theta) = |\sum_{k=0}^{K-1} w(k,\theta) \cdot s_k|^2$$

  where:

$$w(k,\theta) = e^{-j\frac{2\pi}{\lambda}x_k cos\theta}$$

- Single antenna moving in a single direction

$$B(\theta) = |\sum_{k=0}^{K-1} w(k,\theta) \cdot s(t_k)|^2$$

  where:

$$w(k,\theta) = e^{-j\frac{2\pi}{\lambda}t_k v cos\theta}$$

The method adapted to this paper's conditions, K x K antenna array, in our case K = 2:

$$B(\theta,\phi) = |\sum W \cdot S|^2$$

where:

$$W = w_1 w_2^T$$

$$w_1 = e^{-j\frac{2\pi}{\lambda}xcos(\theta)}$$

$$w_2 = e^{-j\frac{2\pi}{\lambda}xcos(\phi)}$$

$$x = [x_0, x_1, ..., x_k - 1]^T$$

$$x_0 = 0, x_1 = L, x_k - 1 = (K-1)L$$

$$L = \frac{\lambda}{2}$$

$$S = [s_{ij}]_{K*K}$$

s_{ij}: Channel coefficient at antenna (i, j) - from a specific tag

5.2 Autoencoder

5.2.1 Basic single hidden layer autoencoder

- The input data is mapped onto the hidden layer (*encoding*).

$$y = f(x;\theta) = \sigma(Wx + b)$$

- The hidden layer is mapped onto the output layer (*decoding*).

$$z = g(y;\theta) = g(f(f;\theta);\theta') = \sigma(W'y + b')$$

  Where:

$$x = \text{input}$$
$$y = \text{encoded data}$$
$$z = \text{decoded data}$$
$$\sigma = \text{non-linear activation function}$$
$$f(x;\theta) = \text{a function of x parameterized by } \theta$$

- The output of the autoencoder is tested against the original input using mean squared error:

$$e = \frac{1}{2m}\sum_{i=1}^{m} ||x_i - z_i||^2$$

- During training, small alterations are made to the parameters of the model via backpropogation, in an attempt to improve the reconstruction error.
  This is handled by TensorFlow's optimizers.

- The autoencoder design is known as *undercomplete*, the hidden layer has a lower dimensionality than the input and output layers.

5.2.2 The Denoising Criterion

The denoising works as follows:

- The input x is corrupted via function $q(x) = x_{corr}$
- x_{corr} is used as in the previous model
- Error computation the same.

Corruption must repeated every time the same input is fed into the autoencoder.
only corrupting once would lead to the corruptions being recognised as valid.
Corrupting differently each time avoid overfitting.

There are a number of ways to introduce noise into the data, discuss why masking noise chosen above other methods.

- Masking noise

  A percentage (often 50%) of the input data is set to 0.

- Salt and Pepper noise

  A percentage of elements of the input data (chosen at random) are set to the maximum or minimum value according to a fair coin flip.

- Additive White Gaussian noise

  A common noise model that attempts to mimic the effect of many random processes that occur in nature, involving adding random values from a gaussian distribution to the data.

Masking noise will be implemented, as the majority of testing will be done using masking noise, it seems to be most suited for use with autoencoders, as is suggested in the Autoencoder paper[15].

5.2.3 Stacking Autoencoders

- Corrupt the input x to x_{corr} using one of the corruption functions
- The deep neural network can have as many hidden layers as desired, for an n layer network:
  - First hidden layer

    This network operates as a succession of the mappings $x_{corr} \mapsto z_1$ and $z_1 \mapsto x'$, so:

    $$z_1 = \sigma(w_1 x_{corr} + b)$$

    $$x' = \sigma(w_1' z_1 + b_1')$$

    Train the network to minimize reconstruction error:

    $$L(x, x') = ||x - x'||^2$$

    The decoding layer is now dropped, to use the encoded z_1 as input to the next layer.
  - Second Hidden Layer

    Much the same as the first layer, a 3 layer neural network is constructed, with input z_1. This time there is no need to corrupt the input data, so:

    $$z_2 = \sigma(w_2 z_1 + b_2)$$

    $$z_1' = \sigma(w_2' z_2 + b_2')$$

    Reconstruction error:

    $$L(z_1, z_1') = ||z_1 - z_1'||^2$$
  - Final (nth) Hidden Layer

    This is repeated for as many iterations as there are desired layers.

    $$z_n = \sigma(w_n z_{n-1} + b_n)$$

    $$z_{n-1}' = \sigma(w_n' z_n + b_n')$$

    Reconstruction loss:

    $$L(z_{n-1}, z_{n-1}') = ||z_{n-1} - z_{n-1}'||^2$$

- Once all the layers are trained, we are left with a vector of weight bias pairs, one for each encoding layer as the decoding layers are just mirrors of their encoding counterparts.
  For the ith reference tag:

$$f_i = [(w_1, b_1), (w_2, b_2), ..., (w_n, b_n)]$$

- A number of parameters will be implemented as variables, to be altered as necessary:
  - layers
  - optimizer
  - epochs
  - batch size
  - learning rate
- Dataset Requirements
  As is the case with neural networks in general, a large dataset is required to achieve optimal results. This is easy for us to achieve as we can generate data as needed.

