UPPSALA
UNIVERSITET

IT 23 064
Degree project 30 credits
July 2023

ORF: On-board Radiometric Fingerprinting

Mikolai-Alexander Gütschow

Master's Programme in Embedded Systems

UPPSALA
UNIVERSITET

ORF: On-board Radiometric Fingerprinting

Mikolai-Alexander Gütschow

Abstract

Radiometric fingerprinting systems leverage unique physical-layer signal characteristics originating from individual hardware imperfections to identify transmitter devices. However, existing systems are limited by the need for specialized hardware and non-trivial computations to capture physical-layer signals and extract fingerprinting features, hindering their pervasive deployment. ORF, for the first time, demonstrates the feasibility of implementing an entire radiometric fingerprinting system on a low-cost and low-power embedded system on chip, with over 92% average accuracy on the task of identifying one out of 32 different transmitter devices.

Faculty of Science and Technology

Uppsala University, Uppsala

Supervisor: Wenqing Yan Subject reader: Christian Rohner

Examiner: Pontus Ekberg

Acknowledgments

I would like to thank my supervisor Wenqing Yan for the opportunity to work on this project in the first place, and especially for her continuous support and guidance throughout the thesis work. Thanks as well to my reviewer Prof. Christian Rohner who gave valuable feedback and suggestions for further investigations. Last but not least, I want to thank my friends Ankit, Axel, George, Gholam and Ibro, who apart from the office also shared many lunch and fika breaks with me and made the time at university especially enjoyable.

Contents

List of Figures

List of Tables

List of Acronyms

AoA angle of arrival

AoD angle of departure

ASK amplitude-shift keying

BLE Bluetooth Low Energy

CFO carrier frequency offset

COTS commercial off-the-shelf

CRC cyclic redundancy check

CTE constant tone extension

DFE direction finding extension

FSK frequency-shift keying

MAC medium access control

ML machine learning

OQPSK offset quadrature phase-shift keying

PHR PHY header

PHY physical layer

PLL phase lock loop

PSK phase-shift keying

QPSK quadrature phase-shift keying

RAM random-access memory

ROM read-only memory

SHR synchronization header

SoC system on chip

WSN wireless sensor network

1 Introduction

Wireless sensor networks (WSNs) have been widely deployed in various applications, aiming to provide valuable insights through real-time data collection, analysis, and decision-making. Especially when employed in safety-critical scenarios, ensuring the authenticity and authorization of data collected in a WSN is a major concern: Data transmitted over the air can be easily forged, allowing a potential attacker to inject false data and harm the overall system. Traditional approaches for authentication build on cryptographic methods like encryption or signing of messages. For the hardware employed in WSNs, which is commonly constrained in energy and computational power, those tasks are quite demanding, and they usually lead to a certain transmission overhead. More importantly, the distributed nature of such networks, the lack of a central unit communicating with all devices and the often unidirectional nature of the communication all lead to the cryptographic key distribution and management being a challenging problem.

Radiometric fingerprinting is an alternative approach for authentication. It identifies transmitter devices not based on cryptographic methods, but by capturing certain characteristics of the received signal that stem from hardware imperfections of the individual transmitter devices [1]. Thereby, this approach avoids both the key distribution challenge and the transmission overhead.

In order to analyze its characteristics, the physical-layer signal needs to be captured in a structure-preserving way (a process commonly referred to as *I/Q sampling*) and be made available to a software stack. However, the commercial off-the-shelf (COTS) devices which typically constitute a WSN make use of a hardware-accelerated receiver pipeline and only present the decoded bit-level information to software, discarding the physical-layer signal information. Up until now, the high cost of specialized hardware needed for I/Q sampling has effectively hindered the deployment of radiometric fingerprinting systems in a distributed architecture [2].

A different application which requires analysis of physical-layer signals is the direction finding extension (DFE) added to the Bluetooth Low Energy (BLE) specification in version 5.1 [3]. Within the last years, several new COTS device models with support for BLE DFE have been presented. Depending on the system design, the signal analysis can be performed as part of the receiver hardware or be delegated to software. The latter case has the potential of enabling other applications based on the available I/Q sampling information.

ORF builds on this possibility and, for the first time, shows the feasibility of an *on-board radiometric fingerprinting* system, deployed in its entirety on a COTS device. It leverages the flexibility of the I/Q sampling functionality on an nRF52833 system on chip (SoC) and extracts a radiometric fingerprint from a 480 µs long IEEE 802.15.4 transmission using a typical coherent-receiver pipeline implemented in software. The extracted fingerprint is finally attributed to a known device using an on-board Random Forest classifier.

The overall system reaches an average precision of more than 92% on a challenging dataset with 32 devices of four different types, while occupying not more than 75 % of the available memory. The classification of a single received frame takes roughly one second and consumes about 12 mJ in terms of energy.

The remainder of this report is organized as follows: Section 2 introduces common radiometric fingerprinting system architectures and concepts used in their implementation. The system design of ORF is briefly summarized in Section 3, listing several challenges arising from its on-board deployment. Section 4 goes into detail on the implementation of the on-board radiometric fingerprinting system. The performance of ORF is evaluated in Section 5 and final conclusions are drawn in Section 6.

2 Background

This section first gives an overview of the three stages of a common radiometric fingerprinting architecture and then provides relevant background information for the implementation of these individual stages in ORF.

2.1 A Primer on Radiometric Fingerprinting

Device fingerprinting in general is used to increase the security in wireless networks by complementing cryptographic approaches with information obtained from unique transmitter characteristics. These characteristics can be extracted from the transmitter's behavior at several layers of the network stack. In general, the amount of transmitter-specific information and the granularity of identification improves as lower layers are considered: While medium access control (MAC) layer information can be used to efficiently distinguish between devices of different types, the *radiometric fingerprint* extracted at the physical layer (PHY) even allows for individual device identification by capturing the peculiarities of the physical-layer signal caused by manufacturing imperfections of the transmitter hardware [4].

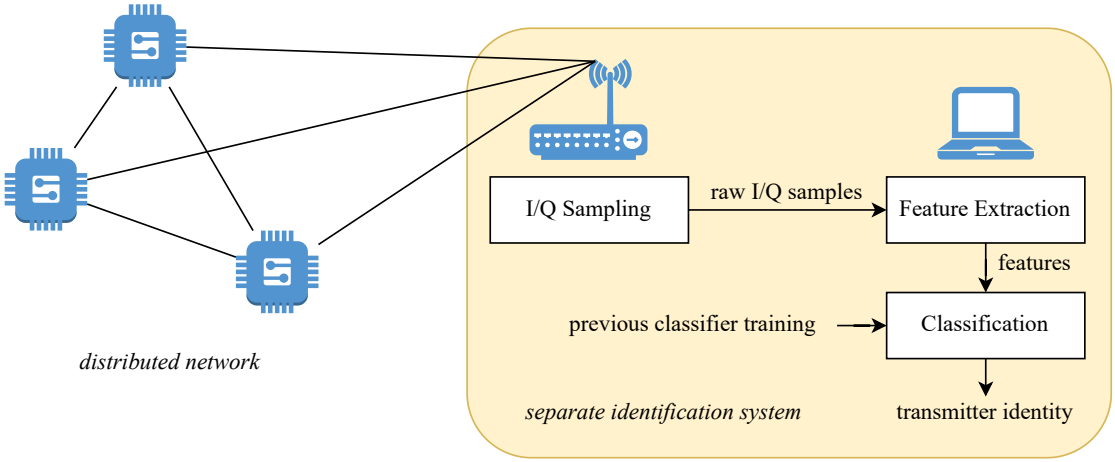


Figure 2.1: Traditional radiometric fingerprinting system architecture. Specialized and expensive hardware is used for I/Q sampling, feature extraction and device identification.

A typical fingerprinting system implementation acts in three stages upon reception of a signal, as depicted in Figure 2.1 and detailed in the following three sections:

1. The signal is *sampled* with high resolution both in time and amplitude for subsequent digital processing.
2. The acquired signal samples are processed in order to *extract features* which capture the uniqueness of the transmitter device as well as possible.
3. Those features are used as input to a previously trained *classifier* which judges the corresponding device identity.

Previous work has shown the effectiveness of radiometric fingerprinting techniques to enhance the security of wireless networks [4], [5]. They have been implemented for a variety of physical-layer (PHY) protocols, including IEEE 802.11 (Wi-Fi) [5]–[7], IEEE 802.15.4 (ZigBee, Thread) [8]–[11], LoRa [12], [13] and BLE [14].

Although the benefits of radiometric fingerprinting techniques are most apparent in distributed networks of small COTS devices, the employed system architecture typically relies on a single central receiver which is responsible for device identification [6], [8]. This discrepancy comes from the fact that I/Q samples are traditionally not available on COTS devices, resulting in the need of costly specialized hardware.

2.2 Sampling Wireless Signals

2.2.1 Modulation of Radio Waves

A basic signal wave $a(t)$ can be described by a sine with a certain *amplitude* A, *frequency* f and *phase* ϕ as shown in Figure 2.2:

$$a(t) = A \cdot \sin(2\pi f t - \phi) \tag{2.1}$$

The *period* $T = \frac{1}{f}$ of the wave is the reciprocal of the frequency and the *wave length* λ is proportional to the period and the transmission velocity v in a given medium: $\lambda = T \cdot v = \frac{v}{f}$. Radio waves travel through air at approximately the speed of light, i.e., $v \approx c \approx 3 \times 10^8\,\mathrm{m/s}$.

In order to encode information in the signal $a(t)$, its characteristics must change over time in a specified way known to the transmitter and the receiver, depending on the information to be transmitted. This process is called *modulation*. Any of the three components frequency, amplitude and phase can change over time, resulting in, respectively, frequency-shift keying (FSK), amplitude-shift keying (ASK), phase-shift keying (PSK); or a combination thereof [15].

Signals generated in such a way are known as *baseband signals*. For wireless transmission, they are usually modulated again using a so-called *carrier wave* to form a *passband signal* $s(t)$ with a fixed *carrier frequency* f_{tx}, which is significantly higher than the baseband

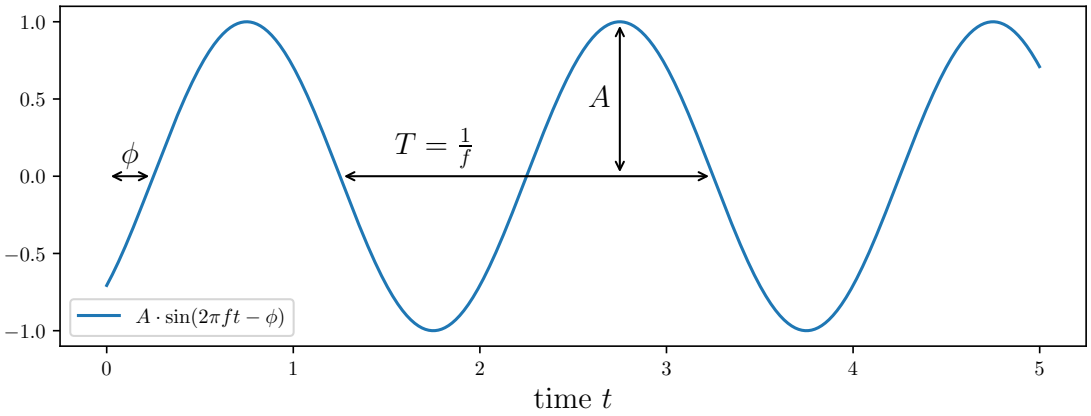


Figure 2.2: Generic sine wave with amplitude A, frequency f and phase ϕ.

frequency f. Since the carrier frequency is fixed, there are still two parameters which are allowed to change, namely the amplitude and phase of $s(t)$.

