A Node Localisation System for High-Performance Computing Facilities

Thavish Annal
For the degree of MPhys in Physics and Astronomy
Supervisors: Prof. Richard Bower & Dr Alastair Basden
Department of Physics, Durham University
Submitted: April 24, 2020

Abstract

We present the design and development of a decentralised wireless localisation system for nodes in high-performance computing (HPC) facilities. Ten Bluetooth Low Energy (BLE) transceivers (Adafruit Feather nRF52840 Express microcontrollers) are used to construct a prototype network that can be retrospectively fitted to nodes. We propose recovering relative transceiver positions by comparing the relative strengths of received signals across the network. When averaging repeated measurements of received signal power P(dBm), Gaussian kernel density estimation is used with a bandwidth of 0.65 dBm to filter out secondary multipath signals. By running five transceivers in a rack over a 30 hour period, we measure the uncertainty on P(dBm) due to slow-changing random noise to be 0.3 dBm. We discuss our reordering algorithm, designed to recover the relative positions of N transceivers from an unordered $N \times N$ matrix plot of P(dBm) values. The last stage of this algorithm uses single-linkage hierarchical clustering to sequence transceivers. We observe that our prototype is capable of correctly recovering physical order when ten transceivers are spaced at intervals of 2 U or less. Using data from multi-rack configuration tests, we examine how reordered matrix plots can indicate which rack a transceiver is in. We derive an empirical signal propagation model that accounts for random noise. Its key parameters are a path loss exponent n, received signal power P_0 (dBm) at a distance $d_0 = 1$ U, and the standard deviation σ_p of a zero-mean Gaussian random variable. Our model is fit to four datasets using χ^2 minimisation; the resultant average values of n, P_0 (dBm), and σ_p are 1.83 ± 0.06, -15.8 ± 0.8, and 2.7 ± 0.1, respectively.

Contents

1. Introduction

First introduced in the 1970s, high-performance computing (HPC) has become a fundamental tool in scientific research. Modern HPC facilities, often termed supercomputers, are designed to rapidly execute data-intensive workloads and high-precision floating-point arithmetic [1,2]. This is typically achieved via massively parallel system architecture; coordinated tasks are split between nodes interconnected on a high-bandwidth network [3,4]. To help optimise floor space and access in a facility, these nodes, sometimes called servers, are mounted in equipment racks.

Racks are built in standard 19-inch (48.26 cm) wide frames [5]. Equipment height is measured in discrete rack units (RU or U) where, by definition, 1 U equals 44.45 mm. Standard racks are 42 U tall, while constituent nodes are ordinarily 1 U to 4 U high [5,6]. As a facility undergoes gradual maintenance and expansion, nodes tend to move between racks, often in an arbitrary order. Efficiently locating relevant hardware during system failures can help greatly reduce computational downtime [7].

However, the sheer number of densely packed racks in an HPC facility makes keeping track of node locations highly non-trivial. For context, Summit, the world's fastest supercomputer, has 4608 nodes spread across 520 m^2 of floor space [8]. Tracking rack equipment is a challenge shared by data centres, where thousands of servers need to be monitored for resource allocation, hardware maintenance, and regular audits [9]. These tasks are particularly important in colocation centres, i.e. shared facilities which rent out physical rack space to multiple clients, and house privately owned servers [10].

General indoor localisation is an active field of research [11–13]. Traditional positioning systems like satellite trilateration lack the precision and signal strength required to function indoors [11, 13]. In data centres, metal racks create highly attenuating and complex environments. To work effectively, wireless tracking systems need to overcome the effects of strong signal interference and multipath propagation [14].

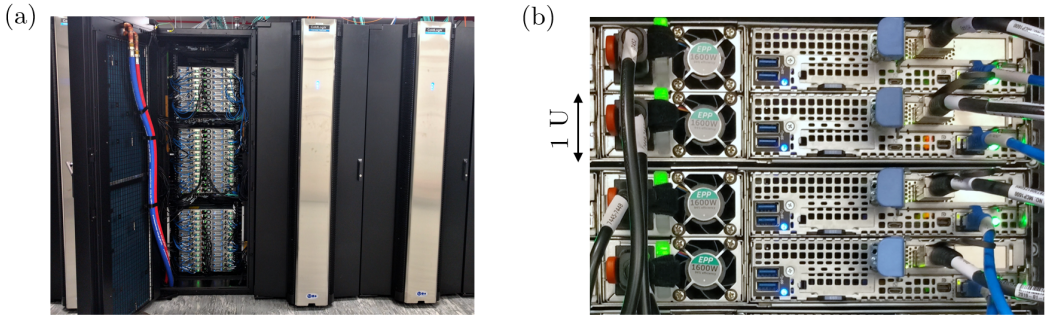

Figure 1.1: (a) Three racks in Durham University's HPC facility, the Cosmology Machine (COSMA). The leftmost rack door is open, revealing nodes used to test the prototype developed in this work. **(b)** Four nodes in a rack, each of height 1 U. Standard racks have 42 fixed mounting 'slots', meaning nodes occupy a discrete set of vertical positions.

Several existing tracking systems make use of radio frequency identification (RFID) tags [9, 14, 15]. These short range devices can be affixed to individual nodes, and do not need to be directly in sight when read. In data centres, RFID tags are either manually scanned [15], or collectively detected by an array of stationary readers [9]. The former method is slow and labour intensive, while the latter fails to provide sufficiently granular data, since the dense deployment of readers is normally cost prohibitive. To reduce deployment costs, one vendor has substituted its RFID scanning arrays with a single reader mounted on an autonomous mobile robot [14].

Alternatives to RFID include simple barcodes, as well as non-wireless systems, like physical cables connecting electronic tags to racks [16]. Unfortunately, none of these solutions provide cost-effective methods of immediate node-level localisation (at a precision of 1 U). When localisation is not performed across nodes simultaneously, positional data can quickly become obsolete, especially in large facilities [9, 16]. A reliable tracking system is essential to any kind of optimisation that is based on the physical layout of equipment.

In this work, we present the design and development of a low-cost tracking system for HPC facilities. We construct a prototype network of Bluetooth Low Energy (BLE) transceivers which can be retrospectively fitted to nodes. Powered by a Universal Serial Bus (USB) port, each transceiver measures the signal strength of transmissions from its neighbours; we examine how the resultant data can be reduced to generate a map of relative positions. To the best of our knowledge, this marks the first attempt at using a decentralised wireless system to achieve node-level localisation in a rack. Much attention is paid to all stages of the data pipeline, including hardware design (Chapter 2), signal processing (Chapter 3), and ordering algorithms (Chapter 4). We test our prototype in an active HPC facility, and fit an empirical propagation model to the data. Finally, we review our system, and briefly discuss steps towards commercialisation.

2. Transceiver Development

2.1 Adopting BLE for Localisation

We begin with an overview of the wireless technology used in this work. Bluetooth Low Energy (BLE) is a short-range radio communications standard which operates in the unlicensed 2.4 GHz industrial, scientific, and medical (ISM) frequency band [11,17,18]. Freely available worldwide, the ISM band is used by several popular wireless systems, including the IEEE 802.11 standard (commonly known as Wi-Fi). BLE devices are notable for their low power consumption, affordability, and small physical profile. Drawing a current of approximately 15 mA at peak transmission power, the average BLE transceiver can run off a 3 V coin cell battery for years [18–20]. BLE devices are often seen in personal networks, proximity detection mechanisms, and — of interest here — indoor positioning systems [11, 12, 18].

Most BLE localisation techniques fundamentally involve monitoring the intensity of radio transmissions. Given a reliable signal attenuation model, it becomes possible to calculate device distance by measuring received transmission strength. The simplest models assume free-space path loss, while more advanced logarithmic models consider shadowing and multipath effects [21, 22]. When signals are transmitted by three or more 'anchor' devices at known coordinates, the position of a receiver can be estimated using basic trilateration [11–13].

A second approach, called fingerprinting, works in two stages. First, received signal strengths from an array of anchor devices are recorded at points across an exhaustive physical grid [18]. This 'fingerprint' database acts as a map; localisation is performed by comparing a receiver's vector of measured signal strengths to the existing database, often using probabilistic methods or neural networks [11, 12, 18].

Regrettably, neither signal-modelling nor fingerprinting algorithms are quite suited to a rack's complex environment. Metal equipment can create unpredictable propagation paths, making devices appear further than they are [14]. Creating a fingerprint database for every rack in a facility would be impractical, particularly since even minor layout changes (like the movement of cables) can cause fingerprint quality to degenerate [11]. Most range-based BLE positioning schemes are accurate to the nearest metre [11, 12, 18, 20], a far cry from the 1 U requirements of node-tracking.

When designing our prototype, we aimed to create a scalable decentralised tracking system that would permit the free movement of nodes in a facility. To reduce set-up time and deployment costs, our transceivers were required to function without the aid of additional hardware, namely fixed BLE anchors. Our prototype therefore makes no attempt at achieving localisation through conventional modelling and mapping techniques. Instead, it compares the *relative* strengths of received signals to determine node proximity. Here we have made the simplifying assumption that signals from different transceivers undergo approximately equal levels of attenuation over a given distance in a rack.

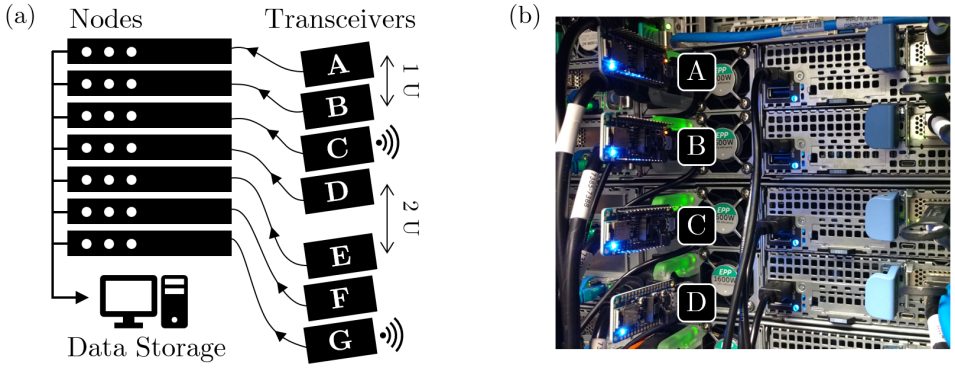

Figure 2.1: **(a)** A network diagram of our prototype. Each transceiver (labelled alphabetically) is connected to an individual node. They have been spaced in integer multiples of a rack unit U. Measurements of received signal strength pass through the nodes into a central storage unit, where the data are processed. **(b)** Four transceivers connected to nodes via micro-USB cables. Their antennas (next to the blue LEDs) face outward.

In other words, though it may not be possible to accurately predict the numerical value of received signal strength in complex conditions, positions can still be inferred from the relative intensity of transmissions, provided they have travelled through the same environment. BLE ad hoc networks function on a similar principle [23]. Our prototype works towards node-level accuracy by exploiting standardised rack design, i.e. nodes are arranged vertically along a discrete set of possible points.

Consider a rack in which each node is connected to a broadcasting transceiver, as displayed in Figure 2.1. For now, assume devices have equal initial transmission powers. We can then compare the strengths of the signals received by any one transceiver to differentiate its neighbouring devices from those several rack units away. If each device reports the strengths of incoming transmissions from all other devices, the resultant data can be collectively reduced and sorted to recover relative physical order. A rack's discrete structure allows us to translate relative order into corresponding vertical positions, a process that is detailed in Section 4.1.

This section has focused on BLE transceivers and signal-strength localisation techniques. There are various other positioning systems, and it is worth mentioning methods which rely on different signal characteristics. A receiver's position can be triangulated by measuring the time of arrival (TOA) or angle of arrival (AOA) of an incoming transmission [11–13, 24]. The former demands precise synchronisation, while the latter requires a directional antenna; neither would be practical in a low-cost decentralised network. Ultra-wideband (UWB) indoor positioning systems make use of TOA triangulation, and have an average accuracy of less than 10 cm [11, 13, 24]. Despite the potential increase in performance, we opted to avoid UWB hardware due to its high cost.

2.2 Device Communication

The BLE standard operates across 40 fixed-frequency channels in the 2.4 GHz ISM band. Each channel has a 2 MHz bandwidth, within which data are transmitted via Gaussian frequency-shift keying (GFSK) modulation [25–27]. There are two kinds of BLE channels, each designed to serve different signal classes. Advertising channels carry undirected broadcasts, while data channels allow for two-way communication and larger data packets [25,27]. Recall how several other wireless standards utilise the ISM band. As displayed in Figure 2.2, BLE advertising channels (numbered 37, 38, and 39) are spread out to reduce potential interference. Advertising packets are transmitted on all three channels to increase the probability of successful delivery [25, 26]. These undirected broadcasts are typically used to discover devices and establish data connections.

When BLE devices communicate, they take on roles defined in the Generic Access Profile (GAP). This is the protocol layer responsible for regulating device interaction, and must be adhered to by all BLE systems [25]. There are four specific device roles set out within the GAP, the simplest two being the Observer and the Broadcaster. As their names suggest, Observers scan advertising channels, while Broadcasters transmit advertising packets [25, 28]. This is illustrated in Figure 2.3.