5.3 Fingerprinting

Once the previous sections are complete, fingerprinting will make use of much of the code produced at each stage.
It will require iterating over every reference tag's data, and performing the following:

1. Compute many random variations of the multipath profile of the reference tag
2. Construct and perform the pretraining and fitting procedures on a Stacked Denoising Autoencoder (or possibly a single-layer autoencoder) with the computed data
3. Extract the weights and bias terms from the Autoencoder
4. Saves these in a database, alongside the other reference tag data

5.4 Location Estimation

With the database described in the fingerprinting section, the final stage will involve:

1. Compute the multipath profile of the desired tag
2. Reconstruct this profile with each of the autoencoders in the database
3. For each reconstruction, carry out the following calculations from the BLE paper:
   (a) Calculate the distance between the input x and reconstruction x_i'

$$d_i = ||x - x_i'||$$

   (b) Calculate $P(x|i)$

$$P(x|i) = e^{-\frac{d_i}{\lambda}}$$

4. The list of $P(x|i)$ values is sorted, and the K (variable) largest are taken.
5. The coordinates of these reference locations are then used in conjunction with the probablities to compute a weighted average, which we will take as the final estimate of location.

The weighted average is calculated as follows, given items $x_1, x_2, ..., x_n$ with corresponding weights $w_1, w_2, ..., w_n$,
let

$$W = \sum_{i=1}^{n} w_i$$

so that new weights can be produced:

$$u_i = \frac{w_i}{W}$$

while maintaining the following:

$$\sum_{i=1}^{n} u_i x_i = \sum_{i=1}^{n} \frac{w_i}{W} = \frac{1}{W} \sum_{i=1}^{n} w_i = 1$$

so we can then calculate:

$$\sum_{i=1}^{n} u_i x_i = \sum_{i=1}^{n} \frac{w_i}{W} x_i = \frac{1}{W} \sum_{i=1}^{n} w_i x_i = \frac{\sum_{i=1}^{n} x_i w_i}{\sum_{i=1}^{n} w_i}$$

leaving us with:

$$\frac{\sum_{i=1}^{n} x_i w_i}{\sum_{i=1}^{n} w_i}$$

Chapter 6

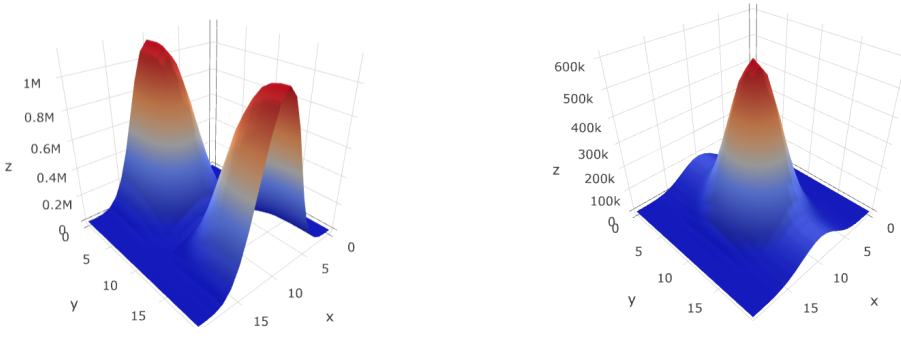
Implementation

The code for this project was written with the use of Visual Studio code[23], while testing was conducted in the Jupyter[9] notebook environment.

Many of the stages of development lend themselves to easy visualization, where appropriate figures are provided displaying initial tests at each stage.

6.1 Computing Signatures

This portion of the code was little more than translation of the equations detailed in the design. A helper function that allows large amounts of distinct data to be produced easily from the same tag was written, which is possible due to the inherent randomness in the signature computation, and necessary due to the huge data requirements of neural networks.



(a) Sample A

(b) Sample B

Figure 6.1: Example of signatures computed from the same tag data

6.2 Constructing the Autoencoder

A consideration for the code was to not only produce a viable autoencoder implementation that accomplished the goals of the project specifically, but a stand-alone, easily usable autoencoder interface that, with the correct tuning, be used for the purposes of the project, but is in no way limited to just that and could easily be made use of in other endeavours.