According to the Harmonic Addition Theorem [16], a linear combination of a cosine and a sine with the same (angular) frequency ω results in a single cosine wave with precisely a scaled amplitude and phase shift:

$$I \cdot \cos(\omega t) - Q \cdot \sin(\omega t) = c \cdot \cos(\omega t + \varphi)$$

with

$$c = \operatorname{sgn}(I) \cdot \sqrt{I^2 + Q^2},\ \varphi = \tan^{-1}\left(\frac{Q}{I}\right). \tag{2.2}$$

If I and Q now change over time, the amplitude c and phase φ of the resulting cosine do so as well.[1]

Therefore, actually *two* baseband signals can be used as $I(t)$ and $Q(t)$ and modulated together to the carrier frequency at the same time:

$$s(t) = I(t) \cdot \cos(2\pi f_{tx} t) - Q(t) \cdot \sin(2\pi f_{tx} t) \tag{2.3}$$

$I(t)$ and $Q(t)$ are commonly called the *in-phase* and *quadrature component* of $s(t)$, respectively. In a typical *direct-conversion* transmitter design, both carrier signals are produced by a single local oscillator (LO) running at f_{tx}. A delayed copy of the in-phase carrier with phase difference 90° is used as the quadrature-phase carrier, since $\cos\left(2\pi f_{tx} t + \frac{\pi}{2}\right) = -\sin(2\pi f_{tx} t)$.

At the receiver, the baseband signals are recovered by multiplying $s(t)$ with the known carriers at the approximate carrier frequency $f_{rx} \approx f_{tx}$ and low-pass filtering the result:

$$\frac{1}{2} I(t) \approx LPF\{s(t) \cdot \cos(2\pi f_{rx} t)\},\ \frac{1}{2} Q(t) \approx LPF\{s(t) \cdot \sin(2\pi f_{rx} t)\} \tag{2.4}$$

[1] $I \cdot \cos(\omega t)$ and $Q \cdot \sin(\omega t)$ are idealized as being orthogonal to each other. This so-called *narrowband assumption* only holds as long as I and Q change very slowly in relation to the carrier frequency.

Figure 2.3 shows a simplified block diagram of the modulation and demodulation parts of the two baseband signals $I(t)$ and $Q(t)$ as defined by Equations 2.3, 2.4.

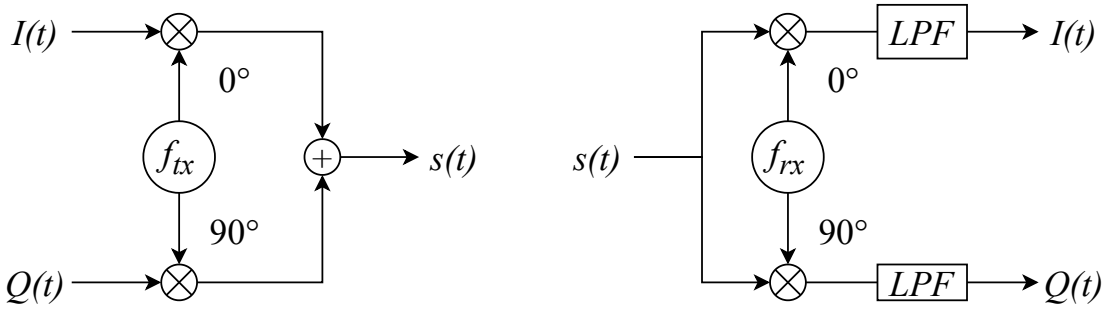
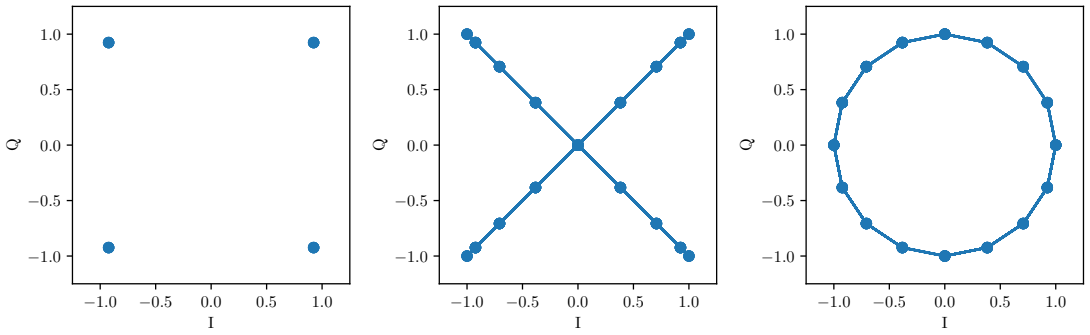


Figure 2.3: I/Q modulation and demodulation.

For further digital processing, the baseband signals are sampled at a given frequency of at least $2f$ according to Nyquist, where f is the maximum frequency of the baseband signals. This process is called *in-phase and quadrature sampling* or *I/Q sampling* for short. Each I/Q sample S_k can be either represented in 2-D cartesian coordinates (I_k, Q_k) or as a phasor in polar coordinates consisting of amplitude c_k and phase φ_k as given in Equation 2.2, with $S_k = c_k \cdot e^{i \cdot \varphi_k}$. Being a point in a two-dimensional space, I/Q samples are often conveniently represented as complex numbers, with the in-phase component treated as the real part and the quadrature component as the imaginary one.

The information-carrying symbols for a given phase or amplitude modulation scheme can be represented as ideal *constellation points* P in the I/Q plot. As an example, quadrature phase-shift keying (QPSK) has four constellation points $P_i \in \{e^{i \cdot \frac{\pi}{4}} \mid i \in 1, \ldots, 4\}$ or, equivalently, $P_i = \left(\pm\frac{\sqrt{2}}{2}, \pm\frac{\sqrt{2}}{2}\right)$ after normalization to amplitude 1. The ensemble of these points P_i is called *constellation diagram* and plotted for QPSK in Figure 2.4a.



(a) QPSK constellation diagram.

(b) I/Q plot of oversampled ideal QPSK signal.

(c) I/Q plot of oversampled ideal OQPSK signal.

Figure 2.4: Examples of I/Q plots.

Since the baseband signal is sampled at a higher frequency than f, several of the obtained samples are taken at times where the signal is changing between two constellation points. The resulting I/Q plot for a QPSK signal resembles an X as simulated in Figure 2.4b with 8x oversampling.

To avoid large amplitude fluctuations due to phase changes of as much as 180°, offset quadrature phase-shift keying (OQPSK) is often preferred over QPSK in practice. In OQPSK, one signal component is delayed by half a symbol period, effectively restricting the phase shift to 90°. When sampling such a signal, the I/Q plot resembles a circle as depicted in Figure 2.4c, since there is no immediate change between opposite constellation points.

2.2.2 Effects of Hardware Imperfections and the Wireless Channel

At the transmitter, the passband signal $s(t)$ is generated in such a way that the transmitted I/Q symbols match the constellation points as closely as possible. However, hardware imperfections manifest themselves in deviations of S_k from P_i in amplitude and phase; as well as in a deviation of f_{tx} from the desired carrier frequency. The former can be explained by *I/Q phase imbalance* and *I/Q offset* while the latter is due to non-perfect high-frequency oscillators.

I/Q imbalance stems from the fact that the two baseband signals $I(t)$ and $Q(t)$ are processed in separate branches in hardware. As depicted in Figure 2.3 above, the two orthogonal carrier waves are generated using a single oscillator and a 90° phase shift. However, in the analog domain, the phase difference between the carriers is never exactly 90° [17], [18]. Since the same applies on the receiver side, the observed I/Q imbalance in the received signal is a combination of the hardware effects of both the transmitter and the receiver.

When the carrier waves additionally expose a DC bias, i.e., a non-zero mean amplitude, the entire constellation diagram is shifted away from the zero origin, resulting in an I/Q offset. Physical-layer standards usually define a combined maximum error tolerance for the amplitude and phase deviations based on the *error vector* between S_k and P_i, $\vec{E_k} = S_k - P_i$.

The carrier waves used for modulation or demodulation at the transmitter or receiver, respectively, are generated in hardware by a local oscillator (LO) running at approximately the frequency defined by the PHY protocol. However, in practice, two different oscillators never coincide exactly in frequency due to hardware imperfections and other environmental factors. There is thus in general a fixed *carrier frequency offset (CFO)* $f_{tx} - f_{rx}$ between transmitter and receiver. Here as well, physical-layer standards usually account for these imperfections and define a certain tolerance for the carrier frequency.

The amplitude, phase and frequency distortions observed at the receiver are however not only a combination of fixed transmitter and receiver hardware imperfections, but

also depend on effects of the often varying wireless channel. In particular, amplitude is affected by distance, noise and multipath effects, frequency is affected by Doppler shifts in case of relative movement between transmitter and receiver, and the exact sampling start and interval at the receiver might result in further amplitude and phase deviations.

For successful radiometric fingerprinting, all of those propagation effects introduced by the wireless channel must be compensated for during receiver synchronization, without loosing the information of the actual hardware imperfections. To that end, ORF implements a typical *coherent demodulation receiver* pipeline which is described in detail in Section 4.2.

2.2.3 Direction Finding as ORF Enabler

Previous work on radiometric fingerprinting relied on complex vector signal analyzer (VSA) hardware as well as on extensive digital signal processing (DSP) and machine learning (ML)-based classifier algorithms employed on the receiver's side [6]–[12]. In contrast, ORF employs an embedded system design running on a standard COTS device, allowing radiometric identification to be integrated in a cheap way into existing systems.

In general, application-oriented systems using a given communication standard are only interested in the decoded version of wirelessly transmitted data, which is why COTS devices usually come with a hardware-accelerated receiver pipeline outputting at most some reception metric such as RSSI in addition to the decoded payload of a frame. In this case, the device software is incapable of getting access to the raw physical-layer signal measurements which are the basis for radiometric fingerprinting techniques.

However, this has recently started to change with the implementation of the major new feature of the Bluetooth® Core v5.1 specification on COTS devices: The direction finding extension (DFE) designed to enhance location services employed over BLE relies on phase-shift observations on the physical-layer signal received at two or more antennas. For software-side implementation of this feature, devices with support for BLE DFE must therefore provide access to raw I/Q samples.

The remainder of this section will explain the theory behind direction estimation based on radio signals and introduce the BLE DFE based on [3], focusing on the parts that are of interest for ORF.

2.2.3.1 Direction Finding Theory

A radio signal like the one described by Equation 2.1 travels outwards from a transmitter antenna T at velocity v on a trajectory following the surface of an expanding sphere. The signal reaches a receiver antenna R_1 placed at a certain distance d_1 from the transmitter at time $t_1 = \frac{d_1}{v}$. If a second receiver antenna R_2 is placed at a known distance d_R from R_1, the same signal will be received at a potentially different time $t_2 = \frac{d_2}{v}$, depending

on the distance d_2 of R_2 from T. The time difference $\Delta t = t_2 - t_1$ is perceived as a phase shift $\phi = 2\pi f \Delta t = \frac{2\pi}{\lambda}(d_1 - d_2)$ of the signal received at R_2 compared to the signal received at R_1.

If $d_R << d_1, d_2$, the wavefront arriving at R_1 and R_2 can be approximated by a plane. Further assuming that d_R is not a multiple of the wavelength λ, the phase shift ϕ can be used to deduce the signal direction as the angle φ by basic trigonometry as $\varphi = \arccos\left(\frac{d_1 - d_2}{d_R}\right) = \arccos\left(\frac{\phi \cdot \lambda}{2\pi d_R}\right)$.

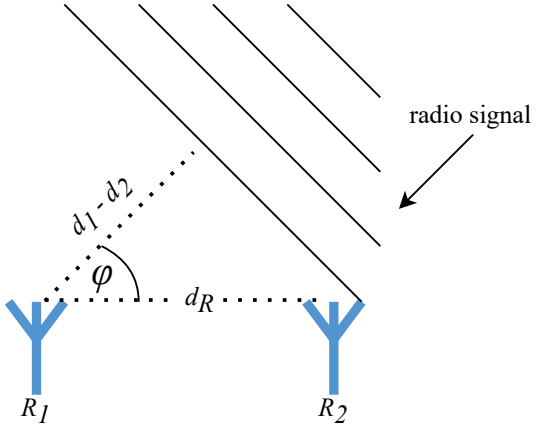


Figure 2.5: The angle φ of a radio signal can be estimated using two antennas placed at a known distance d_R.

In practice, the signal is usually not sampled simultaneously at both antennas. Instead, the receiver switches between the antennas and receives the signal at one of them at a time. As long as the signal is periodic, the time difference translates to a constant phase shift between the signals received at both antennas.

The same principle applies if two transmitter antennas located at a known distance d_T from each other take turns transmitting the same signal to a single receiver antenna. As long as the receiver has knowledge about d_T and which part of the received signal is emitted by which transmitter antenna, the receiver can calculate the angle φ. These setups can also be extended in a straightforward way to larger antenna arrays with more than two antennas at the receiving or transmitting end to allow for a more accurate direction estimation.