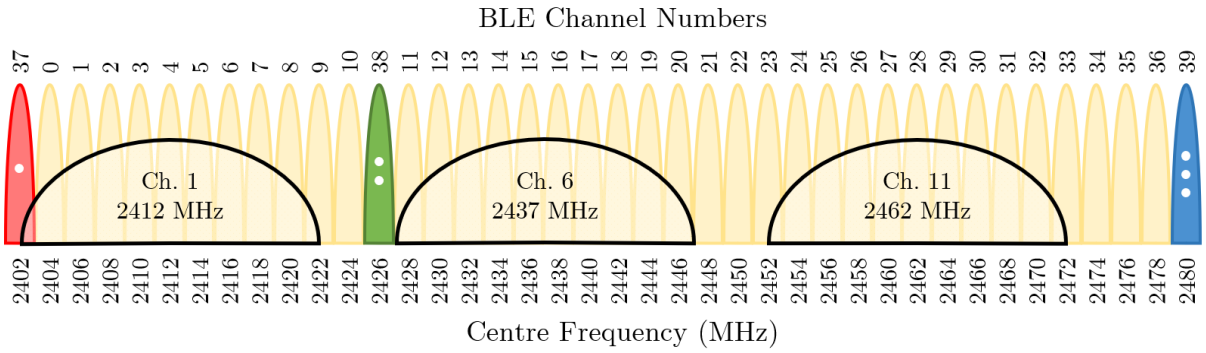

Figure 2.2: The centre frequencies of the BLE standard's 40 channels, each of bandwidth 2 MHz. There are thirty-seven data channels (numbered 0–36) and three advertising channels (numbered 37–39). The primary Wi-Fi channels 1, 6, and 11 share the ISM band; the large black semicircles denote their 20 MHz bandwidth. BLE advertising channels are spread out to reduce potential interference with Wi-Fi and other wireless standards.

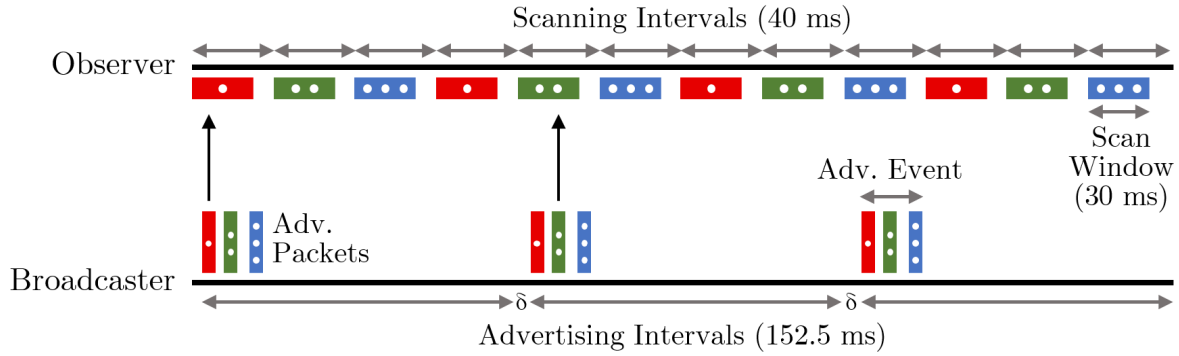

Figure 2.3: The transmission of advertising packets from a Broadcaster to an Observer. Advertising (adv.) channels 37, 38, and 39 have been distinguished by one, two, and three dots respectively. The listed durations match the timing parameters used by our BLE transceivers. The Broadcaster's controller adds a pseudo-random delay δ of less than 10 ms in between each advertising interval. Notice that the last advertising event is unsuccessful; none of the broadcasted packets match the channel being scanned. The two devices are unsynchronised. Time has been drawn to scale.

An Observer's primary parameters are its scan interval and scan window, which control the frequency and duration of scans respectively [25, 29]. Scanning is cyclic, in that exactly one channel is observed per interval. A packet must arrive within the scan window of its corresponding channel to be received by an Observer [25]. The two devices operate in an unsynchronised manner — data can be broadcast with no need for acknowledgment. During an advertising event, the Broadcaster transmits identical packets on each advertising channel. This redundancy helps maintain communication even if there is significant interference present on two of the three channels [26].

Figure 2.3 exemplifies how in most BLE systems, the scanning interval is far shorter than the advertising interval [25, 28]. This means an Observer can cycle through channels relatively quickly, and thus receive packets from multiple Broadcasters. If two devices advertise simultaneously, their packets may be corrupted due to mutual interference [17].

To ensure that this does not happen indefinitely, a pseudo-random delay of less than 10 ms is introduced in between each advertising interval [25, 29]. A BLE device's controller can be programmed to automatically interleave the roles of an Observer and a Broadcaster; this is what allows transceivers to both scan and advertise when part of a decentralised network. In accordance with recognised timing standards [30], we adopted scanning and advertising intervals of 40 ms and 152.5 ms, respectively.

BLE advertising packets are made of four components. In order, these are the 1 byte preamble, 4 byte access address, 2–39 byte protocol data unit (PDU), and 3 byte cyclic redundancy check (CRC) [25, 26]. The CRC is a checksum which verifies that packets have not been corrupted during transmission [31]. In the BLE standard, any advertising packet with a bit error rate greater than 0.1% is discarded upon receipt [25]. The PDU contains a customisable 37 byte payload which lists device-specific information. In our prototype, payloads are each encoded with a universally unique identifier (UUID) and local ID. Transceivers share a common UUID, which serves to differentiate our network from other BLE devices operating in the area. Conversely, each transceiver's local ID is a distinct name, e.g. Alice (A), Bob (B), Carl (C). These are used to differentiate between transceivers, and determine the origin of received advertising packets.

2.3 Transceiver Hardware

Formally, a radio transceiver is a wireless communications device where common circuitry is shared between a transmitter and receiver — its name is a portmanteau of the two integrated components [32]. When first selecting a BLE transceiver, our aim was to find an affordable USB-powered device with a small physical profile. We wanted a development board with native USB support, which would be simple to program and connect to a node. The Adafruit Feather nRF52840 Express microcontroller (the Feather board hereafter) fit these criteria [33]. Displayed in Figure 2.4, the board measures 51 mm × 23 mm × 7 mm, and can fit comfortably alongside nodes in a rack (recall Figure 2.1b). A total of ten Feather boards were acquired to construct our prototype system.

The board connects to a node via a micro-USB cable, by which it exchanges data and draws power. Its BLE functionality is managed by the nRF52840, an integrated system-on-a-chip (SoC). With 1 MB of flash memory, 256 kB of RAM, and a 32-bit processor [19, 33], the SoC is responsible for running the Feather board's main routine. This customised program implements specific advertising and scanning intervals, assigns transmission payload content, and determines whether received data packets should be recorded. The program's source code is available in Appendix A.

As previously noted, BLE advertising packets are frequency-modulated 2.4 GHz radio signals. The Feather board transmits and receives these signals using a ceramic chip antenna [33, 34]. Ceramic antennas are smaller and less susceptible to environmental noise than their metallic trace antenna counterparts [35]. The board's transceiver is a half-duplex system, i.e. its transmitter and receiver share an antenna, and cannot function simultaneously. This is why advertising and scanning intervals must be interleaved.

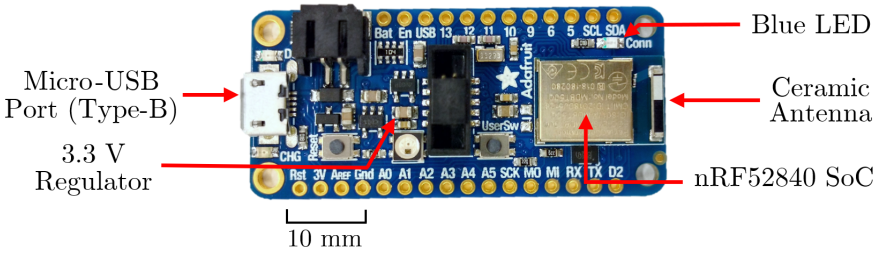

Figure 2.4: The transceiver used in this work, an Adafruit Feather nRF52840 Express microcontroller. Its key components have been labelled. The device weighs six grams.

When the antenna receives an advertising packet, its receiver's analogue-to-digital converter (ADC) samples the signal and returns a digital discrete-time measurement of voltage. During a sampling period, successive voltage measurements are filtered by an infinite impulse response (IIR) filter. This yields a value for the power of a received packet, calculated using the receiver's input impedance [19, 36]. To reflect the large dynamic range of radio transmission strengths, signal power P is measured in decibel-milliwatts (dBm). This is a logarithmic unit defined by the following equation [21, 36]:

$$P(\text{dBm}) = 10 \log_{10} \left[\frac{P(\text{mW})}{1\,\text{mW}} \right]. \tag{2.1}$$

Here, $P(\text{dBm})$ is the ratio of signal power in milliwatts $P(\text{mW})$ to one milliwatt, expressed in decibels. It is often called the received signal strength indicator (RSSI), though this term can sometimes refer to empirical scales of transmission power instead [37].

The Feather board measures BLE signal power at a resolution of 1 dBm. Its receiver reportedly demonstrates a linear response between -90 dBm and -20 dBm [19, 34]. In the following chapters, we will see that measurements of $P(\text{dBm})$ in racks typically fall within this range. When an advertising packet is received, $P(\text{dBm})$ is recorded together with the local IDs of the corresponding transmitting and receiving devices. We wrote a program (available in Appendix B) to enable serial port communication between nodes and devices. This is how the recorded data above are transferred to a central storage unit for further processing (see Figure 2.1a).

To improve the signal-to-noise ratio of transmissions, each device in our network has been set to broadcast at $+8$ dBm, which is the transmitter's maximum output power. Our prototype works by comparing measured values of $P(\text{dBm})$, and is thus dependent on devices having equal transmission powers. The Feather board has a 3.3 V voltage regulator which — assuming a steady load impedance — ensures that the power supplied to a transmitter is maintained at a constant level. We attempted to verify this experimentally by varying the input voltage across a board.

Two transceivers were separated by 20 cm in free space; one acted as a transmitter, and the other as a receiver. The acting transmitter was connected to the circuit displayed in Figure 2.5a. By changing the resistance of the potentiometer, we were able to alter the input voltage V across the transmitting device. First, V was varied between 2.5 V

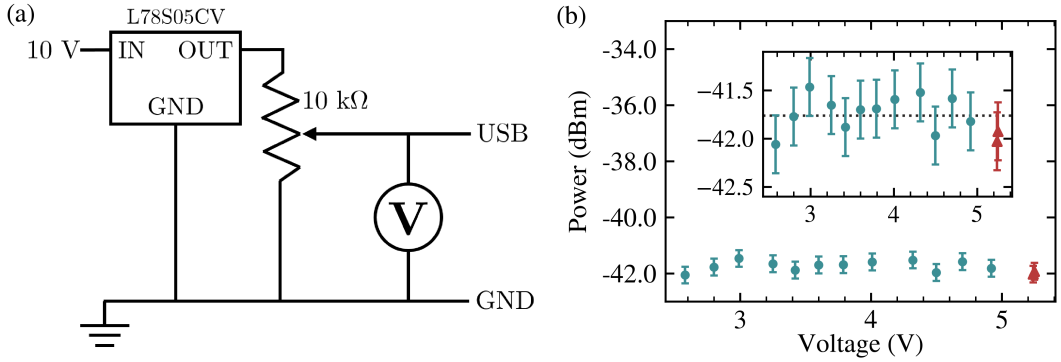

Figure 2.5: **(a)** The electrical circuit constructed to vary the input voltage across an acting transmitter. The 5.0 V voltage regulator (L78S05CV) maintained a constant potential difference across the 10 kΩ potentiometer. The potentiometer's output voltage, measured by the voltmeter, was applied across the USB and GND pins of the transmitter. **(b)** Received signal power as a function of input voltage across the acting transmitter. Each data point has been evaluated from 2500 readings made by the acting receiver. The blue circles and red triangles denote whether the transmitter was powered by the potentiometer or a micro-USB cable, respectively. The inset shows the same data over a smaller power range. Here, mean signal power has been marked by a dotted black line.

and 5.0 V in increments of approximately 0.2 V. The lower bound of this range was the minimum voltage at which the transceiver remained operational. The upper bound was the maximum voltage that could be safely applied across the Feather board's USB and ground (GND) pins without damaging any circuitry [33]. To test V at values above 5.0 V, we powered the board directly via its micro-USB port.

Figure 2.5b displays signal power measured by the acting receiver as a function of input voltage across the acting transmitter. We report a mean received signal power of -41.76 ± 0.05 dBm. Note that this is within each measurement's error, indicating a low level of dispersion. The standard deviation of the data is 0.2 dBm, which is less than the error of 0.3 dBm on each data point (this uncertainty is calculated in Section 3.3). These results suggest that a Feather board's transmission power is indeed independent of its input voltage. Therefore, when our prototype is collecting data, we do not consider inherent fluctuations [38] in the output voltage of each USB port that powers a device.

3. Signal Characterisation

3.1 Mapping Spatial Propagation

Now that we have established how transceivers operate, we can turn to measuring the spatial variation of BLE signal strength. Recall that nodes have a minimum separation of 1 U (44.45 mm). To effectively compare the strengths of received signals, there has to be an observable change in power on this spatial scale. Specifically, the fall in power across a distance of 1 U must be greater than the magnitude of noise present in the signal.

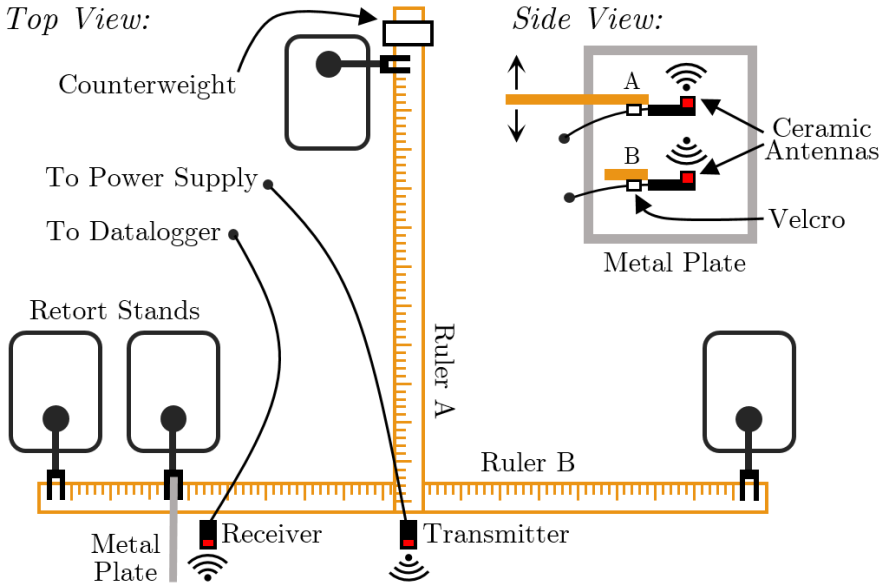

Figure 3.1: A schematic of the apparatus used to vary the relative positions of two transceivers. Elevation was changed by vertically moving the acting transmitter (attached to ruler A). Horizontal separation was changed by moving the acting receiver (attached to ruler B). Viewed from the transmitter's frame of reference, the receiver was able to freely travel across a vertical plane. Both rulers were made of wood to reduce signal reflections. The removable metal plate was used to test multipath propagation.