The input to the Autoencoder is the output of the signature computation, which is in the form of a 21x21 matrix. This will be flattened to a 1-dimensional vector of length 441 before being fed into the neural network.

6.2.1 Single Hidden-Layer Autoencoder

Figure 6.2 is the output of TensorBoard, the visualization library included with Tensorflow.

Starting from the input variable X, which is corrupted, the computation runs through the

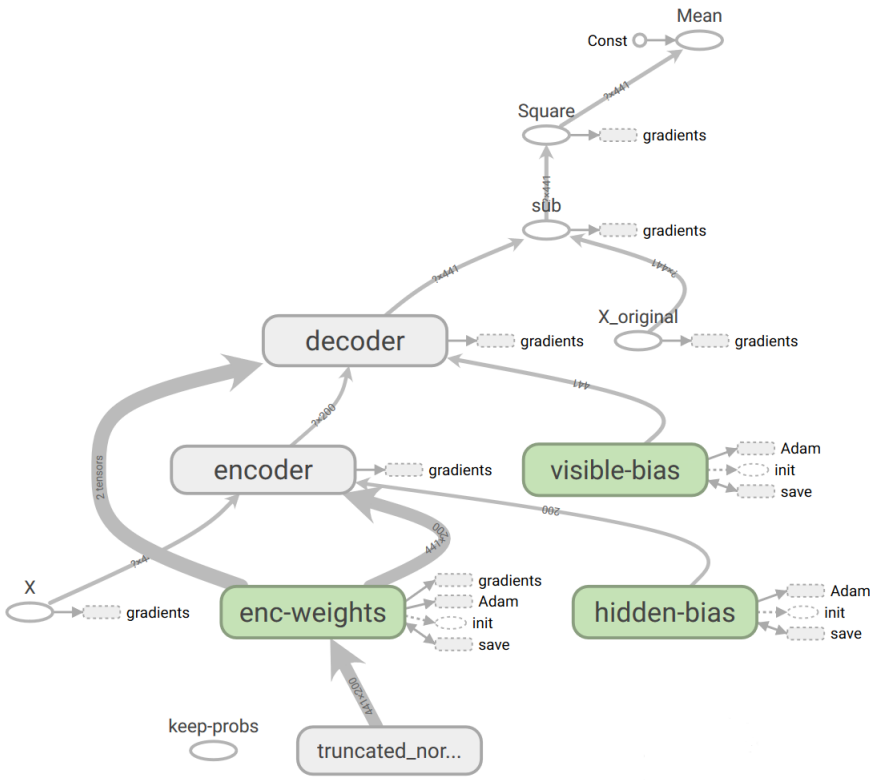


Figure 6.2: Diagram of the single hidden-layer TF graph.

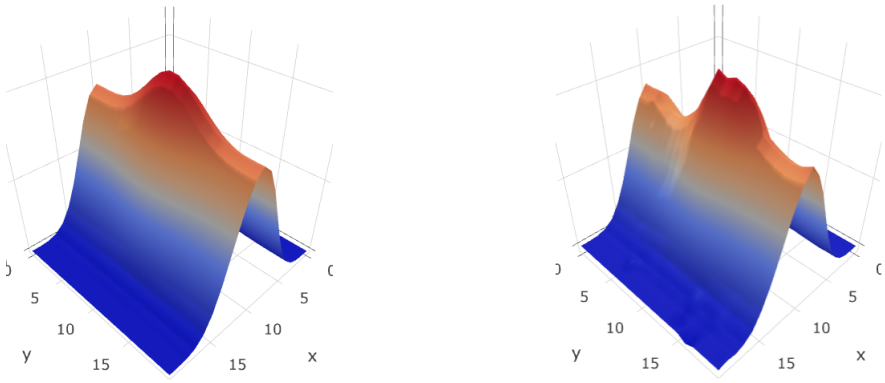
graph as follows:

1. X is encoded by the encoding layer, which is initialized with weights from a truncated normal distribution, and zeros for the bias terms.

2. The encoded X is then decoded by the decoding layer, where the weights are the transpose of the encoding layer, and again zeros for bias.