2.2.3.2 Bluetooth Direction Finding

Bluetooth Direction Finding defines both an angle of arrival (AoA) and an angle of departure (AoD) mode, corresponding to deploying several antennas at the receiver or at the transmitter, respectively. Consequently, in AoA mode, the receiver is supposed to

switch between antennas, while this responsibility is taken by the transmitter in AoD mode.

Since BLE uses FSK modulation, i.e., two different frequencies to transmit zeros and ones, and since changing the frequency in turn changes the wavelength, which is a critical factor in the direction calculation, the direction finding functionality in BLE is not performed over the frame payload itself. Instead, it is implemented as an optional part called constant tone extension (CTE) appended at the end of the frame. The CTE contains only digital ones, resulting in a single frequency and wavelength of this part of the transmitted signal.

The length of the CTE is configurable in units of 8 µs to be between 16 and 160 µs. It consists of a 4 µs long *guard period* and a 8 µs long *reference period*, followed by the *switching period* consisting of alternating switch and sample slots of either 1 or 2 µs each. As the name implies, the switch slots give time for the receiver (in AoA mode) or the transmitter (in AoD mode) to switch between different antennas, while I/Q sampling is performed during the sample slots. When the CTE is present in a BLE frame, the CTEInfo field in the BLE frame header is used to convey information about the length of the CTE, the DFE mode (AoA or AoD) and the length of individual switch and sample slots.

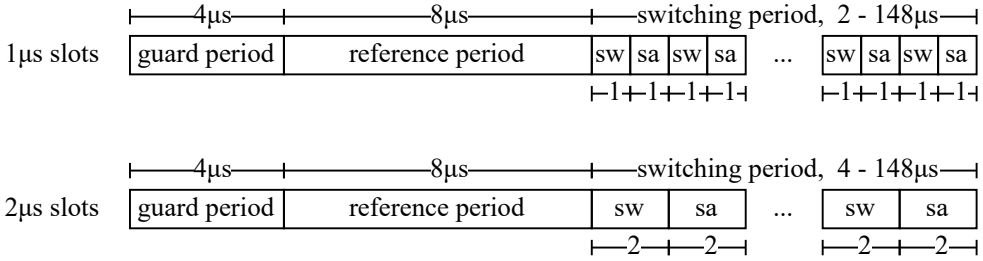


Figure 2.6: Two possible CTE formats depending on the slot length.

According to the specification, a receiver with support for BLE DFE must be able to take one I/Q sample every microsecond during the reference period and one I/Q sample per sample slot. This translates to a required sampling frequency of 1 MHz during the reference period and 500 kHz (250 kHz for 2 µs slots) afterwards.

2.3 IEEE 802.15.4 Signals

ORF recognizes devices communicating according to the IEEE 802.15.4 specification, which is a technical standard designed for low-rate wireless personal area networks (LR-WPAN). 802.15.4 defines both the PHY and MAC layer for these networks [19] and is the basis for many higher layer protocols which are commonly used in Internet of Things (IoT) deployments. Examples are ZigBee, 6LoWPAN, Thread and SNAP.

In order to motivate the coherent-receiver pipeline and the feature extraction implementation of ORF for IEEE 802.15.4-compliant physical-layer signals, this section briefly summarizes the physical-layer aspects of this protocol.

2.3.1 Spreading and Modulation

The standard defines operation in different modes and frequency bands. ORF targets the most commonly used direct sequence spread spectrum (DSSS) mode around 2.45 GHz which employs OQPSK as its modulation technique.

At the sender, the binary data to be transmitted is first split to symbols of four bits each, which are then individually mapped to one of 16 pre-defined 32-bit pseudo-random (PN) *chip* sequences. Such a chip sequence is split in even- and odd-indexed chips that are in turn modulated onto the in-phase (I) and quadrature-phase (Q) carrier, respectively. The Q chips are however delayed by the chip interval T_c, resulting in OQPSK modulation. The pulse shape for modulation is a half-sine defined by

$$p(t) = \pm\text{rect}\left(\frac{t - T_c}{2T_c}\right) \cdot \sin\left(\pi \frac{t}{2T_c}\right). \tag{2.5}$$

Chips representing ones are shaped using a positive amplitude and those representing zeros with a negative amplitude.

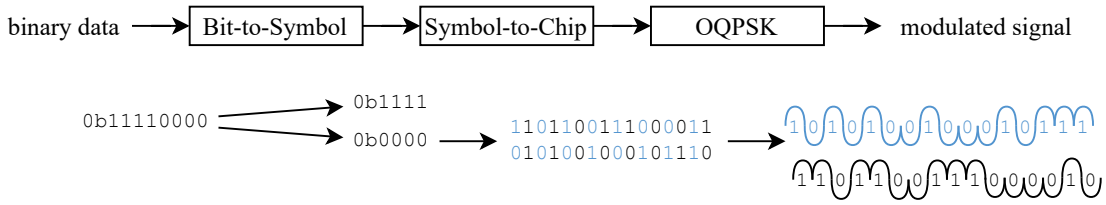


Figure 2.7: IEEE 802.15.4 modulation example for the symbol `0b0000`.

The chip interval is defined as $T_c = 0.5\,\mu\text{s}$, which corresponds to a chip rate of 2Mchips/s. Since every 4-bit symbol is represented by 32 chips, this corresponds to a data rate of 250 kbit/s. One byte thus takes 32 microseconds to be transmitted.

2.3.2 Frame Format

A frame transmitted via the 802.15.4 physical layer conforms to the format illustrated in Figure 2.8. It consists of the following basic components:

- a five-byte synchronization header (SHR) with fixed content, which allows a receiving device to synchronize and lock onto the bit stream;
- a one-byte PHY header (PHR), which contains frame length information; and

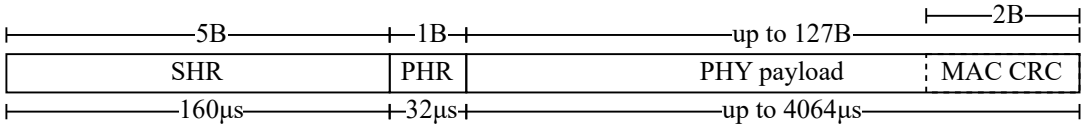


Figure 2.8: IEEE 802.15.4 frame format.

- a variable length payload of at most 127 B, which usually carries the MAC sublayer frame and commonly ends with a 2 B MAC cyclic redundancy check (CRC).

The longest possible frame thus takes $\frac{(5+1+127)\cdot 8\,\text{bit}}{250\,\text{kbit/s}} = 4.256\,\text{ms}$ to transmit.

2.4 The Classification Problem

In general, the goal of classification is to attribute an observation to one out of a finite amount M of possible categories, or *classes*, $C = \{C_i \mid i \in 1, \ldots, M\}$. Usually, the observation is first analyzed and quantified into *features* that try to capture the essence of the observation in a lower-dimensional vector. The feature vector $\vec{f}$ is then used as input to the classification algorithm, or *classifier*, c, which implements the function $c(\vec{f}) \in C$.

Since it is in general hard to define the decision boundaries for c manually, supervised ML algorithms are widely used to define c based on some pre-classified (*labeled*) data [20]: The classifier is *trained* on a *training set* $D_{\text{train}} = \{(\vec{f_i}, C_i)\}$ in such a way that for most $(\vec{f_i}, C_i) \in D_{\text{train}} : c(\vec{f_i}) = C_i$ holds. As long as the training set is a statistically accurate representation of the real observations, c can be expected to generalize well.

Commonly, the performance of a given classifier c is quantified using a *test set* D_{test} with $D_{\text{test}} \cap D_{\text{train}} = \emptyset$, i.e. with labeled data that is not used during training. There are several metrics to evaluate the classification outcome. For this work, the simple statistical metrics *precision* and *recall* are considered. These metrics can be easily explained using the common concept of the *confusion matrix* CM of size $M \times M$ representing a summary of the classification outcome. In that matrix, the rows represent the true classes, while the columns represent the predicted classes. Each entry $CM_{ij} = |\{\vec{f_i} : (\vec{f_i}, C_i) \in D_{\text{test}} \wedge c(\vec{f_i}) = C_j\}|$ corresponds to the number of samples of D_{test} belonging to the (true) class C_i and classified as the (predicted) class C_j. The confusion matrix of an ideal classifier would be a diagonal matrix with $CM_{ij} = 0,\ i \neq j$, i.e., produce no misclassification.

Precision is the fraction of correctly classified samples for a certain predicted class C_j.

$$\text{precision}_j = \frac{CM_{jj}}{\sum_{i=1}^{M} CM_{ij}}$$

Conversely, recall is the fraction of correctly classified samples of a certain true class C_i.

$$\text{recall}_i = \frac{CM_{ii}}{\sum_{j=1}^{M} CM_{ij}}$$

To assess the classifier performance over all classes, both the *macro*-average and the *worst-case* value is reported for each of those metrics. The macro-average is a simple arithmetical mean across classes. It gives equal weight to all classes making it a good option for balanced classification tasks.

$$\text{precision}_{\text{macro}} = \frac{1}{M} \sum_{j=1}^{M} \text{precision}_j$$

$$\text{recall}_{\text{macro}} = \frac{1}{M} \sum_{i=1}^{M} \text{recall}_i$$

The worst-case is the minimum value of a metric across classes and thereby reports the respective metric for the class where the classifier performed the worst.

$$\text{precision}_{\text{wc}} = \min_{j=1,\ldots,M} \text{precision}_j$$

$$\text{recall}_{\text{wc}} = \min_{i=1,\ldots,M} \text{recall}_i$$

Another metric which is often used for ML models in general is the *accuracy*. It corresponds to the *micro*-average of both of the metrics precision or recall, which is calculated as the fraction of correctly classified samples over the whole test set:

$$\text{accuracy} = \frac{\sum_{i=1}^{M} CM_{ii}}{\sum_{i=1}^{M} \sum_{j=1}^{M} CM_{ij}}$$

Especially for imbalanced datasets, it is handy to consider the confusion matrix normalized either by row or column. Instead of containing absolute numbers, it then reports the fraction of samples with respect to true or predicted classes, respectively.

As an example, consider the three-class ($M = 3$) confusion matrix depicted in Figure 2.9. The performance metrics for this confusion matrix would be calculated as:

$$\begin{aligned}
\text{precision}_{\text{macro}} &= \frac{1}{3} \cdot \left(\frac{8}{8+0+5} + \frac{5}{2+5+0} + \frac{5}{0+0+5} \right) = 0.7766 \\
\text{precision}_{\text{wc}} &= \frac{8}{13} = 0.6154 \\
\text{recall}_{\text{macro}} &= \frac{1}{3} \cdot \left(\frac{8}{8+2+0} + \frac{5}{0+5+0} + \frac{5}{5+0+5} \right) = 0.7667 \\
\text{recall}_{\text{wc}} &= \frac{5}{10} = 0.5000 \\
\text{accuracy} &= \frac{8+5+5}{8+2+5+5+5} = 0.7200
\end{aligned}$$

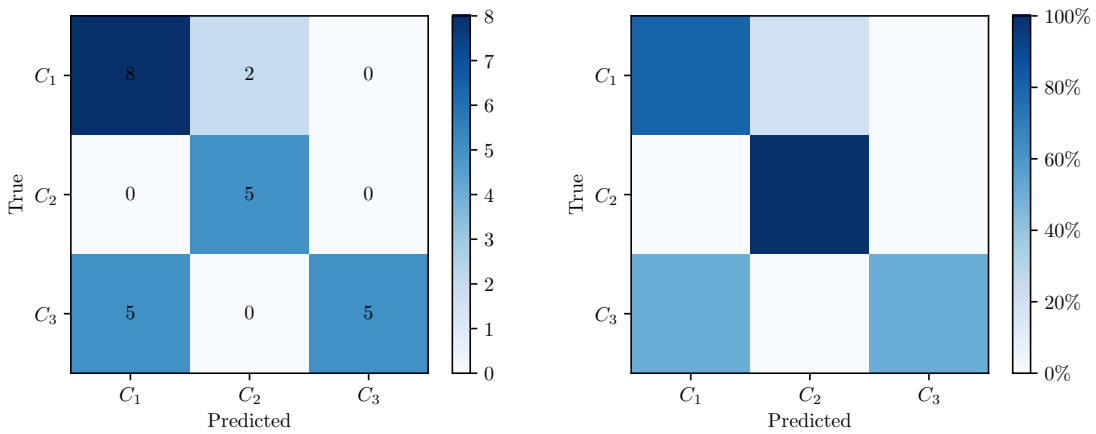


Figure 2.9: Example of a confusion matrix. The right matrix is an alternative representation of the left one obtained by normalizing over rows (true values).