When an electromagnetic radio transmission travels through an indoor environment, noise manifests due to diffraction, reflection, and scattering [21,39,40]. These phenomena give rise to adverse shadowing and multipath propagation effects. Shadowing refers to the increase in noise and signal attenuation caused by obstacles blocking the primary transmission path between transceivers. Multipath propagation refers to when secondary propagation paths are created by a signal's reflections off its surrounding surfaces. As energy is lost during reflection, these secondary signals are typically attenuated, and may interfere with the primary signal upon reaching a receiver [11,21,39].

The fraction of energy that reflects off a surface is dependent on (amongst other factors) material properties, like relative permittivity and conductivity [21,39–41]. Metals, which have far higher conductivities than most building materials [41], are particularly good reflectors of radio waves. This is why metal surfaces often create unpredictable radio propagation paths, which in turn increase noise levels. These issues are prevalent in racks, whose dense metal environments cause noisy fluctuations in signal strength.

When first developing our prototype, we had to determine whether the variations in the strength of a Feather board's transmissions were sufficient to differentiate points separated by 1 U. We thus set out to map spatial propagation. We wanted to minimise noise levels, both to demonstrate proof of concept, and to create a 'baseline' dataset which could then be compared with noisier measurements. This required us to measure signal strength in controlled laboratory conditions, where shadowing and multipath effects could be minimised. To this end, we assembled the apparatus displayed in Figure 3.1.

Two transceivers were attached to wooden rulers; one acted as a transmitter, and the other as a receiver. The apparatus was set up such that relative to the acting transmitter, the acting receiver was able to freely move across a vertical plane. To reduce potential signal reflections, we utilised non-metal equipment wherever possible. Long wooden rulers ensured that load-bearing metal retort stands were at least 50 cm away from both transceivers at all times. Note that this apparatus was also used when the input voltage across an acting transmitter was being varied (see Section 2.3).

To map the spatial variation of signal strength in free space (defined here as a region with no obstacles or reflecting surfaces), the acting receiver was moved across a $20\,\text{cm} \times 60\,\text{cm}$ vertical grid in 5 cm increments. This spatial resolution is of a comparable magnitude to the minimum 4.445 cm separation between nodes. 2500 readings of received signal strength were made at each point. We were careful to maintain a stationary external environment throughout the duration of the experiment.

It is important to recognise that due to the nature of our apparatus (the presence of cables and supporting equipment), we were unable to recreate theoretical perfect-vacuum propagation conditions. Though efforts were made to minimise the level of noise in our experiment, shadowing and multipath effects were by no means completely eliminated. The next section looks at how measurements of received signal strength can be filtered and processed to account for multipath propagation.

3.2 Signal Processing

Figure 3.2 presents sample data collected when the spatial variation of signal strength was measured as discussed in the previous section. The first panel (Figure 3.2a) displays 2500 readings of signal power made when the two transceivers were separated by 20 cm. Constructing a histogram of recorded values reveals two isolated groups of data.

This division can be explained by considering the effects of multipath propagation. It is likely that the acting receiver detected two distinct signals; a primary signal which travelled directly along the main transmission path, and a weaker secondary signal which reached the receiver via an indirect path, having been produced by reflections off a nearby surface. Naïvely calculating mean signal power using all recorded data yields a value of $-26.92 \pm 0.04\,\text{dBm}$. A visual inspection of the histogram shows that this lies in between the two groups of data, where no actual readings were made. This is clearly illogical.

Evidently, to calculate an average value of signal power, readings corresponding to multipath signals must first be filtered out. In a continuous dataset, groups can be identified by evaluating turning points and isolating peaks. While this is not possible in a histogram, there exist data smoothing techniques which, given discrete measurements, can make continuous approximations of random variables. One such technique is kernel density estimation (KDE). This non-parametric method of estimating an observable's probability density function assumes nothing about the underlying distribution of data.

KDE's smooth discrete measurements by centring a kernel function on each data point; these kernels are then summed to produce a density estimate [42, 43]. The smoothness

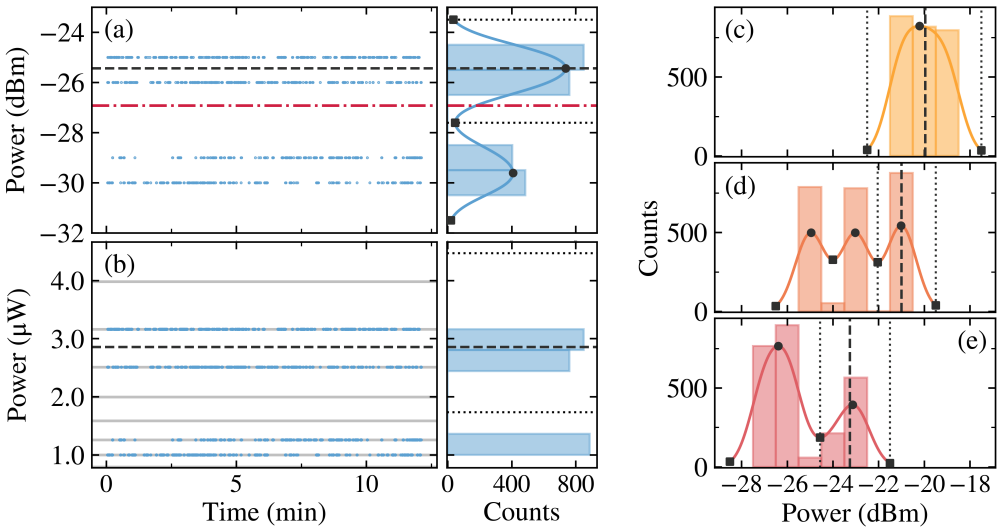

Figure 3.2: Our Gaussian kernel density estimation (KDE) program acting on four datasets. These were recorded when two transceivers were held at varying positions in free space using our laboratory apparatus. **(a)** Received signal power in dBm, together with a histogram of the data. 2500 readings were made over a 12 minute period. The transceivers were separated by 20 cm. The program begins by constructing a kernel density estimate, drawn here as a blue curve. The curve's minima and maxima have been marked with black squares and circles, respectively, and its first peak has been bounded by dotted lines. The red dotted-and-dashed line denotes a mean naïvely calculated using all recorded data. **(b)** The data above, plotted in μW instead of dBm. Since readings have a resolution of 1 dBm, horizontal grey lines have been added to indicate possible measurement values. The dashed black line marks the mean of points bounded within the first peak. This mean is converted back to dBm, before being adopted as our final reduced value of signal power. **(c,d,e)** Our KDE program successfully acting on three varied datasets, recorded when the transceivers were separated by 5 cm, 15 cm, and 25 cm, respectively. Note how none of the four histograms share a clear parametric distribution.

of the estimate is dependent on the width of each kernel, which is determined by a free parameter known as bandwidth. When constructing a kernel density estimate, bandwidth should be carefully selected such that the data are neither over- nor under-smoothed.

Figure 3.2a displays a density estimate constructed from the aforementioned recorded data. Here, we have utilised a Gaussian kernel function (to produce a smooth estimate) and a bandwidth of 0.65 dBm (established through trial and error). Once the turning points of the density estimate had been evaluated, data within the first peak could be isolated. These data were then used to determine the average value of received signal power P(dBm). However, recall that the decibel-milliwatt is a logarithmic unit. It was therefore necessary to convert readings into SI units before calculating the arithmetic mean of the filtered data. This process has been illustrated in Figure 3.2b. The final reduced value of P(dBm) is -25.44 ± 0.01 dBm, which is 41.3% greater than the naïve mean calculated using all recorded data.

The reported uncertainty on this value is the standard error α_P of data points within the first peak. It was calculated when readings were in SI units, since standard errors cannot be directly evaluated with logarithmic units. α_P was converted from microwatts to decibel-milliwatts via the functional approach [44]. Specifically, by treating (2.1) as a function f dependent on received signal power in microwatts P, the standard error α_f on P(dBm) was obtained using

$$\alpha_f^{\pm} = \left| f(\bar{P} \pm \alpha_P) - f(\bar{P}) \right|. \tag{3.1}$$

Note that though (3.1) produced asymmetric error bars, the average value of α_f has been reported instead for simplicity. In Figure 3.2a, P(dBm) has a percentage uncertainty of 0.3%. Considering the effects of shadowing and random noise, this is a relatively low figure which warrants attention. In the next section, we will see that measurements made over a longer time period suggest the current error on P(dBm) has been underestimated.

The data reduction routine described above allowed us to average repeated measurements of signal power by first filtering out secondary multipath signals. A Gaussian KDE program (available in Appendix C) was written to automate the process. This program was applied to each dataset collected when mapping free-space propagation. Recall that the acting receiver was moved across a vertical grid in 5 cm increments, and that 2500 readings were made at each point.

Figure 3.3 presents the results of the experiment. Received signal power P(dBm) was strongest at the origin, when the transceivers were directly in contact with each other. Across the grid, P(dBm) varied between -37.18 ± 0.02 dBm and -17.00 ± 0.01 dBm. This upper bound was limited by the acting receiver's saturation point, i.e. the Feather board was unable to register readings above -17 dBm. Along the central vertical axis, there is a difference of at least 2 dBm between the values of adjacent data points.

To approximate the continuous change in P(dBm), bicubic spline interpolation [45] was carried out at 0.5 cm intervals over the grid. The resultant contour plot displays how, on average, the value of P(dBm) decreased with increasing distance. Limited by grid size, we lack sufficient data to assess whether the variation in P(dBm) was isotropic.

The data above suggest that it is in fact possible to measure a difference in BLE signal power between points separated by 1 U. While this is a promising result, free-space conditions differ significantly from rack environments. In the next section, we examine the spatial variation of signal power inside a rack.

3.3 Signal Propagation in Racks

Before proceeding, the reader is reminded of how our prototype operated in an active HPC facility. Ten Feather boards were arranged vertically alongside nodes in a rack at intervals of 1 U (or an integer multiple thereof). Each device measured the power of incoming transmissions from all other devices in the network; these data were then passed to a central storage unit. Here, measurements were sorted according to their corresponding transmitting and receiving devices, creating subsets of data which were

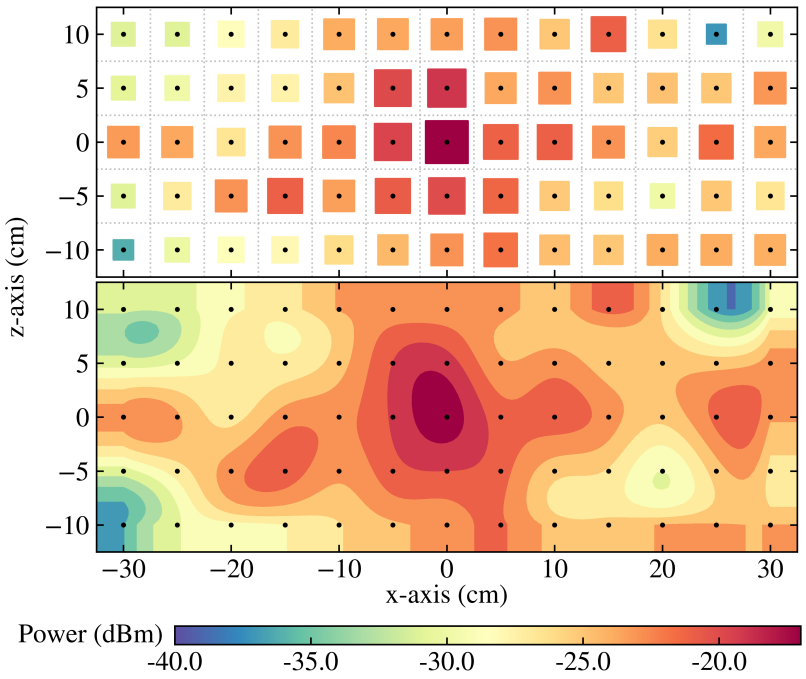

Figure 3.3: Received signal power over a 20 cm × 60 cm vertical grid in free space. The acting transmitter was located at the centre of the grid, while the acting receiver was moved in 5 cm increments. Each black dot marks the coordinates of a data point, i.e. the filtered value of signal power evaluated from 2500 readings. The area of each square in the upper panel is proportional to signal power; less negative values correspond to larger squares. A bicubic spline was used to interpolate the data over 0.5 cm intervals, allowing for the construction of the 12-level contour plot displayed in the lower panel. Notice how signal power is strongest at the origin, and visibly falls with increasing distance.

then filtered using our Gaussian KDE program. This program evaluated the power of primary signals which travelled directly between pairwise transceivers.

Each device reported a total of 2×10^4 signal power measurements per dataset. This is an order of magnitude greater than the 2.5×10^3 readings per dataset recorded when mapping free-space propagation. The increase in sample size was made to account for the higher number of acting transmitters broadcasting to each device. Sample sizes were limited by our allocated time in the HPC facility and the laboratory.