3. The resulting X' is then compared with the original non-corrupted X via mean squared error.

4. The optimizer (Adam in this case) is responsible for updating weights and bias terms after each iteration.



(a) Original Signature (b) Reconstruction

Figure 6.3: Original signal and reconstruction from basic single hidden layer autoencoder

Encoder / Decoder

The encoder and decoder blocks shown in figure 6.4 are composed of a few operations:

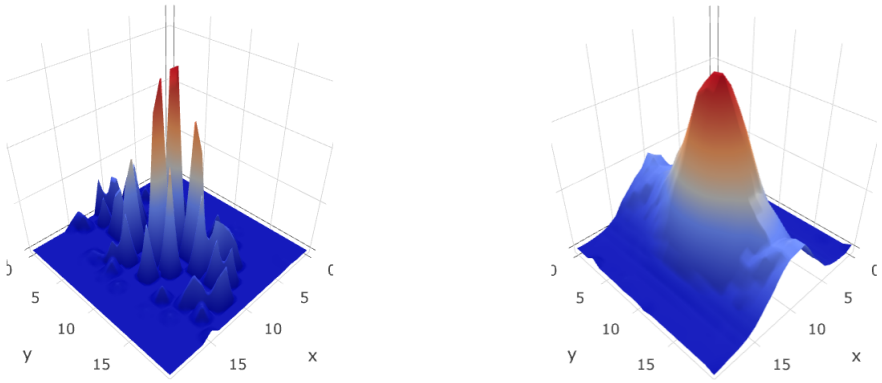
- Encoder
  1. Input X undergoes matrix multiplication with the encoding weights
  2. The hidden bias terms are then added to the result
  3. Then the result is passed through an activation function, in our case, the rectified lienar unit (or relu) was chosen.
- Decoder
  1. As the decoder is a mirror layer to the encoder, the encoding weights are transposed.
  2. Matrix Multiplication.
  3. Visible bias terms are added.

(a) Encoding Layer (b) Decoding Layer

Figure 6.4: Expanded view of the encoder and decoder from the tensorboard graph.

6.2.2 Corruption technique

In the BLE Localization paper[4], the data was corrupted via entry-wise product of the input and a vector of 0s and 1s. However in translating this to python, it makes sense to construct the corrupted data in a more imperative style, setting random cells in the input array to 0.



(a) Corrupted Signature (b) Reconstruction

Figure 6.5: Example reconstruction of corrupted input

6.2.3 Stacked Denoising Autoencoder

Pretraining

As each corresponding encode/decode layer is constructed as a seperate autoencoder initially, a function is built into the autoencoder class that returns the weights and bias terms.
So the stacked autoencoder class only has to create multiple instances of the single hidden layer autoencoder, train them as is specified in 5.2.3, and extract their weights/bias terms.

Finetuning

Since the network is instantiated with weights and bias terms that are much close to the final values than a normal network's would be thanks to the pretraining, the normal training procedure can be considered more of a finetuning phase.
Figure 6.6 is the TensorBoard output for a 3 hidden layer Stacked Denoising Autoencoder, built using the pretrained weights and bias terms from 3 single hidden layer autoencoders.

Aside from the pretraining procedure, the stacked autoencoder is very similar to the single-layer variant in its structure.

In this graph, the **keep-probs** tensor is connected to each of the layers, this is the tensor responsible for handling dropout, which may be used to prevent overfitting.

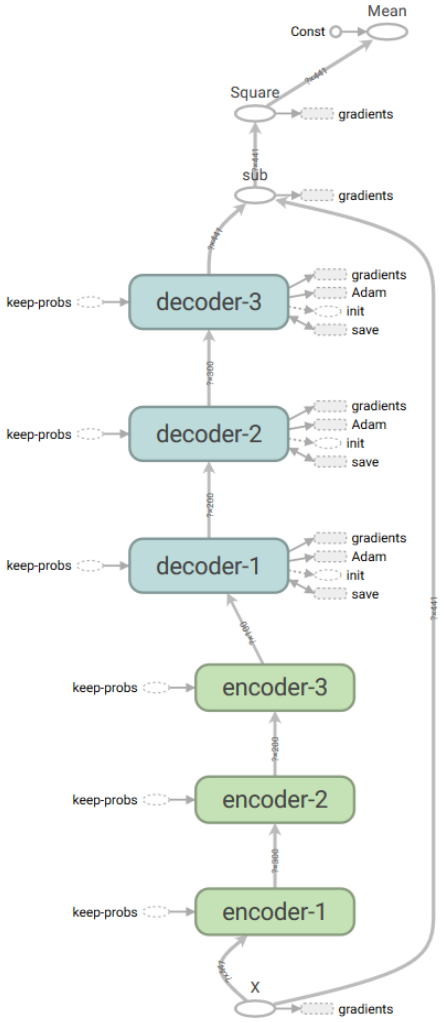


Figure 6.6: Diagram of the final TF graph.

6.3 Comparison / Weighted KNN

Translating the formulas laid out in the design section to python was made trivial by some of python's extremely succinct syntax, in this case the **list comprehension**.
This allows mathematical notation such as:

$$P(x|i) = e^{-\frac{d_i}{\lambda}}$$

That must be calculated for each element in a list to produce a new list, and so is computed with just:

```
1                    [np.exp(-1 * (d / 1/3)) for d in Di ]
```

Where each d in Di is the reconstruction error of a certain refeerence location.