3 System Design

The system design of ORF consists of a COTS device which performs I/Q sampling, feature (fingerprint) extraction with the help of a coherent demodulation receiver pipeline, and on-board classification using a simple ML model. The classifier model is trained off board on previously extracted features. Figure 3.1 gives an overview of the implemented system. In contrast to the typical architecture of previous systems depicted in Figure 2.1, one or even several identification devices can be cheaply integrated into existing network deployments.

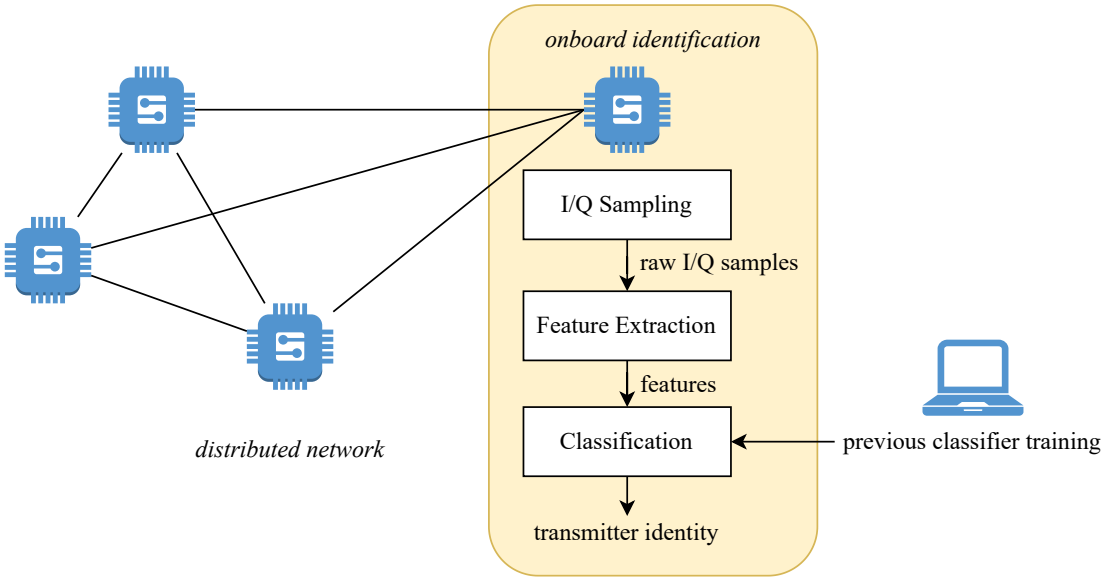


Figure 3.1: ORF system design. I/Q sampling, feature extraction and device identification are performed on one of the COTS devices constituting the distributed network.

3.1 Challenges

The goal of ORF is to showcase the feasibility of a complete radiometric fingerprinting system deployed on a COTS embedded device. This is challenging because of several restrictions imposed by such hardware:

- ***Raw signal data availability and quality.*** Raw signal measurements are available to the software stack in order to support Bluetooth DFE, which relies on phase shift measurements between different antennas, detected during antenna switching. According to the BLE specification, a receiver has to sample once every 2 or 4 µs (depending on the switch/sample slot size) of the optional CTE with a length of 16 to 160 µs at the end of a BLE frame (cf. Section 2.2.3.2). The use-case of radiometric fingerprinting however requires:
  - Continuous sampling at a single antenna without antenna switching;
  - A sampling rate of at least twice the data frequency (following Nyquist), preferably a higher oversampling rate for better data quality;
  - Sampling over a longer period, preferably a whole frame; and
  - Sampling independently of the used physical-layer protocol.

  Just the fact that a platform supports BLE DFE therefore does not automatically imply that it will be suitable for an on-board radiometric fingerprinting system. The first implementation stage (detailed in Section 4.1) investigates the quality of the data obtained on the selected hardware platform.
- ***Memory constraints.*** Limited random-access memory (RAM) space restricts the amount of data that can be kept and processed at once, again limiting the oversampling rate and/or the part of a received frame that can be considered for the fingerprint. It also affects the resources available during the feature extraction process as described in Section 4.2. The available read-only memory (ROM)/flash space in turn mainly limits the classifier model size, as detailed in Section 4.3.
- ***Time and energy cost.*** The fingerprinting classifier should work as close to real-time as possible, to be able to classify frames as they are received. However, the computational power of embedded devices is limited compared to the systems used in previous work. At the same time, energy consumption has to be taken into account, especially when employed on battery-powered devices. Being a prototype, time and energy efficiency are not the main goals of ORF. Nevertheless, both aspects are briefly evaluated in Section 5.5.

3.2 Hardware Platform

Nordic Semiconductor's **nRF52833 SoC** [21] is chosen as the hardware platform for ORF. It features a 2.4 GHz transceiver with support for BLE v5.1, 802.15.4-2006 as well as two proprietary protocols and is built around a 64MHz Arm® Cortex®-M4F with 512 kB of ROM and 128 kB of available RAM. As part of the nRF52 series, it is code-compatible to older devices of the series which are widely used for research and in commercial products.

Being a proof of concept, ORF shows the general feasibility of on-board radiometric fingerprinting. It is expected to be applicable to other COTS platforms too, as long as they provide access to raw I/Q signal measurements in sufficient quality.

4 Implementation

The implementation of ORF[1] is realized in three different *stages*, corresponding to the three parts of the typical radiometric fingerprinting system design depicted in Figure 3.1. During the first stage, the feasibility of obtaining I/Q samples in sufficient resolution is explored on the selected hardware platform and for the given physical-layer signals. The second stage comprised the definition of *features* derived from the I/Q samples, as well as an on-board implementation of the feature extraction pipeline. For the third stage, a classifier is trained off board and employed on the hardware platform for on-board radiometric-fingerprint classification. The three stages are described in detail in the following three sections.

4.1 Stage 1: I/Q Sampling

The goal of stage 1 is to investigate the accessibility of I/Q samples and to judge their suitability for radiometric fingerprinting on the selected SoC. This section elaborates on the system parameters for I/Q sampling available on the nRF52833 SoC and on the constraints imposed by the platform, thereby addressing the first set of challenges listed in Section 3.1.

4.1.1 BLE DFE Parameters and Constraints

Bluetooth Direction Finding support on the nRF52833 is described in its Product Specification [21, Sec. 6.8.12] and further detailed in a White Paper [22]. It is realized by providing software access to samples of the raw incoming signal, in addition to the decoded payload data of received frames. As all peripherals on this SoC, the radio is controlled by memory-mapped configuration registers. This part will list the parameters available for DFE which are configured in a certain way to address the challenges listed in Section 3.1.

Continuous Sampling at Single Antenna. The SoC supports both the AoA and AoD mode of Bluetooth DFE as explained in Section 2.2.3. Setting `DFEMODE.DFEOPMODE` to `AoD` has the effect of disabling antenna switching at the receiver [21, Sec. 6.8.12.2]. Although the BLE standard only requires samples to be taken during the sample slots, the nRF hardware implementation samples continuously throughout the switching period

[1]The source code is publicly available under MIT license at https://codeberg.org/ialokim/ORF.

[22, Table 2]. However, the first I/Q sample after the reference period is always aligned with the first sample slot, thereby skipping the preceding switch slot [22, Sec. 3.2]. This is empirically confirmed and further investigated with a number of small experiments in the next section.

Oversampling. The sampling rate depends on a combination of parameters. `DFECTRL1.TSAMPLESPACINGREF` defines the sample interval during the reference period of length 8 µs at the beginning of the CTE. If `CTEINLINECONF.CTEINLINECTRLEN` is enabled, the sampling rate after the first 8 µs is adapted during frame reception according to the information contained in the CTEInfo part of the BLE frame header. If disabled, manual configuration is possible through setting `DFECTRL1.TSAMPLESPACING` [21, Sec. 6.8.12.7]. The smallest possible interval for both parameters is 125 ns which corresponds to a sampling rate of $\frac{1}{125\,\text{ns}} = 8\,\text{MHz}$.

Sampling Start. The start of the sampling operation is configurable through `DFECTRL1.DFEINEXTENSION` either to right after the CRC (where the CTE is expected for BLE frames), or to the `ADDRESS` event which is triggered at the beginning of the frame payload [21, Sec. 6.8.12.4]. Additionally, an offset from the start trigger can be configured through `DFECTRL2.TSAMPLEOFFSET`. To start sampling as soon as possible, it is left at the default value of zero.

Sampling Duration. The sampling duration is upper-bounded by `DFECTRL1.NUMBEROF8US` in multiples of 8 µs, while `DFEPACKET.MAXCNT` restricts the maximum amount of recorded samples. Just by the amount of bits corresponding to each parameter in the configuration registers, they cannot exceed certain values: The latter consists of 14 bit, theoretically allowing for up to $2^{14} - 1 = 16383$ samples. The former consists of 6 bit, resulting in the effective maximum sampling duration of $(2^6 - 1) \cdot 8\,\mu\text{s} = 63 \cdot 8\,\mu\text{s} = 504\,\mu\text{s}$. Out of those, the fixed 4 µs guard period is not used for sampling. At the selected sampling rate of 8 MHz, 500 µs correspond to $8\,\text{MHz} \cdot 500\,\mu\text{s} = 4000$ samples, which is the actual maximum amount of samples that can be recorded on this SoC.

Sample format and resolution. For later processing of the data, it is more beneficial to record the samples as I/Q values, i.e., cartesian coordinates in the in-phase and quadrature plane (refer to Section 2.2.1 for more information).

Table 4.1 summarizes the parameters of interest for ORF together with their chosen values.

Table 4.1: Description of the I/Q sampling parameters available on the nRF52833 SoC, with the values chosen for ORF.

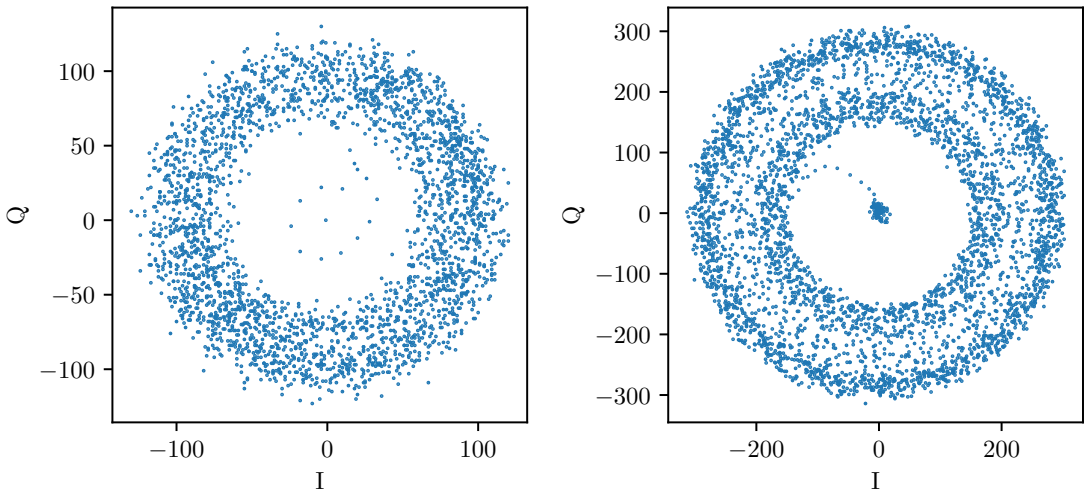
Parameter	Value	Description
`DFEMODE.DFEOPMODE`	`AoD`	no antenna switching at receiver
`CTEINLINECONF.CTEINLINECTRLEN`	`Disabled`	manual configuration of sampling
`DFECTRL1.SAMPLETYPE`	`IQ`	save samples in I/Q format
`DFECTRL1.DFEINEXTENSION`	`Payload`	I/Q sampling after `ADDRESS`

Parameter	Value	Description
`DFECTRL1.TSAMPLESPACING`	125 ns	sampling rate $\frac{1}{125\,\text{ns}} = 8\,\text{MHz}$
`DFECTRL1.TSAMPLESPACINGREF`	125 ns	also sample first 8 µs at 8 MHz
`DFECTRL1.NUMBEROF8US`	63	$63 \cdot 8\,\mu\text{s} = 504\,\mu\text{s}$
`DFECTRL2.TSAMPLEOFFSET`	0	no delay before sampling start
`DFEPACKET.MAXCNT`	≤ 4000	length of I/Q sample buffer

4.1.2 I/Q Sampling beyond BLE DFE

As mentioned in Section 3, the nRF52833 SoC supports more radio modes apart from BLE, namely 802.15.4-2006 and two proprietary protocols offering different data rates. All parameter names mentioned in the last part refer to the BLE terms DFE or CTE and it is not specified whether they have any effect when operating in a different radio mode.