During a preliminary experiment to study the effects of antenna orientation, the ten transceivers were at intervals of 1 U. Two datasets were recorded, one where devices were angled such that their antennas faced the back of the rack, and another where devices were held horizontally such that their antennas faced straight outwards. The latter orientation has been displayed in Figure 2.1b. The results of the experiment are displayed in Figure 3.4. In both the angled and straight orientations, at least one transceiver saturated at -17 dBm. The minimum values of received signal power P(dBm) in the angled and straight datasets are -32.3 ± 0.3 dBm and -43.6 ± 0.3 dBm, respectively. The variation of P(dBm) with respect to distance is greater in the straight dataset.

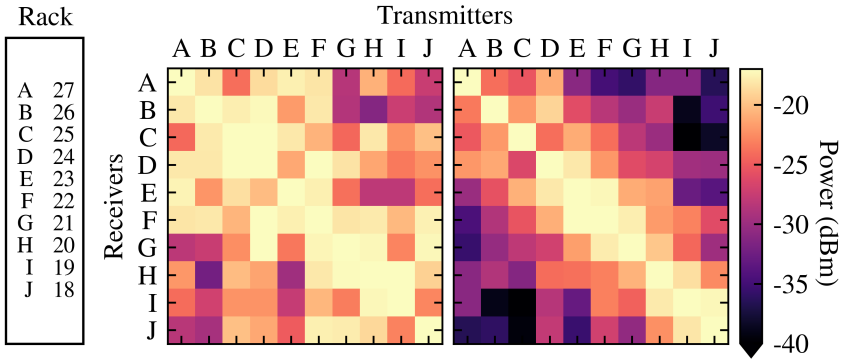

Figure 3.4: Matrix plots of the received signal power recorded by ten transceivers in a rack. Acting receivers and acting transmitters have been listed in order along the vertical and horizontal axes, respectively. Since a transceiver cannot receive signals from itself, values along the main diagonal have been set to zero. **(Left)** The 'angled' dataset, recorded when device antennas were angled towards the back of the rack. **(Right)** The 'straight' dataset, recorded when device antennas faced straight outwards. Note how the highest values of signal power correspond to transmissions sent between adjacent devices; these lie on both sides of the main diagonal. **(Far Left)** The vertical positions of transceivers within the rack, which were unchanged between datasets. Standard racks have 42 fixed mounting slots. Here, a transceiver's local ID (A, B, C, etc.) has been listed with the number of the node (27, 26, 25, etc.) it was positioned next to.

In the straight dataset, the highest values of $P(\text{dBm})$ clearly correspond to signals sent between adjacent transceivers. Meanwhile, in the angled dataset, the values of $P(\text{dBm})$ span a smaller range, making it harder to differentiate transceivers. These results strongly suggest that the orientation of a Feather board's antenna affects signal propagation within a rack. The larger magnitude of signal attenuation observed in the straight dataset is better suited to distinguishing transceivers separated by 1 U. Throughout the remaining experiments presented in this work, antennas were positioned facing straight outwards.

As discussed in the previous section, the uncertainty on $P(\text{dBm})$ can be evaluated by calculating the standard error of points within the first peak of a kernel density estimate. However, due to the limited sample size of datasets, this approach fails to account for slow-changing sources of random noise. For instance, the precision of a Feather board's ADC can be adversely affected by thermal noise and bias current instability [46, 47]. These phenomena produce random fluctuations in measured signal power which may only become apparent over prolonged periods of time.

To investigate the effects of slow-changing random noise, five broadcasting transceivers (spaced at intervals of 2 U in a rack) were set to record data over a 30 hour period. The resultant dataset was split into 60 subsets of 2×10^4 readings each. This was done to replicate the regular sample size of measurements made in racks. Our Gaussian KDE program was then run on individual subsets to evaluate respective values of $P(\text{dBm})$ and standard error. To quantify the fluctuations in measurements made by any one acting receiver, we calculated the standard deviation of its corresponding $P(\text{dBm})$ values.

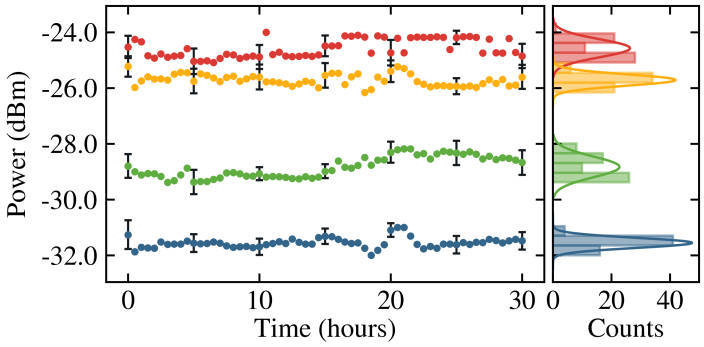

Figure 3.5: The fluctuations in received signal power over a 30 hour period. The acting receiver recorded data from four acting transmitters, hence the four groups of data. Each data point has been evaluated from a set of 2×10^4 readings using our Gaussian KDE program. The red, yellow, green, and blue points denote whether signals were received from a transmitter at a distance of 2 U, 4 U, 6 U, or 8 U, respectively. Mean signal power decreases with increasing distance. Error bars (included on every 10th point) have been scaled to 25σ to be visible. A histogram of each data group has been plotted; the overlaid curves are Gaussian functions which share the mean and standard deviation of the group.

Figure 3.5 displays the data from one of the five transceivers. As an acting receiver, it recorded data from four acting transmitters. The average standard error on individual measurements of P(dBm) is 0.01 dBm. Contrast this with the average standard deviation of P(dBm) across the four groups of data, 0.3 dBm. Evidently, neglecting slow-changing random noise can result in the underestimation of uncertainty. We have thus cautiously adopted 0.3 dBm as the error on all measurements of P(dBm) made in this work. This assumes that the random noise is distributed log-normally, and is constant in decibel-milliwatts instead of in SI units. We justify these claims with reference to the histograms in Figure 3.5, which have approximately equal widths. If noise levels were constant in SI units, we would expect histogram width to decrease with increasing signal power, since the data have been expressed in logarithmic units.

4. Node Localisation

4.1 The Reordering Algorithm

With measurements of received signal strength at hand, we are finally ready to discuss the localisation of nodes in a rack. The matrix plots presented in Figure 3.4 have been ordered, i.e. axes have been sequenced based on the physical positions of transceivers. In practice this sequence would be unknown — the determination of relative node positions is the entire purpose of our prototype. Matrix plots of undetermined systems have randomly ordered axes, and thus often appear ‘scrambled’. Node localisation can be achieved by reordering a scrambled matrix to recover its ‘diagonal’ structure, in which the highest values of received signal power lie alongside the main diagonal. Our prototype attempts to do this using a reordering algorithm, which we developed after testing several data

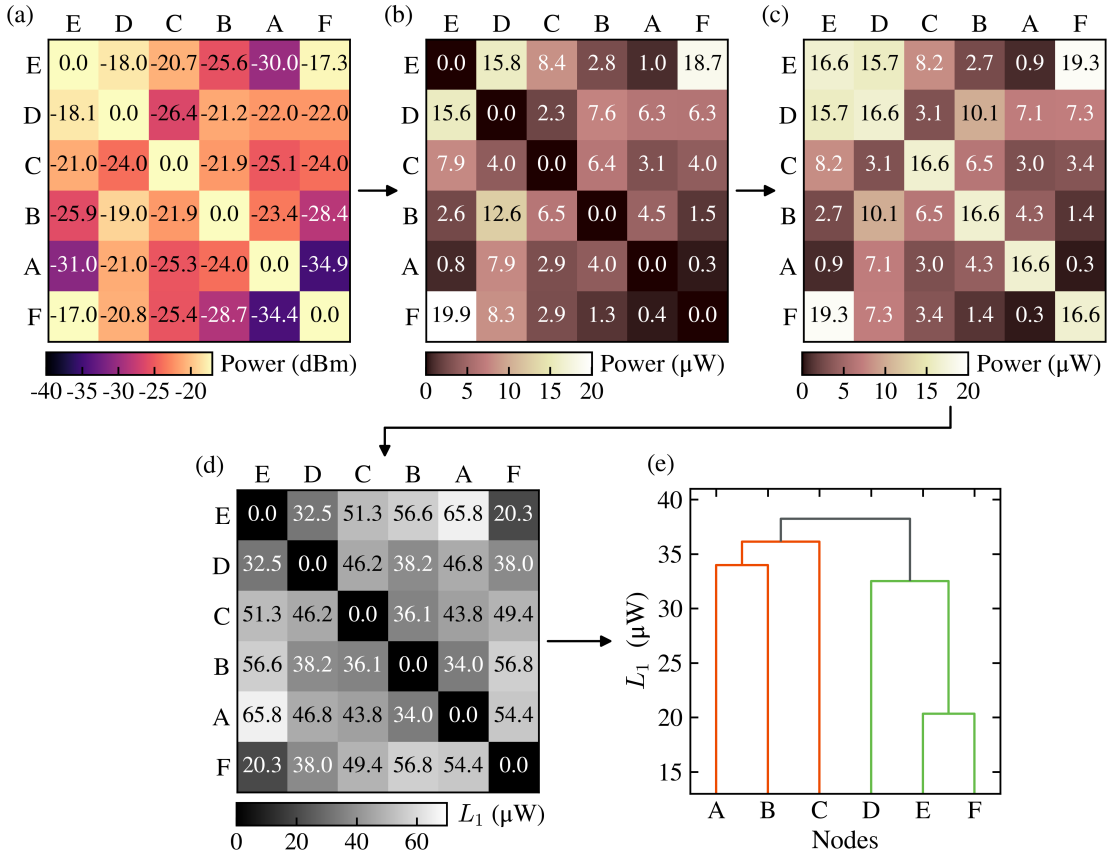

Figure 4.1: Our reordering algorithm acting on an unordered 6×6 dataset of signal power. In each matrix plot, acting receivers and acting transmitters have been listed along the vertical and horizontal axes, respectively. **(a)** Received signal power in dBm. Since a transceiver can't receive signals from itself, values along the diagonal have been set to zero. **(b)** The same dataset, converted to SI units. **(c)** The preprocessed matrix, generated by averaging dataset elements across the main diagonal. Values along the diagonal have been changed to the 95th percentile of data. **(d)** The symmetric distance matrix, containing the Manhattan distance L_1 of each possible pair of row vectors from the preprocessed matrix. **(e)** A dendrogram displaying the single-linkage clustering sequence used to recover relative transceiver positions. Horizontal bars denote the value of L_1 at which clusters merged. Colours have been added to aid technical explanations.

reduction methods and matrix processing techniques. Its routine will now be illustrated using an unordered 6×6 subset of the 10×10 straight dataset displayed in Figure 3.4.

The algorithm begins by converting data from decibel-milliwatts to SI units (see Figure 4.1a). This helps to simplify arithmetic calculations involving multiple values of signal power. Note how, due to the effects of random noise, the dataset is not symmetric. For instance, as displayed in Figure 4.1b, the power of the signal that transceiver D received from transceiver E was 0.2 μW lower than that E received from D. To compensate for these fluctuations, matrix elements are averaged across the main diagonal. At this stage (Figure 4.1c), for reasons that will become clear shortly, the diagonal's assigned value is changed to the 95th percentile of the data.

We now have a preprocessed matrix which can be reordered. Effective reordering requires a method of gauging the pairwise similarity (or dissimilarity) of rows, so that elements which are alike can be placed next to each other. By treating each row of an $N \times N$ dataset as an N-dimensional vector, we can evaluate dissimilarity by simply calculating the distance between vectors. This is why elements on the main diagonal are changed; row dissimilarity in an ordered matrix is minimised when diagonal elements are of a similar magnitude to the highest values of received signal power. Here, the use of the 95th percentile (established through trial and error) helps to avoid outliers in data.

A common metric used to quantify the distance between vectors is the Minkowski distance L_p, given by [48–50]

$$L_p(u, v) = \left(\sum_{i=1}^{N} |u_i - v_i|^p \right)^{1/p}, \tag{4.1}$$

where u and v are two N-dimensional vectors, and $p \geq 1$. Setting $p = 1$ yields the Manhattan distance L_1, which is the sum of the absolute differences of vector elements. Setting $p = 2$ yields the Euclidean distance L_2, which is a generalisation of Pythagoras' theorem in N dimensions. Higher orders of p give greater weight to outliers [49], hence our choice of the L_1 metric to evaluate row vector dissimilarity.

Formally, the number of unique L_1 values in a network of N transceivers is given by the binomial coefficient $\binom{N}{2}$. Returning to the 6×6 preprocessed matrix in Figure 4.1c, we find that calculating the L_1 value of each possible pair of row vectors produces 15 unique distances. Figure 4.1d presents these values as a symmetric distance matrix.

The sequencing of distance matrices to explore patterns in data is an active field of research [50–52]. Often termed seriation, this form of combinatorial data analysis has practical applications in many disciplines, including archaeology, bioinformatics, and operations research. Seriation methods are diverse. Different techniques are typically optimised towards recovering specific structural patterns in data. To determine relative node positions, we need a seriation method well suited to recovering the 'diagonal' structure of a distance matrix, despite potentially high levels of noise in data.

During algorithm development, 18 different seriation methods were tested on varied datasets. The majority of these were implemented using the software package described in [51]. Methods were judged based on their time complexity and the quality of resultant orderings, i.e. the number of transceivers correctly sequenced. Two initially promising candidates were the Bipolarisation algorithm (BA) and Bond Energy algorithm (BEA).

The former is a deterministic routine which iteratively moves the largest elements in a distance matrix away from the main diagonal [50, 53]. However, while the BA reportedly excels at revealing 'diagonal' structure [50], we found that in tests with noisy data, its output quality suffered. The BEA is a non-deterministic clustering routine which maximises the 'bond energy' of adjacent binary matrix elements by permuting rows and columns [51, 54]. Though the BEA performed well when given binary forms of our datasets, we were unable to find a way of automating the thresholding of matrices.