6.4 Implementation Issues

Normalization

Once the autoencoder was constructed and initial testing was underway, it became apparent that the scale of the data was difficult to work, i.e. having reconstruction error values in the millions for very similar looking graphs.
This lead to later calculations resulting in values beyond python's default floating point number precision, and everything defaulting to 0.
This was remedied by normalizing all data before the autoencoders come into play, however this normalization caused the stacked autoencoder's pretraining procedure to perform poorly.
This was also remedied, by performing the pretraining procedure as usual with raw data, then finetuning with normalized data, the desired outcome was acheived.

Single-Layer vs Stacked

Upon initial testing and visual assessments of some of the reconstructions being output from the various types of autoencoders, it became apparent that there was not a huge difference in the effectiveness of single-layer vs stacked.
This is not so much of an issue, however it prompted the implementation of single-layer autoencoders into the fingerprint database module, so that testing could be done with both types.

Chapter 7

Testing

7.1 The Autoencoder

The loss function used to train the networks is that mean squared error, detailed in 5.2.1.

This section will provide testing data gathered from autoencoders with varying parameters:

- Single Layer
  - Hidden Neurons

    The number of features that the input will be reduced to, will have effects on the level of detail in the representation learnt by the network.
  - Learning Rate

    Determines how fast variables in the network change, the aim is to find a learning rate slow enough that the network converges to something useful, while remaining fast enough that it can be trained in a reasonable time.
  - Denoising Criterion

    In theory allows the network to learn a more robust representation of the data.
- Stacked Autoencoder
  - Number of Layers

    Adding more hidden layers in theory allows the network to learn the underlying structure of increasingly more fundamental attributes of the data. Also involves choice of hidden neuron count as above.

The stage at each bullet point includes all parameters listed in all previous points.

7.1.1 Single Hidden Layer Autoencoders

Each test was conducted with 1500 datapoints in the training sample.

Non-Denoising

		Hidden Neurons		
		200	100	50
LR	0.01	0.0000135618	0.0000306061	0.0000314950
	0.1	0.0005191307	0.0004789408	0.0005094618
	1	0.0005808574	0.0005314169	0.0005200315

Table 7.1: Non-Denoising Training loss values for varying parameters

Table 7.3 shows that there is little difference between learning rates of 1 and 0.1, while 0.01 can provide better performance by an order of magnitude, and works slightly better when paired with higher hidden neuron counts.
Of course this does not mean a great deal, as we have yet to introduce the corruption phase, so these results will be compared with data collected from networks trained on corrupted input.

Denoising

		Hidden Neurons		
		200	100	50
LR	0.01	0.0000165272	**0.0000146730**	0.0000165247
	0.1	0.0005019119	0.0005104115	0.0004476772
	1	0.0005712868	0.0004595273	0.0004245776

Table 7.2: Denoising Training loss values for varying parameters

(LR = Learning Rate)

So from these 2 tables we can see that training the autoencoders on corrupted data has little to no effect on their ability to make accurate reconstructions.

7.1.2 Stacked

We will we taking the best performing learning rate from 7.1.1, and varying the number of layers along with the hidden neuron count in each layer.

The input layer to every network consists of 441 neurons, for the halving operation we will truncate the result to produce an integer.

		Hidden Neuron Determination		
		Halving	Subtracting 100	Subtracting 50
L	2	0.0005077193	0.0002196721	0.0005682223
	3	0.0001574151	0.0010425992	0.0011063076
	4	**0.0001294870**	0.0006784446	0.0067398702

Table 7.3: Denoising Training loss values for varying parameters

(L = Number of Layers)

7.1.3 Single-Layer vs Stacked - Training Loss

So the best training losses achieved were:

- Single-Layer: **0.0000146730**
  - Hidden Neurons - 100
  - Learning Rate - 0.01
- Stacked: **0.0001294870**
  - Number of Layers - 4
  - Hidden Neuron Determination - Halving

Which leads us to an easy decision regarding which type of autoencoder to proceed with, the single-layer variant's training loss is an order of magnitude lower so it can be assumed it will perform better during the localization phase.

7.2 Location Estimation

7.2.1 Determining K value

This localization technique is highly dependant on the value of K used for the K nearest neighbours computation, so some testing was done to determine the best possible value.

For k values 1 to 30, localization estimates were taken for all 400 target tags, after which a mean was calculated for each value of K.
This can be seen below in figure 7.1.