During development of ORF, both the proprietary protocol operating at 2 Mbit/s and the 802.15.4 radio mode have been investigated by setting the `MODE` [21, Sec. 6.8.15.51] register accordingly. In both modes, it is possible to obtain raw I/Q samples during frame reception. Examples of I/Q samples obtained for a single transmission of random payload can be seen in Figure 4.1.



(a) nRF proprietary protocol at 2 Mbit/s

(b) IEEE 802.15.4 2450 MHz PHY protocol

Figure 4.1: I/Q plots of raw data obtained when sampling frames of different physical-layer protocols.

Out of these two options, 802.15.4 is selected as physical-layer protocol for ORF because it follows a known specification (which is not the case for the proprietary radio mode),

it is widely used in research and industry, and it uses a simple I/Q-based modulation with a single carrier. Additionally, previous research has shown the viability of radiometric fingerprinting based on 802.15.4-compliant signals [8]–[10]. However, with the adequate adaptions to the receiver pipeline and feature extraction, on-board radiometric fingerprinting is expected to be realizable with any of the other radio modes as well.

The manual I/Q sampling operation is not specified for any other radio mode than BLE. After showing that I/Q sampling is generally possible, two questions remain open:

- **Which part of the frame is captured in the I/Q samples?** The frame format differs for 802.15.4 mode operation from the BLE frame format. Notably, the addressing fields are defined as part of the MAC layer [21, Sec. 6.18.13.1] and are delayed by four octets compared to the address part of a BLE frame [21, Sec. 6.18.1]. The `ADDRESS` event, which is defined to trigger the I/Q sampling, is not mentioned in the receive sequence diagram for 802.15.4 mode, and instead replaced by the `FRAMESTART` event, which is triggered just after the PHR. On the other end, it is not specified until when the I/Q sampling goes on with (MAC layer) CRC enabled, only until the `PAYLOAD` event just before or until one of the events just after the CRC part of the packet (`END` and `CRCOK` are defined for 802.15.4 operation, while `PHYEND` is not).
- **Is the frame continuously sampled throughout this part?** As discussed in the previous section, sampling is in general performed continuously on the nRF hardware. However, for DFE operation, the I/Q sampling is defined to be interrupted just after the reference period and for the duration of the first switch slot. It is not specified whether this also applies while sampling over the packet payload and if so, what size is assumed for the switch slot.

Both questions are empirically investigated by several timer-based experiments and by inspecting the recorded samples: To that end, the programmable peripheral interconnect (PPI) [21, Sec. 6.15] and one of the timer/counter peripherals [21, Sec. 6.28] offered on the nRF52833 are used in conjunction to measure the time between different radio peripheral events.

A frame with 14 B PHY payload is transmitted in 802.15.4 mode at a data rate of 250 kbit/s, corresponding to a transmission pace of 32 µs/B (cf. Section 2.3). The PHY payload alone thus corresponds to 448 µs on-air transmission time, while the one-byte PHY header corresponds to additional 32 µs. Given the sampling rate of 8 MHz, the expected amount of samples can be deduced from this transmission time and is summarized in the table below for reference. The actual amount of recorded I/Q samples can be read from `DFEPACKET.AMOUNT` after each reception.

The timer is configured to count at 8MHz—matching the selected sampling rate—, from a given radio event E_{start} to another event E_{stop}. E_{start} is set to the `ADDRESS` or `FRAMESTART` event, while E_{stop} is tested to be one of `PAYLOAD`, `CRCOK`, `END` or `PHYEND`.

Table 4.2: Results of the timing experiments investigating the start and length of I/Q sampling with respect to different radio events, when using the IEEE 802.15.4 PHY radio mode. The counter interval matches the sampling interval of 125 ns.

Actual amount of I/Q samples	3799 ± 1
Counter expected, PHY payload	3584
Counter expected, PHY header + payload	3840
Counter, E_{start} = `FRAMESTART`, E_{stop} = `PAYLOAD`	3193 ± 1
Counter, E_{start} = `ADDRESS`, E_{stop} = `PAYLOAD`	3449 ± 1
Counter, E_{start} = `FRAMESTART`, E_{stop} = `END`	3583 ± 1
Counter, E_{start} = `ADDRESS`, E_{stop} = `END`	3840 ± 1
Counter, E_{start} = `ADDRESS`, E_{stop} = `PHYEND`	3840 ± 1
Counter, E_{start} = `ADDRESS`, E_{stop} = `CRCOK`	3840 ± 1

The counter values reported in Table 4.2 lead to the following conclusions which are also visualized in Figure 4.2:

- First of all, these counter-based experiments are valid, since the counter value between the events defined for 802.15.4 (`FRAMESTART` and `END`) matches the expected number for the PHY payload.
- The `ADDRESS` event (and thereby the I/Q sampling) is triggered at the start of the 1 B PHY header field, which corresponds to the 32 µs difference from the `FRAMESTART` event.
- The `PAYLOAD` event is triggered approximately 49 µs before the `END` event in 802.15.4 mode, which corresponds to roughly 1.5 B, a bit less than the 2 B 802.15.4 MAC CRC field.

The amount of I/Q samples obtained roughly matches the counter values from `ADDRESS` to one of `END`, `PHYEND` or `CRCOK`. This suggests that the I/Q sampling is started after the synchronization header and includes the PHY header as well as the entire PHY payload together with the CRC field. The actual amount of I/Q samples is still around 40 lower than expected, corresponding to 5 µs. Out of these, 4 µs correspond to the fixed guard period defined in BLE DFE.

The fact that one more microsecond worth of samples is missing suggests that a switch slot duration of 1 µs is assumed, and that a time equivalent to the first switch slot is skipped, even while sampling the payload. Figure 4.3a confirms that the in-phase and quadrature component of the sampled signal indeed show a discontinuity exactly after 64 samples, i.e., after 8 µs which correspond to the reference period duration. Since continuous samples are needed for the subsequent pipeline of ORF, the first 64 samples are simply discarded from further processing.

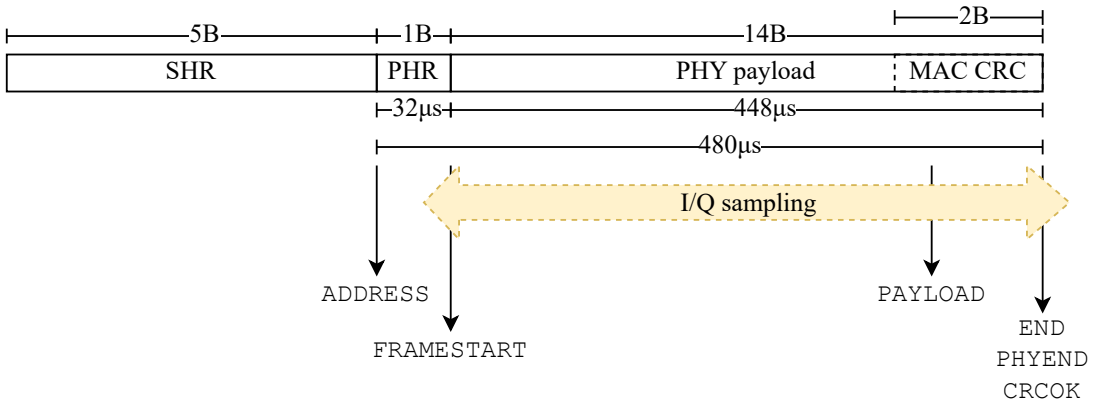
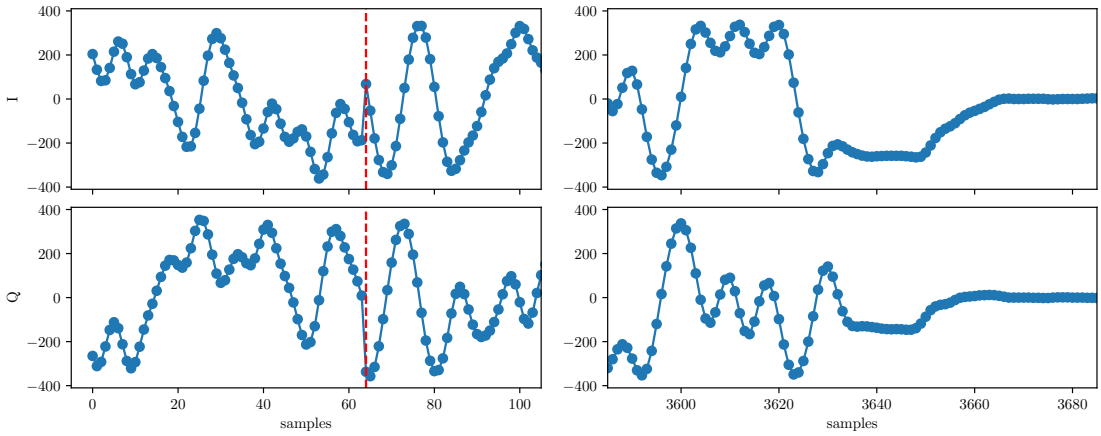


Figure 4.2: Estimated I/Q sampling duration with respect to the received IEEE 802.15.4 frame.



(a) Discontinuity after the reference period.

(b) Zero-samples at the end of the recorded frame.

Figure 4.3: In-phase (I) and quadrature (Q) component of start and end of a sampled IEEE 802.15.4 frame.

Additionally, a visual inspection of the end of the recorded signal in Figure 4.3b shows that the modulated signal stops after approximately 3620 to 3660 samples, depending on the transmitter device considered. The 140 to 180 missing samples correspond to 17.5 µs to 22.5 µs or less than 1 B of a recorded frame. However, this suggests that sampling does in fact not start exactly at the `ADDRESS` event, but at some point during the PHY header field. Final conclusions on exactly which part of the frame is recorded cannot be drawn at this point. The receiver synchronization implemented during stage 2 provides more insights on this question.

4.2 Stage 2: Feature Extraction

Stage 2 of the implementation of ORF entails processing the raw I/Q samples obtained in the previous stage, in order to extract transmitter-specific characteristics that can be used as features for the later classification stage. This section first presents the features selected for ORF and then provides a description of the implementation details to obtain those features on board.

4.2.1 ORF Features

Radiometric fingerprinting features are commonly classified in two groups: *Synchronization-based* features are obtained during receiver synchronization, while *constellation-based* features are defined on the constellation diagram obtained after successful receiver synchronization. Aligned with previous work, the carrier frequency offset (CFO) is the only synchronization-based feature considered for ORF. Most of the constellation-based features are inspired by previous work in [6], [8], [18], [23].

The following list gives a comprehensive overview of the features used for ORF, based on the hardware imperfection effects explained in Section 2.2.2. The implementation details are presented later in Section 4.2.9.

- ***Carrier Frequency Offset (CFO)*** is defined as the difference $f_{tx} - f_{rx}$ in the frequencies used for modulation (f_{tx}) and demodulation (f_{rx}). The carrier frequency tolerance defined for IEEE 802.15.4 is $\pm 40 ppm$ [19, Sec. 6.9.4], which for the 2450 MHz PHY protocol used in ORF translates to a frequency error per device of at most 99 200 Hz. Consequently, the combined carrier frequency offset is bound by $|CFO| \leq 2 \cdot 99\,200$ Hz.
- ***I/Q Offset (IQO)*** refers to the center of the I/Q plot, which may not coincide with the origin due to DC bias of the carrier waves.
- ***I/Q Skew (IQS)*** quantifies the I/Q phase imbalance as the difference in magnitude between the constellation points in neighboring quadrants of the I/Q plot.