Table 4.1: The Manhattan distance L_1 of pairs of observations (transceivers) in our 6×6 distance matrix. The 15 values have been listed in increasing order of magnitude. Our seriation algorithm works through this list to recover the relative order of transceivers.

Pair	L_1(µW)	Pair	L_1(µW)	Pair	L_1(µW)
E F	20.3	B D	38.2	C E	51.3
D E	32.5	A C	43.8	A F	54.4
A B	34.0	C D	46.2	B E	56.6
B C	36.1	A D	46.8	B F	56.8
D F	38.0	C F	49.4	A E	65.8

In our empirical tests, the seriation method that most consistently recovered node positions was the single-linkage hierarchical clustering algorithm first formalised by Gruvaeus and Wainer (the GW algorithm hereafter) [50, 51, 55]. This deterministic routine reorders a distance matrix by clustering observations (transceivers) based on their L_1 values. The term single-linkage refers to how, during sequencing, the distance between two clusters is taken to be the lowest L_1 value of a pair of unclustered transceivers [49]. The sequencing of our 6×6 distance matrix can be explained with reference to a dendrogram and a list of L_1 values, available in Figure 4.1e and Table 4.1, respectively.

The GW algorithm begins sequencing by clustering the pair of observations with the lowest L_1 value, i.e. transceivers E and F at L_1(E,F) = 20.3 µW. The next lowest value is L_1(D,E) = 32.5 µW. D therefore joins the first pair of transceivers, which yields the dendrogram's green cluster. When a cluster is formed, the resultant sequence is determined by comparing the L_1 values between possible pairs of observations at the cluster's edges. Here, D has been placed beside E, since L_1(D,E) < L_1(D,F). Next, transceivers A and B cluster at 34.0 µW. Neither of these observations are part of the existing green cluster; they thus form an independent red cluster in the dendrogram.

Continuing, C is placed in the red cluster, next to B. The algorithm skips L_1(D,F), since these transceivers are already part of the same cluster. The final clustering at L_1(B,D) = 38.2 µW is non-trivial. While the L_1 value shows that the red and green clusters must link together, the sequence of this link depends on observations at the edges of both clusters. The algorithm compares the four values L_1(A,D), L_1(A,F), L_1(C,D), and L_1(C,F). The lowest of these, L_1(C,D), indicates that C should be placed next to D. As displayed in the dendrogram, this yields the final order of observations. Reassuringly, this alphabetical sequence matches the physical order of the transceivers in a rack.

The GW seriation algorithm is the final stage of our reordering algorithm (available in Appendix D). Due to a rack's discrete structure, recovering the relative positions of transceivers allows for node-level localisation, provided the number of nodes in a rack is known. For instance, in a full standard rack of 42 nodes, a sequence of 42 transceiver IDs translates directly into a set of physical locations. In the next section, we examine our algorithm's performance with various physical configurations of transceivers.

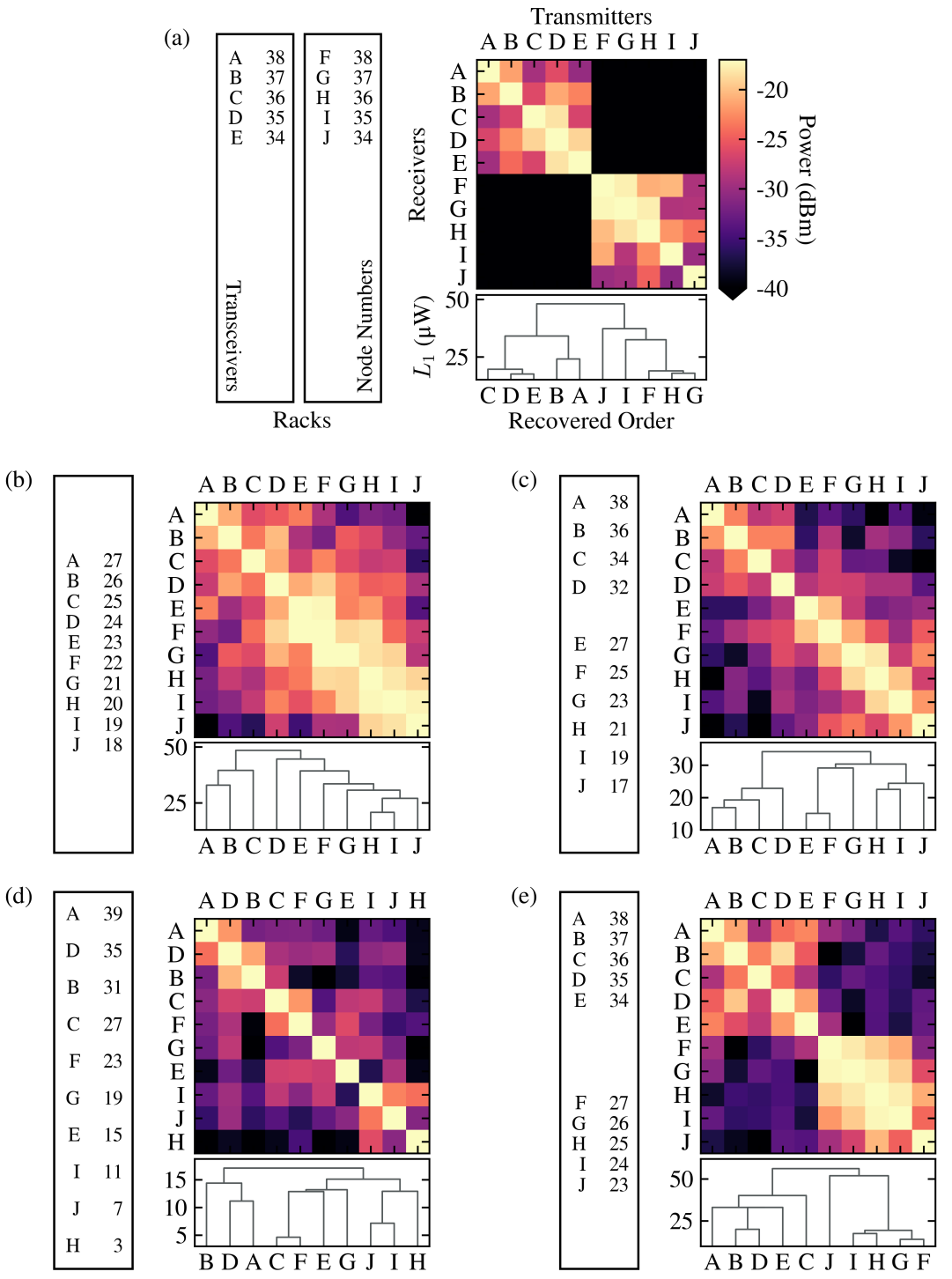

Figure 4.2: The datasets produced by testing ten transceivers in five different physical configurations. In each matrix plot of signal power, acting receivers and acting transmitters have been listed in order along the vertical and horizontal axes, respectively. Since a transceiver cannot receive signals from itself, values along the main diagonal have been set to zero. Schematics (black rectangles) of each physical configuration have been included. These denote vertical positions within a rack. The local IDs of transceivers have been listed with the numbers of the nodes they were positioned next to. The dendrograms below each matrix display the output of our reordering algorithm, i.e. the recovered relative positions of transceivers, and their corresponding hierarchical clustering sequence.

4.2 Testing Transceiver Configurations

Once a functional reordering algorithm had been developed, we were able to look into the reliability of our prototype. To study its performance in potential operational scenarios, we tested our transceivers in five different physical configurations. In the first three configurations, transceivers were spaced at intervals of 1 U, 2 U, and 4 U, respectively. This was done to emulate common node heights. Next, to emulate vertical gaps in a rack (often created when nodes are removed for maintenance), transceivers were separated into two groups that were kept 7 U apart. Finally, to examine the variation in received signal power P(dBm) through rack walls, nodes were split between two adjacent racks.

Figure 4.2 presents the resultant datasets from these physical configuration tests. Our reordering algorithm was run on each matrix plot. The dendrograms indicate the order in which transceivers clustered; they are useful tools for tracing where, if at all, the seriation of a dataset went wrong.

In Figure 4.2a (the two-rack configuration), the division between transceivers is reflected by the matrix plot's black regions. Though transmissions did travel through the racks' common wall, signals were consistently received with a power of less than -40 dBm. Our reordering algorithm differentiated the two groups of transceivers, but failed to recover order within each group.

Note the characteristic 'diagonal' bands of high P(dBm) values in the matrix plots of Figure 4.2b and Figure 4.2c (the 1 U and 2 U interval configurations, respectively). In both cases, our algorithm correctly recovered the relative positions of transceivers. By contrast, the matrix in Figure 4.2d (the 4 U interval configuration) lacks any visible 'diagonal' structure. Here, the ordering produced by our algorithm is wrong. The dendrogram shows how three pairs of transceivers were incorrectly swapped during clustering. In two of these pairs, E-G and J-I, the swapped transceivers were physically adjacent.

In Figure 4.2e (the two-group configuration), the highest values of P(dBm) correspond to signals sent between transceivers in the same group. While our reordering algorithm successfully identified the two physical groups, the overall recovered order of transceivers is wrong. In particular, the sequence FGHIJ has been incorrectly flipped.

These results suggest that our prototype is capable of correctly recovering relative positions when all ten transceivers are spaced at intervals of 2 U or less. Higher values of P(dBm) are likely to have better signal-to-noise ratios, and therefore lead to better-defined 'diagonal' structure in ordered matrix plots. The presence of this structure appears to have a positive impact on resultant ordering quality. Our algorithm was able to differentiate physical groups of transceivers, but evidently struggled to recover relative positions within these groups. This may indicate that it has difficulty sequencing small groups of observations (five transceivers as opposed to ten).

The clear divisions in the matrix plot of the two-rack configuration imply that in a multi-rack network, it should be possible to distinguish which rack a transceiver belongs to. To further investigate the fall in P(dBm) through rack walls, the ten transceivers were split between three consecutive racks. Figure 4.3 displays the results of this experiment.

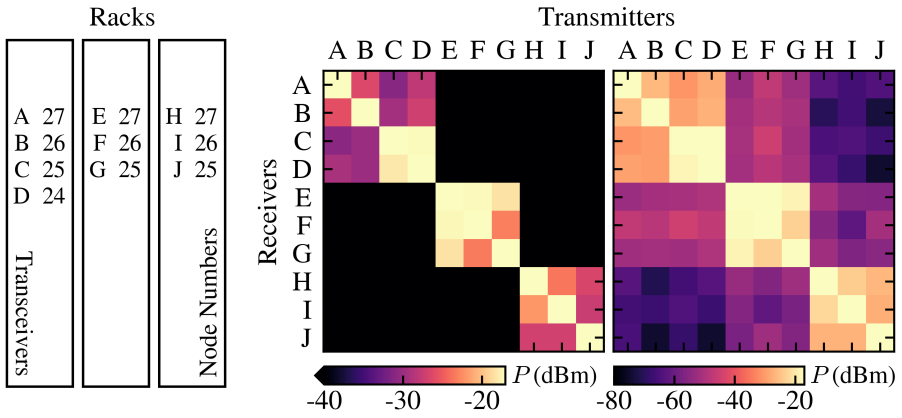

Figure 4.3: Matrix plots of received signal power. Ten transceivers were split between three consecutive racks. Both plots contain identical datasets, but have been displayed using different colour-maps. Acting receivers and acting transmitters have been listed in order along the vertical and horizontal axes, respectively. Values along the main diagonal have been set to zero. The black rectangles denote vertical transceiver positions. Each transceiver's ID has been listed with the number of the node it was positioned next to.

Transmissions that travelled through rack walls were consistently received with a power of less than $-40\,\text{dBm}$. The matrix plot shows that there is a difference in $P(\text{dBm})$ values corresponding to acting transmitters that were located one or two racks away. For instance, compare the power of the signal that A received from E ($-53.0 \pm 0.3\,\text{dBm}$) with that it received from H ($-62.8 \pm 0.3\,\text{dBm}$). These data suggest that by studying the variation of $P(\text{dBm})$ across a multi-rack network, it may be possible to recover the relative layout of racks in an HPC facility.

4.3 An Empirical Model of Propagation

When testing different physical configurations, we were limited by available hardware. Our prototype was developed using a network of only ten transceivers. It would be useful, if not economical, to be able to simulate datasets of received signal strength for arbitrary configurations of N transceivers within a rack. These datasets could then be scrambled to study the performance of our reordering algorithm across diverse node configurations. Such simulations first require modelling the spatial variation of $P(\text{dBm})$. However, shadowing effects and slow-changing random noise mean that most reliable propagation models are not deterministic functions of distance. Instead, to account for random noise, the best models consider the statistical variance of received signal strength [21, 22, 56].

We now present an empirical propagation model that attempts to predict the value of $P(\text{dBm})$ at a distance d_R from an acting transmitter. It considers random fluctuations in signal power and accounts for the path loss seen in rack environments. Its derivation has been based on that of the log-normal shadowing model (LNSM) [21, 22], which describes the magnitude of path loss in a given environment. Our model has fewer free parameters, making it easier to fit to recorded data.

We begin by assuming that received signal power $P_R\,(\mathrm{mW})$ is inversely proportional to the distance d_R between an acting receiver and acting transmitter, raised to a path loss exponent n. Then

$$P_R \propto \left(\frac{1}{d_R}\right)^n \quad \Rightarrow \quad \frac{P_0}{P_R} = \left(\frac{d_R}{d_0}\right)^n, \tag{4.2}$$

where P_0 is the signal power received at a reference distance d_0. For simplicity, we set $d_0 = 1\,\mathrm{U}$. Higher values of n indicate higher rates of signal attenuation in a given propagation environment. When $n = 2$, we recover the free-space inverse-square law. In practice, due to shadowing and waveguide effects, n varies between 1.5 and 6 [21, 22].