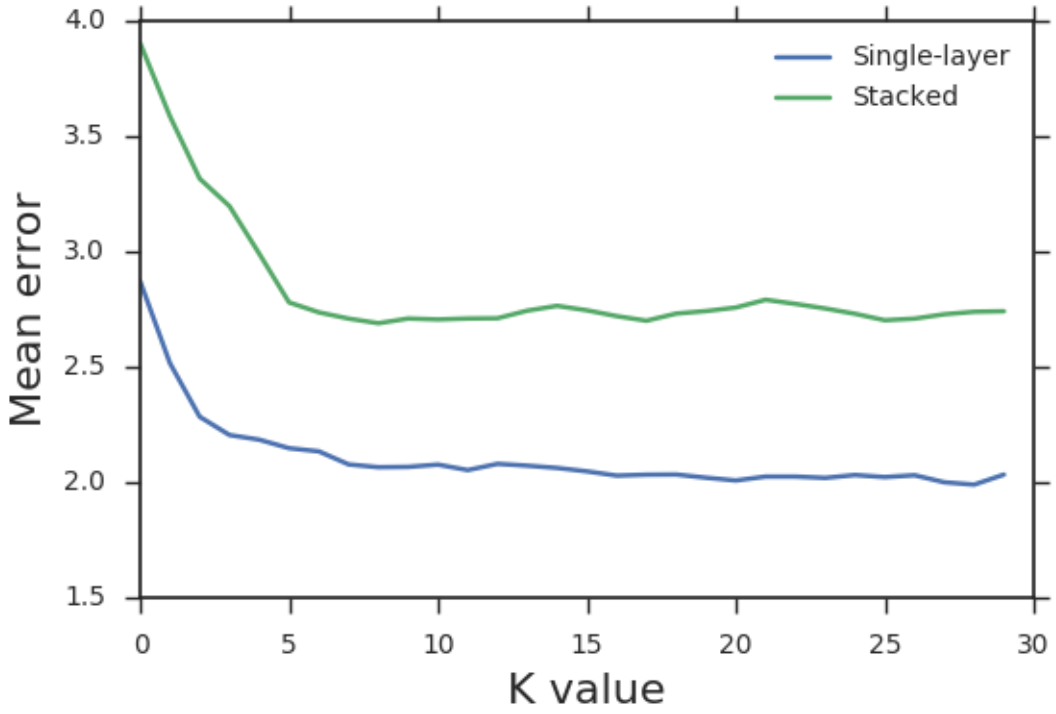


Figure 7.1: Mean localization error vs K value

As the results of the testing in the previous section suggest, error for the stacked autoencoder is higher than that of the single-layer variant across the range of k values.

For the single-layer results, after around K = 15, there is no significant change other than small fluctuations in the error, which is expected as it it similar to the results in the BLE paper[4], where the final K value was 9.
Somewhat similar can be said for the stacked results, though they begin to rise towards the end.

7.2.2 Single-Layer vs Stacked - Localization Accuracy

The tests here will be carried out with the best performing single-layer and stacked autoencoder parameter variations as determined previously. Table 7.4 contains some sample results, with K = 15 for the single-layer variant, and k = 11 for the stacked:

	Estimation - (x, y), (meters)			Error (meters)	
Tag	Single-Layer	Stacked	Actual	Single-Layer	Stacked
50	(14.45, 34.58)	(14.28, 35.82)	(12.09, 32.76)	2.979	3.758
100	(13.22, 34.33)	(14.25, 35.80)	(12.79, 33.30)	1.118	2.893
150	(14.12, 35.26)	(14.29, 35.82)	(14.26, 36.25)	0.999	0.428
200	(13.86, 34.74)	(13.84, 35.41)	(12.39, 35.13)	1.525	1.477
300	(14.25, 34.19)	(12.50, 32.21)	(14.78, 36.46)	2.335	4.825

Table 7.4: Estimation Error Samples

Error is calculated as the euclidean distance between the estimate and actual coordinates.

	% of total instances	
Error (meters)	Single-Layer	Stacked
1	12.50	10.50
2	46.00	33.50
3	87.50	59.00
4	99.75	84.75
5	100.0	97.50

Table 7.5: Percentage Accuracy

The values in table 7.5 can be compared to those given in the BLE paper, which achieved distance errors of 1.0m and 2.0m for 52.69% and 92.56%, with mean accuracy of about 1.09m.

The mean for the Single-Layer method was 2.06m, so along with the data in the table, it appears the system developed for this report is roughly half as accurate as that in the BLE paper. Stacked mean was 2.66m, so again noticably higher.