- ***Constellation Cloud Shape (CCS)*** is a collection of features describing the shape of the I/Q sample clouds in terms of their maximum phase and amplitude spread.
- ***Error-Vector Magnitude (EVM)*** is defined in IEEE 802.15.4 as the root mean square average amplitude of the error vector $\vec{E}_k$ between I/Q samples and the corresponding optimal constellation points [19, Sec. 6.9.3].

4.2.2 Feature Extraction Overview

The features defined in the previous section either correspond to by-products or rely on the end-product of successful receiver synchronization, i.e., the compensation of discrepancies in amplitude, frequency, phase and time between the transmitter and the receiver (cf. Section 2.2.2). The I/Q samples that are available on this SoC are taken from the raw incoming signal (i.e., before the receiver synchronization). On the other hand, the decoded, bit-level payload after hardware-accelerated receiver synchronization discards the constellation diagram or further synchronization information. Therefore, the raw data has to be processed again in software to obtain the features.

ORF uses a typical coherent demodulation receiver pipeline architecture for OQPSK, mainly following [15] and using the Matlab Communications Toolbox implementation [24] as a reference. Receiver synchronization with the I/Q sample data available on the SoC is challenging because the synchronization header is not part of the recorded samples and the amount of available I/Q samples is rather small. Therefore, some steps of the receiver pipeline are adapted to account for the limited information and resources available on the SoC.

An overview of the synchronization process is given in Figure 4.4. The individual pipeline steps are described in detail in the remainder of this section, where S_k and S'_k refer to individual I/Q samples before and after each pipeline step, respectively. Figure 4.5 depicts the effects of the individual pipeline steps on the I/Q samples.

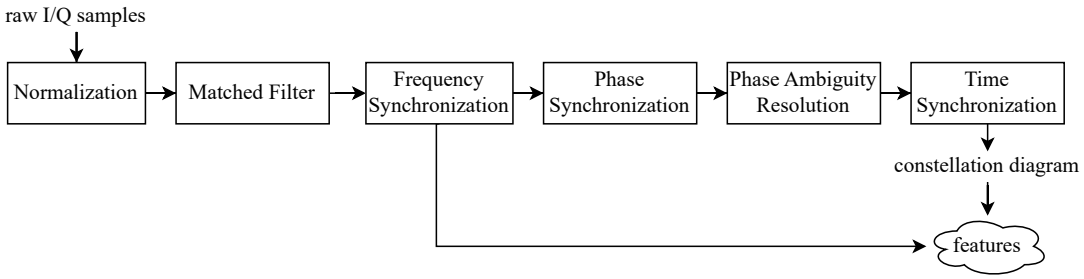
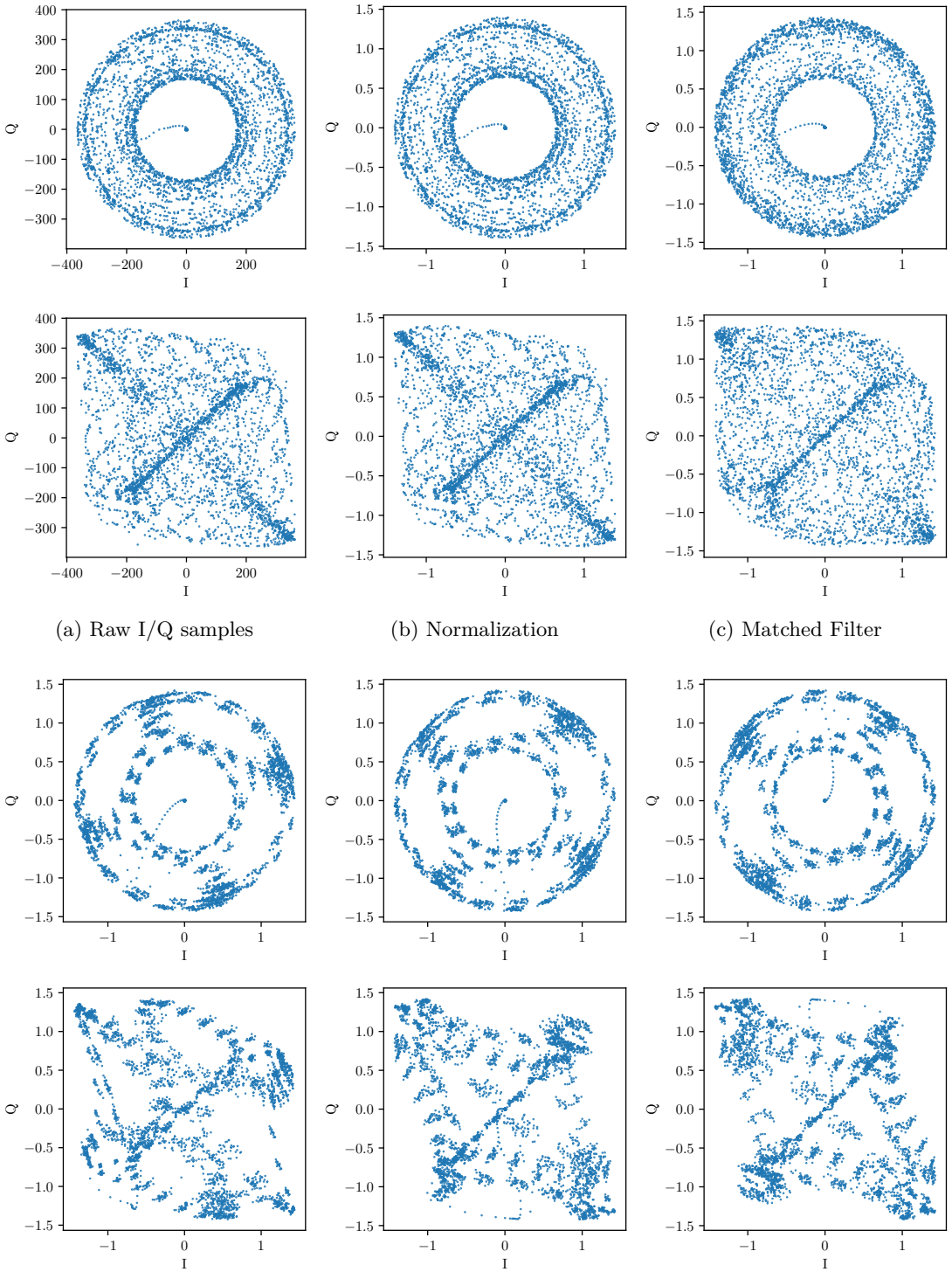


Figure 4.4: Feature extraction based on a typical coherent-receiver pipeline. The CFO feature is obtained from the frequency synchronization step, while all other features are obtained from the time-synchronized I/Q symbols.



(a) Raw I/Q samples (b) Normalization (c) Matched Filter

(d) Frequency Synchronization (e) Phase Synchronization (f) Phase Ambiguity Resolution

Figure 4.5: I/Q plots after different pipeline steps. For each step, the top plot shows the original OQPSK-modulated signal, while the bottom plot corrects for the half-symbol offset, effectively resulting in a QPSK constellation.

A dual implementation approach is employed during development of ORF: Firstly, an off-board version is implemented in Python using `numpy` [25] for fast development and evaluation cycles. Secondly, the on-board code is written in C and makes extensive use of the `CMSIS-DSP` Software Library which is part of the ARM® Cortex® Microcontroller Software Interface Standard (CMSIS) and provides implementations of many common compute processing functions, optimized for Cortex-M processor based devices such as the SoC used for ORF [26]. The same algorithms are used in Python and C and Section 5.2 will show empirically that they produce virtually identical results.

4.2.3 Normalization

A typical receiver pipeline starts with an Automatic Gain Control (AGC) block, which dynamically adjusts the signal amplitude to a desired reference level [15, Ch. 9.5]. This is necessary in general, since the signal amplitude changes over time due to multipath, interference and noise effects.

It is not specified in the official documentation where exactly in the receiver pipeline the I/Q samples are taken. However, it has been confirmed by Nordic Semiconductors[2] that the incoming signal is sampled "at baseband after receive filters", and that "AGC is enabled during the payload section (i.e. one must expect that the gain may change during the payload section)".

ORF can therefore expect amplitude changes to be already compensated for in the obtained I/Q samples. However, in order to facilitate amplitude comparison between frames that are received with differing SNR values (and therefore different mean amplitudes), the I/Q samples are in a first step normalized to have mean amplitude one by $S'_k = \frac{S_k}{|S|}$. Therefore, Figure 4.5b is barely a scaled version of Figure 4.5a.

As part of this pipeline step, the I/Q samples are also converted from `int16_t` as given by the nRF radio peripheral to the IEEE 754 `float32_t` datatype for further processing. Both operations can be implemented in place in a straightforward way and thus require no extra memory.

4.2.4 Matched Filter

A filter is commonly used at the receiver which correlates the incoming in-phase and quadrature samples (separately) with the known pulse shape $p(t)$ used during modulation at the transmitter (in the case of IEEE 802.15.4 a half-sine pulse as defined in Equation 2.5). This so-called matched filter achieves maximal signal-to-noise ratio (SNR) in the presence of additive white noise [27], thereby improving the input data quality for the receiver pipeline and consequently the synchronization quality. Correlation corresponds to convolution of the conjugated time-reversed pulse shape $p'(t) = p^*(-t)$. In discrete

[2] https://devzone.nordicsemi.com/f/nordic-q-a/99749/nrf52833---collecting-i-q-samples/426505

time, the output of the matched filter is thus calculated individually for the in-phase and quadrature components by $S' = p' * S$, i.e. $S'_k = \sum_{i=0}^{SPS} p'_i \cdot S_{k-i}$.

When implemented in a naive way, convolution cannot be performed in place because of the data dependency of later samples on already-processed previous ones. However, since the length of p' (excluding zero-samples) matches the amount of recorded samples per I/Q symbol (SPS), which is very small compared to the length of S, it is possible to compute each S'_k considering a copy of $S_k, \ldots, S_{k-SPS}$. The buffer holding those copies needs to be of length SPS and can be safely allocated on the stack.

4.2.5 Frequency Synchronization

The goal of the frequency synchronization step is to estimate and correct the carrier frequency offset CFO between the oscillators in the transmitter and the receiver. As detailed in the following, ORF implements the frequency offset detection described in [28], combined with the frequency interpolation technique from [29] to achieve accurate frequency estimation.

IEEE 802.15.4 2450 MHz PHY uses OQPSK with an I/Q symbol rate of 1 MHz per signal component. The spectrum of such a signal obtained by a Fourier Transform is spread between -1 MHz and 1 MHz as shown in Figure 4.6, making direct carrier frequency estimation very difficult. To overcome that problem, the information-bearing OQPSK signal S is squared (doubling the frequencies), which results in a single-tone signal on the in-phase component of S^2 due to the nature of OQPSK modulation.

If $CFO = 0$, the predominating frequency components of S^2 are ± 1 MHz. However, if $CFO \neq 0$, the peaks in the spectrum are equally shifted to $\pm 1\,\text{MHz} + 2 \cdot CFO$. Estimating CFO thus boils down to finding the peaks around ± 1 MHz in the spectrum of S^2. As discussed before in Section 4.2.1, $|CFO| < 200$ kHz according to the standard. That means it is sufficient to search for spectral peaks $2 \cdot 400$ kHz around ± 1 MHz, respectively. The bottom plot of Figure 4.6 shows the spectrum of S^2, visualizing the search regions, the recognized peaks and the estimated carrier offset.

The Discrete Fourier Transform (DFT) used in practical systems such as ORF operates on time-discrete signals, like S, and produces a discrete spectrum, which is calculated at frequencies that are multiples of the *frequency resolution*, or *frequency bin* size, $\Delta f = \frac{f_s}{N_{DFT}}$. f_s denotes the sampling frequency of the time-discrete signal and N_{DFT} the amount of samples considered for the DFT. That means, all frequency components between two integer multiples of Δf are distributed over the surrounding two frequency bins. With $N = 3800$ available samples and the sampling frequency of $f_s = 8$ MHz, this results in $\Delta f \approx 1053$ Hz on the original frequency scale of S.