Throughout this work, measurements were recorded in decibel-milliwatts. The process of fitting our model to data can thus be simplified by converting (4.2) into logarithmic form. Taking the logarithm of both sides and multiplying by 10 gives

$$10\log_{10}\left[\frac{P_0\,(\mathrm{mW})}{1\,\mathrm{mW}}\right] - 10\log_{10}\left[\frac{P_R\,(\mathrm{mW})}{1\,\mathrm{mW}}\right] = 10n\log_{10}\left[\frac{d_R}{d_0}\right]. \tag{4.3}$$

Since $P(\mathrm{dBm}) \equiv 10\log_{10}[P(\mathrm{mW})/1\,\mathrm{mW}]$ and $d_0 = 1\,\mathrm{U}$, we obtain

$$\begin{aligned} P_0\,(\mathrm{dBm}) - P_R\,(\mathrm{dBm}) &= 10n\log_{10}[d_R] - \cancelto{0}{10n\log_{10}[d_0]}, \\ P_R\,(\mathrm{dBm}) &= -10n\log_{10}[d_R] + P_0\,(\mathrm{dBm})\,. \end{aligned} \tag{4.4}$$

Here, $-10n$ and $P_0\,(\mathrm{dBm})$ are the gradient and intercept, respectively, of a linear model which can be fit to recorded data. However, (4.4) only predicts the average distance-dependent value of $P_R\,(\mathrm{dBm})$. We have yet to account for random fluctuations in received signal strength.

Several studies have empirically demonstrated that, at a given distance from a wireless transmitter, the spread in measured $P_R\,(\mathrm{dBm})$ values can be described by a log-normal distribution centred on the average value of $P_R\,(\mathrm{dBm})$ at that distance [21, 22, 57, 58]. Quantitatively, this means random noise can be modelled using a Gaussian random variable $N(0, \sigma_p)$ with a mean of zero and standard deviation σ_p. This yields the final form of our empirical propagation model:

$$P_R\,(\mathrm{dBm}) = -10n\log_{10}[d_R] + P_0\,(\mathrm{dBm}) + N(0, \sigma_p)\,. \tag{4.5}$$

To evaluate n and $P_0\,(\mathrm{dBm})$, (4.4) was fit to four of our datasets. These datasets were chosen due to their relatively high signal-to-noise ratios. To convert data into a form that would be suitable for fitting, matrix plots were re-expressed by plotting $P_R\,(\mathrm{dBm})$ as a function of the logarithm of internode distance d_R. Figure 4.4a displays an example in practice; the straight dataset first introduced in Figure 3.4 has now been presented as a scatter plot. Note that the data are homoscedastic, i.e. the uncertainty on each data point is 0.3 dBm.

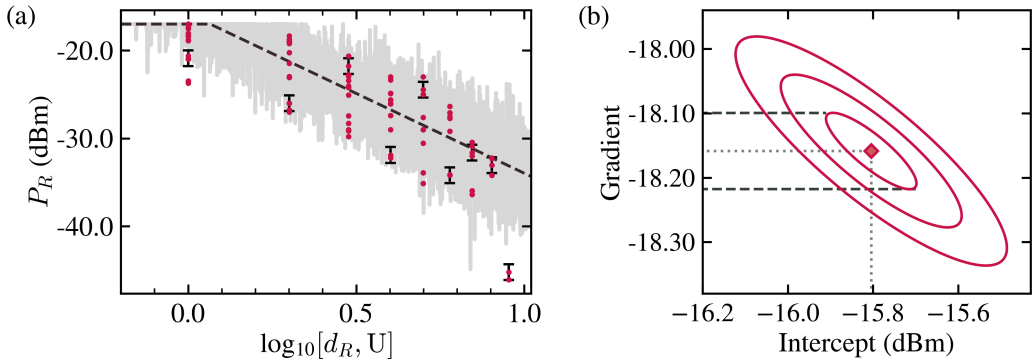

Figure 4.4: **(a)** Received signal power $P_R\,(\mathrm{dBm})$ as a function of the logarithm of internode distance d_R. The data (red points) were recorded when device antennas faced straight outwards, i.e. this is the straight dataset. Error bars (included on one point at each internode distance) have been scaled to 3σ to be visible. χ^2 minimisation was used to fit our signal propagation model (dashed black line) to the data. The grey line (shaded region) denotes the output of our propagation model when using the average values of optimised parameters across four datasets. Note the horizontal cut-off at $-17\,\mathrm{dBm}$; this reflects the saturation point of our BLE transceivers. **(b)** The error surface plot of the straight dataset. The three contours correspond to $\Delta\chi^2 = 1, 4,$ and 9, respectively. The dashed lines mark the extrema of the ellipse along the gradient parameter axis. The gradient's uncertainty is the interval between its optimised value and these tangent lines.

Table 4.2: The optimised parameters of our signal propagation model across four datasets. Datasets have been named after the figure in which they were first presented. The reported uncertainty on the average value of each parameter is its standard error.

Dataset	n	$P_0\,(\mathrm{dBm})$	$\sigma_p\,(\mathrm{dBm})$
Figure 3.4 (straight)	1.841 ± 0.006	-18.0 ± 0.1	2.557
Figure 4.2b	1.816 ± 0.006	-15.8 ± 0.1	2.861
Figure 4.2c	1.697 ± 0.009	-14.9 ± 0.1	2.907
Figure 4.2e	1.972 ± 0.006	-14.60 ± 0.07	2.481
Average	1.83 ± 0.06	-15.8 ± 0.8	2.7 ± 0.1

Fitting was carried out by minimising the goodness-of-fit parameter χ^2, defined as the sum of the squares of normalised residuals R. These were calculated using [44]

$$R_i = \frac{P_i - P(d_i)}{\alpha_i}, \tag{4.6}$$

where P_i, α_i, and $P(d_i)$ are the ith data point, its uncertainty, and the value of the fit, respectively. The uncertainties on the gradient and intercept of each fit were determined by plotting a contour map of the elliptical error surface at $\Delta\chi^2 = 1$ Specifically, the error on an optimised parameter was taken to be the interval between its value and the extremum of the ellipse. This process has been illustrated in Figure 4.4b using the error surface of the straight dataset.

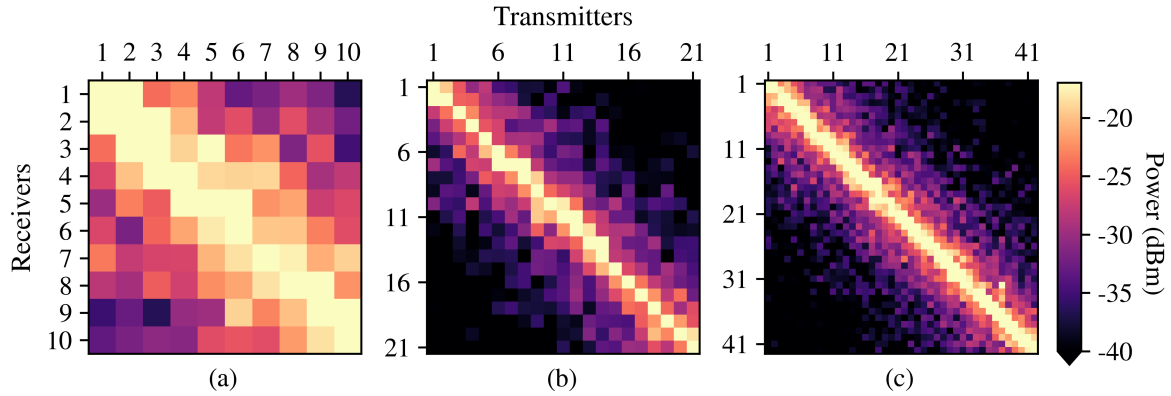

Figure 4.5: Three simulated datasets generated using our signal propagation model. Acting receivers and acting transmitters have been listed along the vertical and horizontal axes, respectively. Note the 'diagonal' structure in each matrix plot. **(a)** 10 simulated transceivers were spaced at intervals of 1 U. **(b)** 21 simulated transceivers were spaced at intervals of 2 U. **(c)** 42 simulated transceivers were spaced at intervals of 1 U. In practice, configurations (b) and (c) would span the height of a standard rack with 42 slots.

To evaluate σ_p using a given dataset, data points were first split into groups according to their internode distance d_R. We calculated the standard deviation of each group; the median of these standard deviations was then adopted as σ_p. Our results have been summarised in Table 4.2, together with the average values of n, $P_0\,(\text{dBm})$, and σ_p across the four aforementioned datasets.

Using these average parameters, we were finally able to apply (4.5) towards producing simulated datasets of received signal strength. To account for the Feather board's saturation point, we set an upper bound of $-17\,\text{dBm}$ on all generated values of $P_R\,(\text{dBm})$ (see Figure 4.4a). Figure 4.5 displays three datasets that were simulated using our propagation model. Each matrix plot was scrambled, then passed to our reordering algorithm. We gauged the quality of resultant orderings by calculating Spearman's rank correlation coefficient [59]. This non-parametric technique quantitatively assesses the association between two ranked lists, which in our case were the original and recovered order of simulated datasets. We defer the presentation of these results to a future work.

5. Conclusion

In this work, we attempted to create a node localisation system for high-performance computing (HPC) facilities by constructing a prototype network of ten Bluetooth Low Energy (BLE) transceivers. We proposed comparing the relative strengths of received signals across a set of transceivers to recover their relative positions. BLE transceivers were chosen due to their affordability and small physical size. Their wireless standard enabled unsynchronised device communication, which was well suited to our decentralised network. We discussed the operation of our transceivers, then examined how their transmission power appeared to be independent of input voltage.

A Gaussian kernel density estimation program was written to filter out secondary multipath signals so that repeated measurements of signal power could be averaged. Using a pair of transceivers, we mapped the spatial variation of received signal power P(dBm) in free space. Our results indicated that it was possible to measure a difference in BLE signal strength between points separated by 1 U (the minimum distance between two nodes). When testing ten transceivers alongside nodes in a rack, we observed that antenna orientation had a significant effect on the magnitude of signal attenuation. We empirically demonstrated how failing to account for slow-changing random noise would lead to underestimating the uncertainty on P(dBm).

A deterministic reordering algorithm was developed to recover the relative positions of transceivers when given an unordered matrix plot of P(dBm) values. It worked by producing a preprocessed matrix that could then be sequenced via single-linkage hierarchical clustering. Using this algorithm, we tested the performance of our prototype across various physical configurations. Crucially, relative positions in a rack were correctly recovered when all ten transceivers were spaced at intervals of 2 U or less. To the best of our knowledge, this was the first time BLE transceivers had been used to achieve localisation at an accuracy of less than 0.5 m. When testing multi-rack configurations, we found that it was possible to distinguish which rack a transceiver belonged to based on divisions in reordered matrix plots. To be able to simulate data, we derived a signal propagation model which accounted for random fluctuations in P(dBm). Empirical parameters were optimised by fitting our model to four recorded datasets.

A logical step towards the commercialisation of our prototype would be the streamlining of transceivers; the cost of our system could be reduced by removing our BLE microcontroller's redundant components. Having a larger number of transceivers would allow for more thorough tests of potential operational scenarios. These empirical data would help validate any conclusions drawn from simulated datasets. By improving our reordering algorithm, it may be possible to derive the physical locations of nodes in addition to their relative positions. This might be done by evaluating the number of nodes in a rack using reordered matrix plots. Many hours were spent on the development of our prototype. We hope it will be of use to someone in the future.

6. Acknowledgments

This work used the DiRAC@Durham facility managed by the Institute for Computational Cosmology on behalf of the STFC DiRAC HPC Facility (www.dirac.ac.uk). The equipment was funded by BEIS capital funding via STFC capital grants ST/K00042X/1, ST/P002293/1, ST/R002371/1 and ST/S002502/1, Durham University and STFC operations grant ST/R000832/1. DiRAC is part of the National e-Infrastructure.

We thank Dr A. Basden and Prof. R. Bower. Their guidance and support over the past year has been invaluable, and this work would not have been possible without them. We thank Prof. S. Cole, who helped ensure that this report was finished in time. We

thank Prof. I. G. Hughes, who provided useful advice during the development of our Gaussian KDE routine. We are grateful to A. Southren and J. Palmer, for keeping the author well-fed. We apologise to J. Everitt, who lives with the author and has been very patient. We thank H. Petrovic for staying posi. We thank M. Rossetter for always having a contingency plan. We thank L. Dohm-Karatjas for four years of strong fighting. We thank S. Baker for being supportive and kind. Finally, we thank our family, for reasons to numerous to list.

References

[1] E. Strohmaier *et al.*, "The marketplace of high-performance computing," *Parallel Computing*, vol. 25, no. 13, pp. 1517–1544, 1999.

[2] H. Fu *et al.*, "The Sunway TaihuLight supercomputer: system and applications," *Science China Information Sciences*, vol. 59, p. 072001, Jun 2016.

[3] B. Barney, *Introduction to Parallel Computing*. Lawrence Livermore National Laboratory, 2010. URL: `https://computing.llnl.gov/tutorials/parallel_comp/`, last accessed Apr 1, 2020.

[4] M. Turilli *et al.*, "Characterizing the Performance of Executing Many-tasks on Summit," *arXiv e-prints*, Sep 2019. arXiv:1909.03057.

[5] J. West, T. Dean, and J. Andrews, *Network+ Guide to Networks*, pp. 169–170. Cengage Learning (Boston), seventh ed., 2015.

[6] ECIA, *Standard: Cabinets, Racks, Panels, and Assoc. Equipment (EIA/ECA-310-E)*, 2005. URL: `https://global.ihs.com/doc_detail.cfm?gid=SBSSIBAAAAAAAAAA`, last accessed Apr 1, 2020.

[7] B. Schroeder and G. A. Gibson, "A large-scale study of failures in high-performance computing systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 7, pp. 337–350, Oct 2010.