Chapter 8

Conclusion and Future Work

8.1 Comparison to established techniques

The following is a list of stats for relevant localization methods:

- RSSI[24]

  Median - 1.12m

  90th percentile - 2.3m

- AoA[24]

  Median - 0.68m

  90th percentile - 1.49m

- PinIt[24]

  Median - 0.11m

  90th percentile - 0.16m

- BLE Paper autoencoder-based RSSI[4]

  Mean - 1.09m

  92.56% - 2m

- Autoencoder integrated into PinIt (this report)

  Mean - 2.09m

  87.5% - 3m

The results of the system designed in this paper are significantly worse than all of the competing technologies listed here, the system seems to have no redeeming qualities when compared to them by any possible metric.

8.2 Conclusion on viability of technique

It appears that this specific mix of techniques, PinIt and Autoencoder comparison, is not viable in the current state. However, the aim of this project, to test the use of autoencoder reconstruction error as the technique to find the K nearest neighbours of a target tag, is somewhat of a success, as the system does in fact locate tags to somewhere nearby.

So with that, and the fact that better results were found in the original proposal in the BLE paper, it does appear that autoencoders, specifically denoising autoencoders (stacked or otherwise) may have their place in localization technologies. To fully determine their viability, much more testing would need to be done both under different conditions, and with different methods used independantly from the comparison step. So far testing has only been conducted coupling the autoencoder technique with RSSI and PinIt. Therefore it may be the case that the use of autoencoder reconstruction error as a comparison technique works better with some localization techniques than others, as it seems to work better with RSSI than PinIt. Testing with methods such as AoA may yield different results.

8.3 Future Work

This project involves the testing of relatively new technology, and establishing that it is actually viable, so naturally once that is taken care of all manner of potential uses can be implemented and extended.

8.3.1 Further extending the project for MPhil

Autoencoder Toolbox

One academic pursuit involving more work with autoencoders would be to develop a fully-fledged autoencoder toolkit, expanding on the groundwork laid during this project. This could be a useful endeavour for the scientific community as autoencoders have a vast number of applications, and there are no well established open source tools with that type of functionality.

This would involve much more work with tensorflow, expanding the autoencoder variants the interface allows to be created, and some uncoupling of the code from this project.

Integration with Other localization Methods

As mentioned in 8.2, there is potential for some of the work done in this project to be adapted for other localization methods such as AoA. The autoencoder is only one component of the localization system, and so can be integrated into any other localization system relatively easily.

Integration of Dynamic Time Warping

One key aspect of the PinIt system is the use of dynamic time warping for signature comparison, which this project sought to replace by use of autoencoders. While on the surface they are used for the same purpose, it may be possible to improve performance of the PinIt system by employing the autoencoder technique under circumstances with high signal corruption, where the denoising autoencoder does well.

8.3.2 Exploration of Commercial Application

A key motivating use-case for this project was that of an automatic shopping system, so naturally a way in which to move forward into the commercial realm while continuing research on this topic would be to actually implement such a system.
This would present many challenges not faced during this project, as much of the work would be finding a way to utilize the technology with normal people without causing too much hassle, such a system could operate as follows:

1. Shopper enters the store by scanning an app on their smartphone
2. The system begins tracking this customer as they walk around the store via their phone, based on whichever reference location they are nearest
3. As they pick up items, this location data, paired with location data from item packaging allows the system to accurately determine who picked up what and so can add or remove items from a virtual basket as necessary
4. The customer leaves the store once finished, and is automatically charged via the information attached to their account in the app they scanned to get in with.//

Chapter 9

Legal, Social, Ethical and Professional Issues

This project in no way makes use of data originally or directly collected from humans, and the report creation has no tangible effect on humans so no ethical clearance is required from the College Research Ethics Committee (CREC).
However, due to the nature of the technologies developed and explored throughout the project, some ethical concerns do exist regarding potential applications of the technology and it's impact on individuals and society as a whole.

British Computing Society Code of Conduct[3]

The British Computing Society lays out a set of codes intended to ensure IT professionals have a clear set of guidelines to follow when undertaking large scale projects.
Wherever necessary throughout the project, these corresponding guidelines were followed.
Note that very few of these are relevant to the project.

9.1 Potential Impact on Society

Automation

As far as automation goes, building systems capable of performing tasks traditionally carried out by human works inevitable displaces those human workers from their jobs. Of course this is not a new occurrence, with many companies including Tesla Motors relying heavily on automated production lines[5], which provides us with insight as to how these technologies will effect people. It seems as though they result in a redistribution of jobs, moving many positions from the manual labour realm over to more technical roles involved with the operation and upkeep of the automation machinery. So while an immediate effect may be mostly negative in the eyes of some, the net result will be that of removing people from dangerous working conditions (such as in amongst heavy machinery on a factory floor), and placing them in safer, higher paying (in theory) positions.