Looking at the definition of Δf above, it is apparent that there are two possibilities to improve the frequency resolution: Either by decreasing the sampling rate f_s or by increasing the amount of samples N_{DFT}. For the case of ORF with a limited maximum

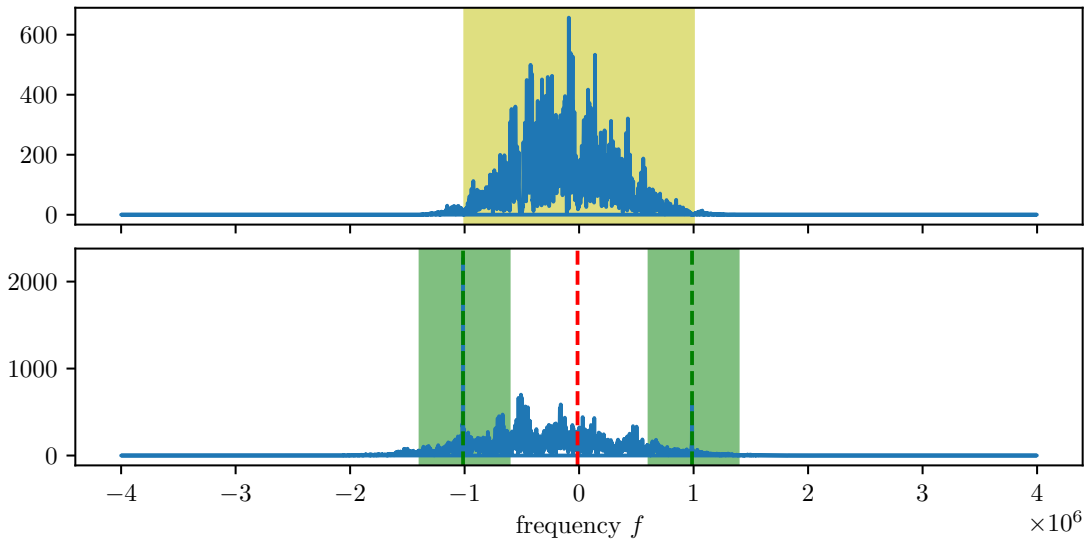


Figure 4.6: Absolute frequency spectrum of S and S^2 obtained using 4096-FFT.

absolute sampling duration, the former would effectively also decrease N_{DFT} (thereby leaving Δf unchanged) and the latter can only be achieved with artificially added zeros. This so-called *zero padding* corresponds to an interpolation in the frequency domain resulting in a better estimate of the maximum of single peaks, but no better distinction between peaks close to each other.

Zero padding is used in ORF mainly in order to let N_{DFT} be a power of two, which is a requirement for Fast Fourier Transform (FFT) algorithms. Considering the amount of actual samples $N = 3800$, memory constraints and the fact that `CMSIS-DSP` provides optimized FFT implementations for signals of up to 4096 samples, $N_{DFT} = 4096$ is chosen for ORF. Since the spectrum is known to contain two single frequency peaks far apart from each other, ORF can slightly benefit from the zero-padding interpolation, giving $\Delta f \approx 977\,\mathrm{Hz}$.

To further improve the frequency resolution, frequency interpolation is performed as follows: First, the time signal S^2 is windowed (i.e., multiplied) with a Gaussian function, which leads to spectral peaks very close to a Gaussian form (the Fourier transform of a Gaussian function is another Gaussian function). Second, the spectral peak is interpolated using Gaussian interpolation (i.e., fitting a Gaussian function to the peak). In this combination, [29] shows that the frequency estimation error on idealized spectra (without noise or interference) is 0.0087 % of the bin size. For ORF, this gives a theoretical frequency resolution of $\Delta f < 0.1\,\mathrm{Hz}$.

After identification and interpolation of the peaks both in the left (f_{left}) and the right (f_{right}) side of the spectrum, the carrier frequency offset (CFO) is calculated as $CFO =$

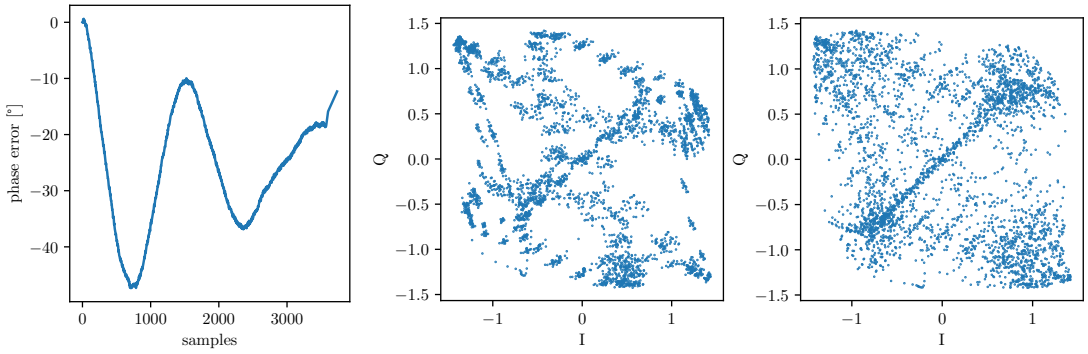
$\frac{f_{\text{left}}+f_{\text{right}}}{4}$. The CFO is kept as one feature and the I/Q samples are jointly frequency-corrected by $S'_k = S_k \cdot e^{-i \cdot 2\pi \cdot CFO \cdot \frac{k}{f_s}}$.

The FFT function provided by `CMSIS-DSP` operates in place, but since the original I/Q samples are still needed after the signal squaring, windowing and Fourier transformation, an additional memory buffer holding 4096 complex values ($4096 \cdot 8\,\text{B} = 32.768\,\text{kB}$) is needed for those operations. In terms of constant data, the 3800 samples of a real-valued Gaussian window ($3800 \cdot 4\,\text{B} = 15.2\,\text{kB}$) need to be stored in ROM.

4.2.6 Phase Synchronization

Although the frequency synchronization step compensates for the carrier frequency offset between transmitter and receiver, the carrier and thereby the I/Q samples at the receiver can still be *out of phase* compared to the transmitter. This effect manifests itself in a rotation of the constellation diagram around the origin [15, Sec. 5.3].

Traditionally, a phase lock loop (PLL) is used to dynamically track the phase of S and to compensate each sample by the current phase error estimate during processing. Since frequency corresponds to a constant phase change over time, the PLL is also able to correct the potential residual frequency offset in the samples. Being a typical control system feedback loop, a nonzero period of time is required to achieve phase lock, i.e., to reduce the estimation error to zero. Approximate expressions for the acquisition time and the tracking error are given in [15, Secs. C.1.5–C.1.6], showing that shorter acquisition time implies larger tracking error, and vice-versa.



(a) Phase error estimate over time.

(b) Offset-compensated I/Q plot before and after the PLL.

Figure 4.7: Effect of a PLL-based phase synchronization approach.

Practical experiments with samples recorded on the SoC show that the acquisition time is in the order of 2500 samples for a reasonably small tracking error. The first two thirds of the samples are thus phase-corrected with wrong estimates, effectively decreasing the

data quality. Figure 4.7 shows the effect of using a PLL for phase synchronization in terms of the phase error estimate over time and the resulting I/Q plot.

To achieve better performance, a different, iterative approach for phase synchronization is employed in ORF: In each step i, the I/Q samples are jointly rotated by an angle $i \cdot \Delta\phi$ and the resulting I/Q plot shapes are compared to the expected one with zero phase offset. The one that matches best is taken as the phase offset estimate $\hat{\phi}$. Since OQPSK has an inherent phase-ambiguity of 90° (which will be accounted for in the next pipeline step), the angle step size is defined as $\Delta\phi = \frac{\pi}{2 \cdot M}$, where $M = 16$ is the selected number of steps for ORF. This approach assumes that the phase offset is constant for the whole frame, i.e. that the remaining frequency error is near zero. Due to the high frequency resolution achieved using interpolation and the short duration of the frame, this assumption holds for ORF.

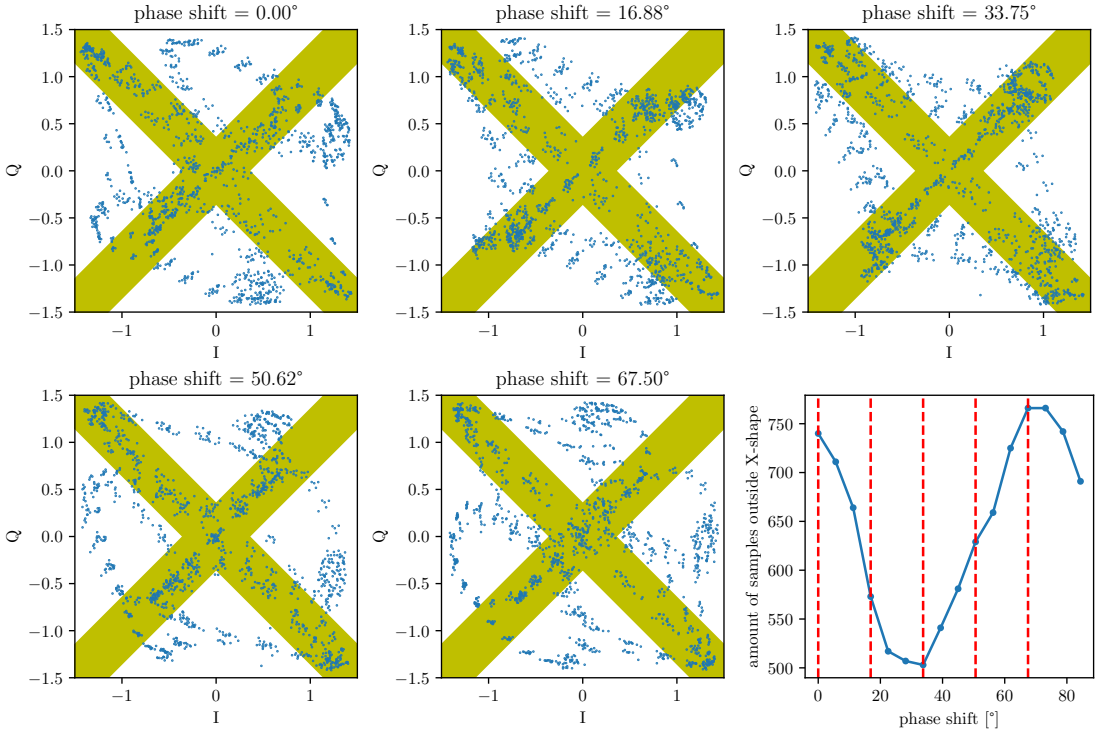


Figure 4.8: Finding the phase offset by comparison to the expected X-shape. I/Q plots for a subset of the 16 phase shift values tried and the amount of samples outside the shape as a function of the phase shift values.

When (virtually) compensating the offset between the in-phase and quadrature components of an OQPSK signal, the expected characteristic I/Q plot shape is identical to the QPSK one, which resembles an X connecting the four ideal constellation points (cf. Section 2.2.1). ORF tests for this shape by counting the amount of samples outside of the shape with a given margin. In particular, all samples S_k with $||S_{k,I}| - |S_{k+SPS/2,Q}|| > 0.35$

are treated as outliers. Figure 4.8 shows I/Q plots for different rotation angles of the example frame, together with a diagram showing the amount of outliers for different rotation angles. $\hat{\phi}$ is taken as the minimum of that function, and the original I/Q samples are jointly rotated with $S'_k = S_k \cdot e^{i\hat{\phi}}$.

In theory, the phase synchronization operation can be implemented in place since it involves no non-linearities. Nevertheless, in practice, to avoid rounding errors due to the use of trigonometric functions (which are implemented using table lookup and interpolation), the test rotations are only applied to copies of the original samples. It is however sufficient to use a smaller amount N_{PS} of the N I/Q samples to recognize the shape, which results in the need for a memory buffer of N_{PS} complex values. Since $N_{PS} < N_{DFT}$ and the spectral information from the previous step is no longer needed, the same memory buffer can be reused for this step. $N_{PS} = 1500$ is chosen for ORF, which translates to 12 kB in terms of memory.

4.2.7 Phase Ambiguity Resolution

As mentioned in the last section, OQPSK modulation has an inherent phase ambiguity of 90°. To resolve this ambiguity, a known sequence of I/Q symbols is correlated with the samples twice, before and after rotation by 90°. The known I/Q symbols are usually taken from the standardized first part of a frame. For IEEE 802.15.4, the synchronization header (SHR) contains 5 fixed bytes which map to a fixed I/Q symbol (chip) sequence. The SHR is not part of the recorded samples (cf. Section 4.1.2), but the decoded payload is available on the SoC. ORF can thus use part of the payload to generate the expected I/Q symbol sequence. In particular, the four most significant bits of the first payload byte are picked to obtain a sequence of 16 I/Q symbols for correlation. Their transmission is started 48 µs after the end of the synchronization header.