[8] V. Melesse Vergara *et al.*, "Scaling the Summit: Deploying the World's Fastest Supercomputer," in *Int. Workshop on OpenPOWER for HPC*, (Frankfurt), Oak Ridge National Laboratory, Jun 2019.

[9] C. Brignone *et al.*, "Real time asset tracking in the data center," *Distributed and Parallel Databases*, vol. 21, pp. 145–165, Jun 2007.

[10] L. Zhang *et al.*, "Online Electricity Cost Saving Algorithms for Co-Location Data Centers," *IEEE Journal on Selected Areas in Communications*, vol. 33, pp. 2906–2919, Dec 2015.

[11] F. Zafari, A. Gkelias, and K. K. Leung, "A survey of indoor localization systems and technologies," *IEEE Communications Surveys Tutorials*, vol. 21, no. 3, pp. 2568–2599, 2019.

[12] G. M. Mendoza-Silva, J. Torres-Sospedra, and J. Huerta, "A Meta-Review of Indoor Positioning Systems," *Sensors*, vol. 19, no. 20, p. 4507, 2019.

[13] A. Alarifi *et al.*, "Ultra Wideband Indoor Positioning Technologies: Analysis and Recent Advances," *Sensors*, vol. 16, no. 5, p. 707, 2016.

[14] J. C. Nelson *et al.*, "Locating and Tracking Data Center Assets using Active RFID Tags and a Mobile Robot," in *10th Int. Conf. on Emerging Technologies (CEWIT)*, (Melville, NY), Oct 2013.

[15] Omni-ID, Ltd., *White Paper: The Solution to Automating IT & Data Center Asset Management*, 2019. URL: `https://www.omni-id.com/pdfs/Omni-ID_IT-Asset-Management-White-Paper_1.pdf`, last accessed Apr 1, 2020.

[16] Raritan, Inc., *Data Sheet: Intelligent Asset Tags and Sensors*, 2016. URL: `https://www.raritan.com/assets/ram/resources/data_sheets/raritan-ds-Intelligent_Asset_Tags_and_Sensors.pdf`, last accessed Apr 1, 2020.

[17] R. Natarajan, P. Zand, and M. Nabi, "Analysis of coexistence between IEEE 802.15.4, BLE and IEEE 802.11 in the 2.4 GHz ISM band," in *42nd Annual Conference of the IEEE Industrial Electronics Society*, (Florence), pp. 6025–6032, Oct 2016.

[18] P. Kriz, F. Maly, and T. Kozel, "Improving Indoor Localization Using Bluetooth Low Energy Beacons," *Mobile Information Systems*, vol. 2016, Mar 2016. Article ID: 2083094.

[19] Nordic Semiconductor ASA, *Data Sheet: nRF52840 Product Specification v1.1*, 2019. URL: `https://infocenter.nordicsemi.com/pdf/nRF52840_PS_v1.1.pdf`, last accessed Apr 1, 2020.

[20] C. Paterna *et al.*, "A Bluetooth Low Energy Indoor Positioning System with Channel Diversity, Weighted Trilateration and Kalman Filtering," *Sensors*, vol. 17, no. 12, p. 2927, 2017.

[21] T. S. Rappaport, *Wireless Communications: Principles and Practice*, pp. 69–80, 102–106, 139–141. Prentice-Hall, Inc. (Upper Saddle River, NJ), 1996.

[22] B. Dezfouli *et al.*, "Modeling low-power wireless communications," *Journal of Network and Computer Applications*, vol. 51, pp. 102–126, 2015.

[23] F. Forno, G. Malnati, and G. Portelli, "Design and implementation of a Bluetooth ad hoc network for indoor positioning," *IEE Proceedings - Software*, vol. 152, pp. 223–228, Oct 2005.

[24] L. Zwirello *et al.*, "UWB Localization System for Indoor Applications: Concept, Realization and Analysis," *Journal of Electrical and Computer Engineering*, vol. 2012, Feb 2012. Article ID: 849638.

[25] Core Specification Working Group, *Data Sheet: Bluetooth Core Specification v5.2*. Bluetooth SIG, Dec 2019. Vol. 6, pp. 1281–1286, 2827–3073, URL: `https://www.bluetooth.com/specifications/bluetooth-core-specification/`, last accessed Apr 1, 2020.

[26] J. F. Ensworth and M. S. Reynolds, "BLE-Backscatter: Ultralow-Power IoT Nodes Compatible With Bluetooth 4.0 Low Energy (BLE) Smartphones and Tablets," *IEEE Transactions on Microwave Theory and Techniques*, vol. 65, pp. 3360–3368, Sep 2017.

[27] A. Nikoukar *et al.*, "Empirical analysis and modeling of Bluetooth low-energy (BLE) advertisement channels," in *2018 17th Annual Med-Hoc-Net Workshop*, (Capri, Italy), pp. 1–6, Jun 2018.

[28] K. Townsend, C. Cufí, and R. Davidson, *Getting Started with Bluetooth Low Energy: Tools and Techniques for Low-Power Networking*, ch. 2–3. O'Reilly Media, Inc. (Sebastopol, CA), 2014.

[29] G. Shan and B. Roh, "Advertisement Interval to Minimize Discovery Time of Whole BLE Advertisers," *IEEE Access*, vol. 6, pp. 17817–17825, Mar 2018.

[30] Apple Inc., *Data Sheet: Accessory Design Guidelines for Apple Devices, Release R11*, Nov 2019. pp. 111–117 URL: `https://developer.apple.com/accessories/Accessory-Design-Guidelines.pdf`, last accessed Apr 1, 2020.

[31] E. Tsimbalo, X. Fafoutis, and R. J. Piechocki, "CRC Error Correction in IoT Applications," *IEEE Transactions on Industrial Informatics*, vol. 13, pp. 361–369, Feb 2017.

[32] D. M. Dobkin, *The RF in RFID*, pp. 103–108. Newnes-Elsevier Inc. (Oxford), second ed., 2013.

[33] L. Fried, D. Halbert, *et al.*, *Data Sheet: Introducing the Adafruit nRF52840 Feather*. Adafruit Ind., Feb 2020. pp. 1–14, URL: `https://cdn-learn.adafruit.com/downloads/pdf/introducing-the-adafruit-nrf52840-feather.pdf`, last accessed Apr 6, 2020.

[34] Raytac Corp., *Data Sheet: BT 5.0 Module (nRF52840), MDBT50Q–1M (Chip Antenna)*, Jun 2018. URL: `https://cdn-learn.adafruit.com/assets/assets/000/068/544/original/Raytac_MDBT50Q.pdf`, last accessed Apr 6, 2020.

[35] M. T. Sebastian, R. Ubic, and H. Jantunen, "Low-loss dielectric ceramic materials and their properties," *International Materials Reviews*, vol. 60, no. 7, pp. 392–412, 2015.

[36] J. Carr, *RF Components and Circuits*, p. 45. Newnes-Elsevier Science (Oxford), 2002.

[37] G. Lui *et al.*, "Differences in RSSI readings made by different Wi-Fi chipsets: A limitation of WLAN localization," in *Int. Conf. on Localization and GNSS*, (Tampere, Finland), pp. 53–57, Jun 2011.

[38] J. T. J. Penttinen, *The Telecommunications Handbook: Engineering Guidelines for Fixed, Mobile and Satellite Systems*, pp. 109–110. John Wiley & Sons, Ltd. (Chichester, West Sussex), 2015.

[39] O. Landron, M. J. Feuerstein, and T. S. Rappaport, "A Comparison of Theoretical and Empirical Reflection Coefficients for Typical Exterior Wall Surfaces in a Mobile Radio Environment," *IEEE Transactions on Antennas and Propagation*, vol. 44, pp. 341–351, Mar 1996.

[40] T. Koppel *et al.*, "Reflection and Transmission Properties of Common Construction Materials at 2.4 GHz Frequency," *Energy Procedia*, vol. 113, pp. 158–165, 2017.

[41] R. Rudd *et al.*, *Technical Report: Building Materials and Propagation.* Office of Communications (Ofcom), Sep 2014. No. 2604/BMEM/R/3/2.0, pp. 31–41.

[42] B. W. Silverman, *Density Estimation for Statistics and Data Analysis*, pp. 7–18. Chapman & Hall/CRC (Boca Raton, FL), 1986 (first ed.), 1998 (reprint).

[43] A. B. Tsybakov, *Introduction to Nonparametric Estimation*, pp. 1–4. Springer Science+Business Media, LLC (New York), 2009.

[44] I. G. Hughes and T. P. A. Hase, *Measurements and their Uncertainties*, pp. 37–51. Oxford University Press (Oxford), 2010.

[45] C. de Boor, "Bicubic Spline Interpolation," *Journal of Mathematics and Physics*, vol. 41, no. 1–4, pp. 212–218, 1962.

[46] R. H. Walden, "Analog-to-Digital Converter Survey and Analysis," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 4, pp. 539–550, 1999.

[47] K. Iizuka *et al.*, "A 14-bit Digitally Self-Calibrated Pipelined ADC With Adaptive Bias Optimization for Arbitrary Speeds Up to 40 MS/s," *IEEE J. Solid-State Circuits*, vol. 41, pp. 883–890, 2006.

[48] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, pp. 72–74. Morgan Kaufmann-Elsevier Inc. (Waltham, MA), third ed., 2012.

[49] T. Hill and P. Lewicki, *Statistics: Methods and Applications, A Comprehensive Reference for Science, Industry, and Data Mining*, pp. 118–119. StatSoft Inc. (Tulsa, OK), 2006.

[50] M. Behrisch *et al.*, "Matrix Reordering Methods for Table and Network Visualization," *Computer Graphics Forum*, vol. 35, no. 3, pp. 693–716, 2016.

[51] M. Hahsler, K. Hornik, and C. Buchta, "Getting Things in Order: An Introduction to the R Package seriation," *Journal of Statistical Software, Articles*, vol. 25, pp. 1–34, Mar 2008.

[52] I. Liiv, "Seriation and Matrix Reordering methods: An Historical Overview," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 3, pp. 70–91, Mar 2010.

[53] L. Hubert, "Some applications of graph theory and related nonmetric techniques to problems of approximate seriation: The case of symmetric proximity measures," *British Journal of Mathematical and Statistical Psychology*, vol. 27, no. 2, pp. 133–153, 1974.

[54] W. T. McCormick, P. J. Schweitzer, and T. W. White, "Problem Decomposition and Data Reorganization by a Clustering Technique," *Operations Research*, vol. 20, no. 5, pp. 993–1009, 1972.

[55] G. Gruvaeus and H. Wainer, "Two additions to hierarchical cluster analysis," *British Journal of Mathematical and Statistical Psychology*, vol. 25, no. 2, pp. 200–206, 1972.

[56] R. Desimone, B. M. Brito, and J. Baston, "Model of Indoor Signal Propagation using Log-Normal Shadowing," in *Long Island Systems, Appl. and Technology*, (Farmingdale, NY), pp. 1–4, 2015.

[57] D. C. Cox, R. R. Murray, and A. W. Norris, "800-MHz Attenuation Measured In and Around Suburban Houses," *AT&T Bell Laboratories Technical Journal*, vol. 63, no. 6, pp. 921–954, 1984.

[58] L. Fenton, "The Sum of Log-Normal Probability Distributions in Scatter Transmission Systems," *IRE Transactions on Communications Systems*, vol. 8, pp. 57–67, Mar 1960.

[59] C. Spearman, "The Proof and Measurement of Association between Two Things," *The American Journal of Psychology*, vol. 15, no. 1, pp. 72–101, 1904.

A. Transceiver Source Code

This customised program, written in the Arduino programming language, was uploaded to each transceiver in our network. It implements specific advertising and scanning intervals, assigns transmission payload content (e.g. local IDs), and determines whether received data packets should be recorded.

```
1  /*Transceiver code, heavily modified from an online example*/
2  /*********************************************************
3    This is an example for our nRF52 based Bluefruit LE modules
4    Pick one up today in the adafruit shop!
5    Adafruit invests time and resources providing this open
6    source code, please support Adafruit and open-source
       hardware
7    by purchasing products from Adafruit!
8    MIT license, check LICENSE for more information
9    All text above, and the splash screen below must be included
10   in any redistribution
11 *********************************************************/
12
13 #include <bluefruit.h>
14 #include <ble_gap.h>
15
16 /* UUID used to differentiate this device. Byte order is
     reversed.
17  * e9e43163-31c6-48cd-b424-b9025b1127f9  */
18 const uint8_t CUSTOM_UUID[] =
19 {
20   0xF9, 0x27, 0x11, 0x5B, 0x02, 0xB9, 0x24, 0xB4,
21   0xCD, 0x48, 0xC6, 0x31, 0x63, 0x31, 0xE4, 0xE9
22 };
23 BLEUuid uuid = BLEUuid(CUSTOM_UUID);
24
25 /* Unique local name of device */
26 char node_name[] = "001";
27
28 void setup()
29 {
30   Serial.begin(9600);
31   /*  Initialises maximum connections using Peripheral = 1,
       Central = 1.
32    *  Though the transceivers do not make any connections,
```

```
33      *  this is still neccessary to initialise advertising &
            scanning */
34    Bluefruit.begin(1, 1);
35    Bluefruit.setTxPower(8);    // Bluefruit.h documents values,
          8dBm is maximum
36    /* Sets the node name */
37    Bluefruit.setName(node_name);
38    /* Sets the blinking interval on BLUE LED */
39    Bluefruit.setConnLedInterval(250);
40    /* Begins advertising */
41    startAdv();
42    /*  Begins scanning for advertising packets. The scan
          interval
43     *  (time between scans) and window (scan duration) are 40
           ms
44     *  and 30 ms respectively   */
45    Bluefruit.Scanner.setRxCallback(scan_callback);
46    Bluefruit.Scanner.filterUuid(uuid); // Filters foreign BLE
          devices
47    Bluefruit.Scanner.setInterval(64, 48); // Units of 0.625 ms
48    Bluefruit.Scanner.start(0);         // Scans indefinitely
49  }
50
51  void startAdv(void)
52  {
53    /* Advertising packet (limited to 31 bytes)*/
54    Bluefruit.Advertising.addFlags(
          BLE_GAP_ADV_FLAGS_LE_ONLY_GENERAL_DISC_MODE);
55    Bluefruit.Advertising.addTxPower();
56    Bluefruit.Advertising.addUuid(uuid);
57    Bluefruit.Advertising.addName(); // Adds device name to
          advertising packet
58    Bluefruit.Advertising.setInterval(32, 244); // Units of
          0.625 ms (20 ms & 152.5 ms)
59    Bluefruit.Advertising.setFastTimeout(5); // 5s before fast
          mode ends
60    Bluefruit.Advertising.start(0); // Advertises indefinitely
61  }
62
63  void scan_callback(ble_gap_evt_adv_report_t* report)
64  {
65    /* Complete local ID of transmitter (5 characters or less)
          */
66    uint8_t buffer[8] = {0};
```

```
67    if  (Bluefruit.Scanner.parseReportByType(
68          report,
69          BLE_GAP_AD_TYPE_COMPLETE_LOCAL_NAME,
70          buffer, sizeof(buffer) ))
71    {
72    /* RSSI value */
73    signed int rssi_val = report->rssi;
74    /*Prints to serial with delay to avoid buffer overflow*/
75    printf("%s,%d,%s\n", buffer, rssi_val, node_name);
76    delay(50);
77    }
78    Bluefruit.Scanner.resume(); // Resumes scanning after report
          recieved
79  }
80
81  void loop()
82  {
83    // Does Nothing
84  }
```