Artificial Intelligence

The use of artificial intelligence in this report does not carry with it any of the sorts of concerns involved with some highly publicized technologies, such as self-driving cars and their potential for bodily harm[18], or even less conspicuous issues such as potential discrimination[10].

Machine learning in the context of this paper is little more than a comparison technique, and so has no potential to cause harm.

Big Data

Many headlines today relate to big data, and the ethics of tracking large populations of people, especially when that data leaks[12]. Of course all applications of localization technology will involve tracking, however the issues lie with the ways in which that data can be used in addition to the explicit purpose of localization, targeted advertising based on certain movement patterns within a space such as a retail environment would be concerning to many.
So in this regard, the technology discussed in this report has no intrinsic unethical aspects.

Bibliography

Other Sources

[1] URL: https://www.ettus.com/product/details/VERT900.

[2] Abdullah Mueen and Eamonn Keogh. *Extracting Optimal Performance from Dynamic Time Warping*. URL: http://www.cs.unm.edu/~mueen/DTW.pdf.

[3] *BCS code of conduct*. URL: http://www.bcs.org/category/603.

[4] Chao Xioa, Daiqin Yang, ZhenZhong Chen, Guang Tan. "3-D BLE Indoor Localization Based on Denoising Autoencoder". In: *IEEE Access* (2017).

[5] Matthew DeBord. *Tesla's future is completely inhuman — and we shouldn't be surprised*. URL: http://uk.businessinsider.com/tesla-completely-inhuman-automated-factory-2017-5. (accessed: 05.04.2018).

[6] *Gradus Technologies*. URL: http://www.gradustech.com.

[7] *How much does an RFID tag cost today?* URL: https://www.rfidjournal.com/faq/show?85.

[8] Jungang Zheng, Yue Liu, Xufeng Fan, Feng Li. "The Study of RSSI in Wireless Sensor Networks". In: *Advances in Intelligent Systems Research* 133 (2016).

[9] *Jupyter website*. URL: http://jupyter.org/.

[10] Erica Kochi. *AI is already learning how to discriminate*. URL: https://work.qz.com/1227982/ai-and-discrimination-what-tech-companies-can-do/. (accessed: 05.04.2018).

[11] Marinos Argyrou, Matt Calder, Arsham Farshad and Mahesh K. Marina. *Understanding Energy Consumption of UHF RFID Readers for Mobile Phone Sensing Applications*. Tech. rep. University of Edinburgh, 2012.

[12] Sam Meredith. *Facebook-Cambridge Analytica: A timeline of the data hijacking scandal.* URL: https://www.cnbc.com/2018/04/10/facebook-cambridge-analytica-a-timeline-of-the-data-hijacking-scandal.html. (accessed: 05.04.2018).

[13] *Numpy website.* URL: http://www.numpy.org/.

[14] *Pandas website.* URL: https://pandas.pydata.org/.

[15] Pascal Vincent, Hugo Larochelle, Isabelle Lajoi, Yoshua Bengio, Pierre-Antoine Manzagol. "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion". In: *Journal of Machine Learning Research 11 3371-3408* (2010).

[16] Paweł Kułakowski, Javier Vales-Alonso, Esteban Egea-López, Wiesław Ludwin. "Angle-of-arrival localization based on antenna arrays for wireless sensor networks". In: *Computers and Electrical Engineering* (2010). URL: https://old.kt.agh.edu.pl/~brus/cee10.pdf.

[17] *Plotly website.* URL: https://plot.ly/.

[18] Sam Levin and Julia Carrie Wong. *Uber Self-Driving Car Kills Woman.* URL: https://www.theguardian.com/technology/2018/mar/19/uber-self-driving-car-kills-woman-arizona-tempe. (accessed: 09.04.2018).

[19] *Scikit-learn website.* URL: http://scikit-learn.org/stable/.

[20] Kistofer S.J. Pister Steven Lanzisera David T. Lin. *RF Time of Flight Ranging for Wireless Sensor Network Localization.* Tech. rep. University of California, Berkeley, 2006.

[21] *Tensorflow website.* URL: https://www.tensorflow.org/.

[22] Bob Violino. *The History of RFID Technology.* URL: http://www.rfidjournal.com/articles/view?1338/2. (accessed: 05.04.2018).

[23] *Visual Studio Code website.* URL: https://code.visualstudio.com/.

[24] Jue Wang and Dina Katabi. *Dude, Where's My Card? RFID Positioning That Works with Multipath and Non-Line of Sight.* Tech. rep. Massachusetts Institute of Technology, 2013. URL: http://groups.csail.mit.edu/netmit/6.888/www/papers/pinit.pdf.