B. Serial Port Communication Code

This Python program enables serial port communication between nodes and transceivers. It is used to control the number of data points recorded in a given dataset. Recorded data are written to .txt files, which then pass to a central storage unit.

```
1   import serial
2   from serial.tools.list_ports import comports
3   import time
4   import sys
5
6   BLE_pid = 32809 # Product ID
7   data_lim = 20000 # Number of data points to record
8   baud_rate = 9600
9
10  def port_names():
11      '''Returns name of connected port, ensures only 1 device
            is connected'''
12      ready = False
13      print("Connect 1 device. Port waiting...")
14      timeout = 0 # Initialises timeout counter to stop program
15      while not ready:
16          try:
17              # Filters by pid to list available BLE devices
```

```
18              BLE_ports = list(p.device for p in comports() if p
                    .pid == BLE_pid)
19              if len(BLE_ports) != 1:
20                  timeout += 1 # Adds to timeout counter
21                  if timeout == 5000:
22                      print('Serial has timed out, wait
                            terminated')
23                      quit()
24                  continue
25              else:
26                  print("Initialising port")
27                  ready = True
28          except KeyboardInterrupt:
29              print('Wait terminated, shutdown requested')
30              sys.exit()
31      return BLE_ports
32
33  def connect_port():
34      '''Connects to port, returns serial object'''
35      BLE_ports = port_names()
36      time.sleep(2) # Leaves time to update port link
37      ser = serial.Serial(BLE_ports[0], baudrate=baud_rate,
            timeout=2)
38      time.sleep(5) # Leaves time to prepare serial object
39      return ser
40
41  ser = connect_port()
42  print("Logging Data from {}".format(ser.name))
43  data_count = 0 # Log number of data points recorded
44
45  while data_count < data_lim:
46      try:
47          byte_data = ser.readline()
48          # Writes data to text file
49          write_data = byte_data[0:len(byte_data)].decode("utf-8
                ")
50          rx_name = byte_data[-4:-1].decode("utf-8")
51          if write_data: # Checks that string isn't empty
52              with open('{}.txt'.format(rx_name), 'a') as file:
53                  file.write(write_data)
54                  data_count += 1
55
56      except serial.serialutil.SerialException:
57          print("Device disconnected!")
```

```
58          ser.close() # Closes previous serial connection
59          time.sleep(2)
60          ser = connect_port()
61          print('Reconnected on {}. Logging Data'.format(ser.
               name))
62          continue
63
64      except KeyboardInterrupt:
65          print("Shutdown requested, {} points logged".format(
               data_count))
66          sys.exit()
67
68  else:
69      print('{} points recorded from {}, log complete'.format(
           data_lim, rx_name))
```

C. Signal Processing Code

Once measurements have been passed to a central storage unit, the following Python program applies our Gaussian KDE routine to the data. Resultant values of P(dBm) are fed into an SQLite database to simplify access and overwriting.

```
1   import numpy as np
2   import glob
3   import sqlite3
4   from sklearn.neighbors import KernelDensity
5   from scipy.signal import find_peaks
6
7   # Kernel Density Parameters
8   b_width = 0.65 # Kernel Bandwith
9   n_points = 1000 # Number of points to extrapolate over
10  buff = 1.5 # Buffer for RSSI extrapolation range
11  h_thresh = 20 # Height threshold for points
12
13  def reduce_data(periph, data, b_width, n_points, buff,
       h_thresh):
14      '''Returns the mean RSSI value for a given peripheral.
           Please refer to
15      plot_histo.py for a breakdown into functions'''
16      # Unpacks rssi data for given peripheral
17      idx = np.where(data[:,0] == periph)
18      rssi = data[idx,1].flatten() # In dBm
19
20      # Kernel Density Estimate of counts
```

```
21      kde_skl = KernelDensity(bandwidth=b_width)
22      kde_skl.fit(rssi[:, np.newaxis]) # Indexed because each
           row is a data point
23      # Returns a range of points to evaluate
24      grid = np.linspace(rssi.min()-buff, rssi.max()+buff,
           n_points)
25      log_pdf = kde_skl.score_samples(grid[:, np.newaxis])
26      counts = np.exp(log_pdf) * len(rssi) # Counts from PDF
27
28      # Returns indices of local maxima and minima when given
           smooth 1D curve
29      max_idx, _ = find_peaks(counts, height=(h_thresh))
30      min_idx, _ = find_peaks(-counts)
31      min_idx = np.concatenate(( [0], min_idx, [n_points-1] )) #
            Concats end-pts
32
33      # Finds indices of minima adjacent to the first peak
34      ins = np.searchsorted(min_idx, max_idx[-1]) # Returns
           insertion index
35      bds_idx = [min_idx[ins-1], min_idx[ins]] # Indices of
           bounds
36
37      # Filters for data points within the peak's bounding
           indices
38      true_vals = np.logical_and(rssi>=grid[bds_idx[0]], rssi<=
           grid[bds_idx[1]])
39      filter_idx = np.where(true_vals) # Indices within peak
40      rssi_power = rssi[filter_idx] # Filtered power values
41
42      power_vals = 10**(rssi_power/10) * 10**3 # Signal values
           in MICROwatts
43      mean_power = np.mean(power_vals) # Mean in micro-W
44      mean_rssi = 10 * np.log10(mean_power * 10**-3) # Mean in
           dBm
45      return mean_rssi
46
47  def db_connect():
48      '''Creates a connection to the database'''
49      con = sqlite3.connect('database.sqlite')
50      return con
51
52  def create_table(con):
53      '''Creates table headers'''
54      cur = con.cursor() # Creates a cursor object
```

```
55      # IF NOT EXISTS prevents duplicate tables
56      headers = '''
57          CREATE TABLE IF NOT EXISTS device_data (
58              device integer NOT NULL,
59              peripheral integer NOT NULL,
60              rssi real NOT NULL,
61              PRIMARY KEY (device, peripheral)
62          )'''
63      cur.execute(headers)
64      pass
65
66  def create_line(con, dev_name, periph, mean):
67      '''Inserts a data line into the database'''
68      cur = con.cursor()
69      # REPLACE INTO ensures there are no duplicates
70      data_line = '''
71          REPLACE INTO device_data (
72              device, peripheral, rssi
73          ) VALUES (?, ?, ?)'''
74      cur.execute(data_line, (dev_name, periph, mean))
75      pass
76
77  txt_list = glob.glob('0*.txt') # Lists all device text files
78
79  sqlite3.register_adapter(np.int32, int)
80  con = db_connect() # Establishes database connection
81  create_table(con)
82
83  # Loops over text files to unpack data
84  for i in range(len(txt_list)):
85      file_name = txt_list[i]
86      data = np.genfromtxt(
87          file_name, delimiter=',',
88          skip_header=1, skip_footer=1, dtype=int   ) # Imports
               txt data
89      dev_name = data[-1,-1]
90
91      all_periphs = np.unique(data[:,0]) # Lists all peripherals
            under a device
92      for periph in all_periphs:  # Loops to find each mean RSSI
93          mean = reduce_data(periph, data, b_width, n_points,
               buff, h_thresh)
94          create_line(con, dev_name, periph, mean) # Inserts
               data
```

```
95          con.commit() # Commit changes
96
97  print('Unpacking Complete')
```

After the database has been created, a second Python program is run to create a matrix plot of P(dBm) values.

```
1   import numpy as np
2   import sqlite3
3   import glob
4
5   def db_connect():
6       '''Creates a connection to the database'''
7       con = sqlite3.connect('database.sqlite')
8       con.row_factory = sqlite3.Row
9       return con
10
11  def read_distinct(con):
12      '''Returns distinct values from device column, i.e a list
            of all devices'''
13      cur = con.cursor()
14      cur.execute('''
15          SELECT DISTINCT device
16          FROM device_data
17          '''     )
18      all_devs = [ line['device'] for line in cur.fetchall() ] #
            List of devices
19      #all_devs = [int.from_bytes(dev[0],byteorder='little') for
            dev in byte_dev]
20      return all_devs
21
22  def read_rssi(con, dev, periph):
23      '''Reads the RSSI corresponding to a given device and
            peripheral'''
24      cur = con.cursor() # Creates a cursor object
25      cur.execute('''
26          SELECT rssi
27          FROM device_data
28          WHERE device={d} AND peripheral={p}
29          '''.format(d=dev, p=periph)     )
30      result = cur.fetchone()
31      if result != None:
32          return result['rssi']
33
34  con = db_connect() # Establishes database connection
```

```
35  all_devs = read_distinct(con) # Returns list of all devices
36  sig_arr = np.zeros(( len(all_devs), len(all_devs) )) # To hold
        signal strength
37  # Assigns data to array
38  for i, dev in enumerate(all_devs):
39      for j in range(i):
40          periph = all_devs[j] # Cannot use enumerate due to
                limited j range
41          sig_arr[i,j] = read_rssi(con, dev, periph) # Lower
                diagonal
42          sig_arr[j,i] = read_rssi(con, periph, dev) # Upper
                diagonal
43
44  np.save('sig_arr.npy', sig_arr)
45  np.save('all_devs.npy', all_devs)
46  print('Assingment complete, array created')
```

D. Reordering Algorithm Code

The first half of our reordering algorithm runs in Python. It generates a preprocessed matrix when given a matrix plot of P(dBm) values.

```
1   import numpy as np
2   from scipy.spatial import distance
3
4   def shuffle(in_arr, num, shuff_idx):
5       '''Shuffles the rows and columns of an array using a given
            set of indices.
6       in_arr is the input array, shuff_idx is the shuffling
            sequence.'''
7       out_arr = np.zeros((num, num))
8       for i in range(num):
9           out_arr[i] = in_arr[shuff_idx[i], shuff_idx] # Fancy
                indexing for j
10      return out_arr
11
12  def average(in_arr):
13      '''Averages the lower and upper triangular halves of the
            matrix'''
14      sig_arr = np.copy(in_arr) # Ensures original array isn't
            modified
15      ave_arr = (sig_arr + sig_arr.T) / 2
16      fill_value = np.percentile(ave_arr, 95) # Value to fill
            diagonal
```

```
17      np.fill_diagonal(ave_arr, fill_value)
18      return ave_arr
19
20  def preprocess(dbm_arr):
21      '''Convenience function to generate pre-processed array
            from raw data'''
22      ord_arr = 10**(dbm_arr/10) * 10**3 # Converts to SI
23      np.fill_diagonal(ord_arr, 0)
24      fill_value = np.percentile(ord_arr, 95)
25      np.fill_diagonal(ord_arr, fill_value) # Fills diagonal
            with 95th percentile
26      ave_arr = average(ord_arr) # Averages across diagonal
27      return ave_arr
28
29  # Loads data
30  num = 10 # Number of devices
31  dbm_arr = np.load('sig_arr.npy')[:num, :num] # Loads data
32  # We can reshuffle the matrix to test different input
        orderings by changing...
33  disorder_idx = np.array(np.arange(num)) # ... a set of
        shuffling indices
34
35  # Calculates preprocessed matrix
36  dbm_arr = shuffle(dbm_arr, num, disorder_idx)
37  ave_arr = preprocess(dbm_arr) # Returns pre-processed matrix
38
39  # Saves preprocessed array
40  np.save('preproc_arr', ave_arr)
```

The second half of our reordering algorithm runs in R, and relies on the software package described in [51]. It uses the preprocessed matrix to calculate a distance matrix, then applies the single-linkage hierarchical clustering algorithm first formalised by Gruvaeus and Wainer. It also calculates the value of Spearman's coefficient for a resultant ordering.

```
1  library(RcppCNPy)
2  library(seriation)
3  library(Matrix)
4
5  # Imports preprocessed array as a matrix
6  sig_arr <- npyLoad("preproc_arr.npy")
7  d_1 <- dist(sig_arr, method='manhattan') # Calculates the
       distance matrix
8  o_1 <- seriate(d_1, method = "GW_single") # Applies seriation
       method
9  ord <- get_order(o_1)
```

```
10  print(ord) # Returns order of transceivers
11  dendo <- hmap(d_1, method = "GW_single") # Creates dendrogram
12
13  # Calculates Spearman's coefficient for a 10 X 10 dataset.
14  ref <- seq(1, 10)
15  spear <- cor(ord, ref, method=c("spearman"))
16  print(spear)
```