Instead of calculating the correlation over the whole signal, it is sufficient to consider the correlation values corresponding to the sample number at which the I/Q symbol sequence is expected to be found. Analysis of recorded data in Figure 4.9 shows that the I/Q symbol sequence corresponding to the four most significant bits of the first payload byte always starts between 118 and 120 samples (with very few outliers at 117 or 121 samples) after the recording started. Therefore, the correlation value is only calculated for time offsets between 115 and 124 samples in the ORF implementation.

The actual sampling start on the SoC can be deduced from this average timing offset of 119 samples, thereby answering the remaining open question from Section 4.1.2: 119 samples correspond to 14.875 µs at the selected sampling frequency of 8 MHz. This means that the continuous sampling (after discarding the first 64 samples from the reference period) starts 48 µs − 14.875 µs = 33.125 µs after the start of the physical header as depicted in Figure 4.10.

Since IEEE 802.15.4 uses pseudo-random I/Q symbol sequences, the correct phase and time offset will be clearly detectable as having the largest absolute correlation value. A

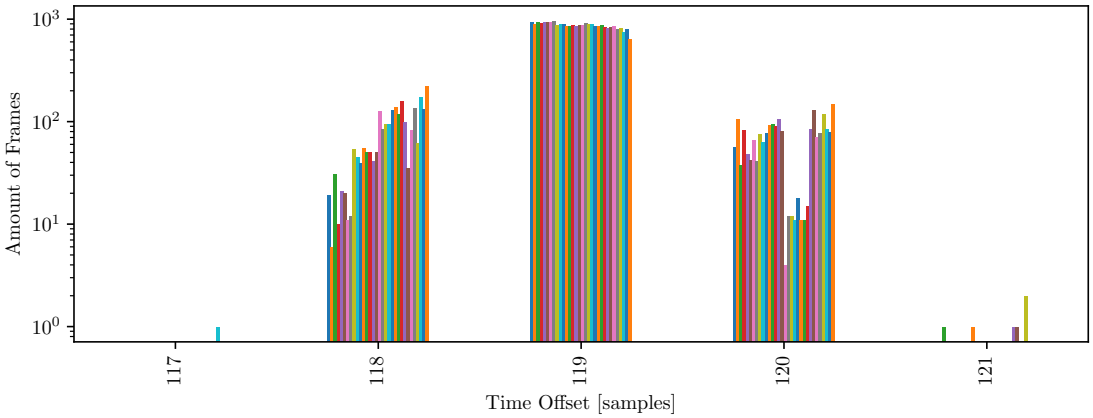


Figure 4.9: Start of the four most significant bits in the first payload byte. Different colors correspond to different transmitter devices.

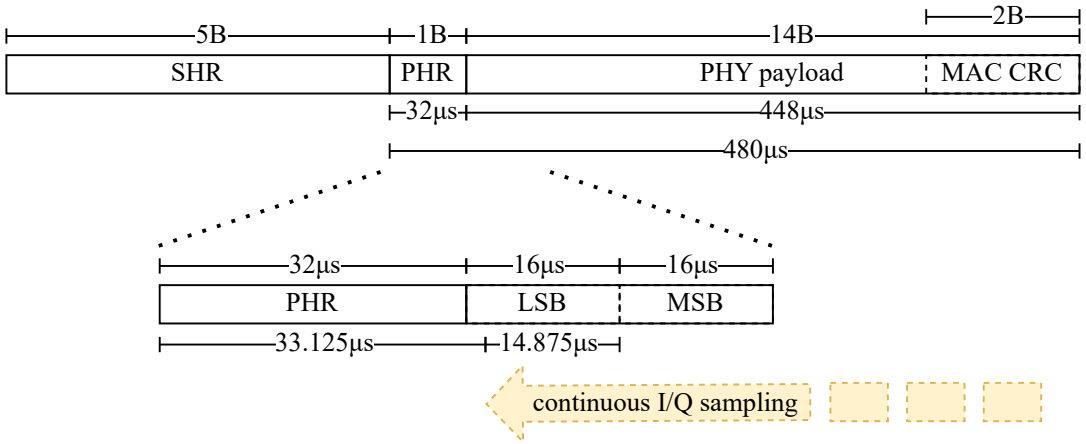


Figure 4.10: Exact start of continuous I/Q samples with respect to the received IEEE 802.15.4 frame.

large negative correlation corresponds to an additional phase shift of 180°. Just as before, the I/Q samples are jointly rotated by the detected phase offset ϕ_{amb} with $S'_k = S_k \cdot e^{i\phi_{\text{amb}}}$. Figure 4.11 shows the start of the sampled signal components together with the expected chip sequence for the detected phase and time offset.

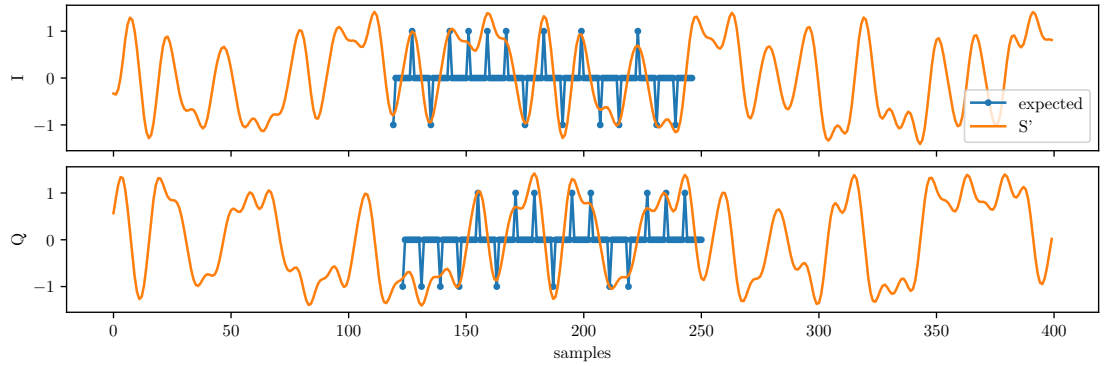


Figure 4.11: Phase ambiguity resolution using correlation. The plot shows the start of the phase-corrected signal S' together with the expected I/Q symbol sequence.

In terms of memory consumption, the phase ambiguity resolution is again implemented on a copy of the I/Q samples, but this time only $N_{PAR} = 10 + 16.5 \cdot SPS$ samples are needed (resulting in 1136 B). This is way smaller than N_{DFT}, too, allowing to re-use the same memory buffer again. In ROM, the complete chip map of IEEE 802.15.4 needs to be stored. Using a naive implementation, this requires $16 \cdot 32 = 512$ B of ROM, but a more memory-efficient implementation using bitarrays could bring this number down to $16 \cdot 4 = 64$ B.

4.2.8 Time Synchronization

After the received samples are corrected in terms of amplitude, frequency and phase, the goal of this last step in the receiver pipeline is to find the optimal sampling instant for each transmitted symbol. This corresponds to the point in time with the largest amplitude per signal component. ORF uses an oversampling rate of eight samples per I/Q symbol, which as a baseline gives eight samples to choose from. However, none of these eight samples might capture the actual maximum amplitude per symbol, resulting in the need of signal interpolation.

Time synchronization in ORF is realized as a PLL, following [15, Secs. 8.4, 8.6] using the (decision-directed) zero-crossing method, a piecewise parabolic interpolator with Farrow structure and coefficient $\alpha = 0.5$, a modulo-1 counter interpolation control and a proportional-plus integrator (PI) loop filter. This setup proved to work well in practice for ORF and is thus not adapted further. Figures 4.12, 4.13 visualize the effect of the time synchronization step on the example frame used for Figure 4.5 in terms of an I/Q plot and time-plots of the two signal components.

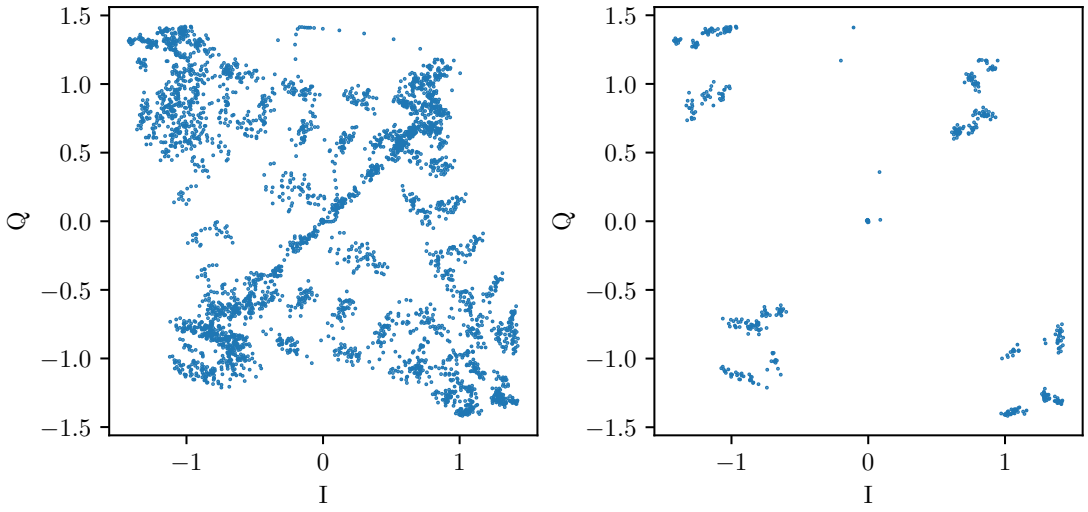


Figure 4.12: I/Q plot before and after the time synchronization step.

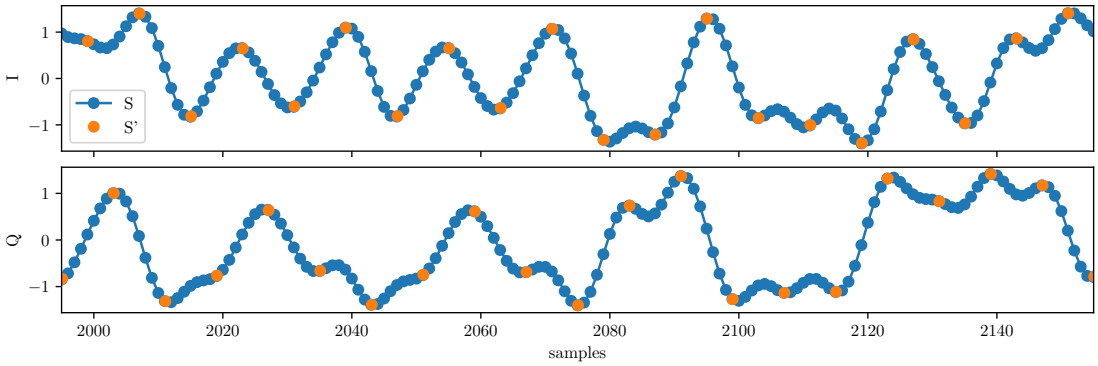


Figure 4.13: Part of the original signal S together with the corresponding time-synchronized and interpolated I/Q symbols S'.

The time-synchronization PLL implementation requires very little additional memory as the I/Q samples are processed in-order one after another. The state information needed for the interpolation and PI loop filters can by stored in less than 110 B and thus can be safely allocated on the stack.

4.2.9 Extraction of Constellation-based Features

The I/Q plot obtained from the last receiver pipeline step resembles a standard QPSK constellation diagram, with I/Q sample clouds instead of the theoretical four exact constellation points. The constellation-based features defined in Section 4.2.1 are extracted from the I/Q plot as detailed in the following.

First, the set of I/Q symbols S_k is split into four subsets $\mathcal{P}_i,\ i \in \{1, 2, 3, 4\}$ according to the quadrant they fall into, attributing them to one of the four ideal constellation points P_i. The center point $\hat{P}_i$ of each constellation cloud $\mathcal{P}_i$ is estimated as the mean of all samples belonging to that cloud:

$$\hat{P}_i = \frac{1}{|\mathcal{P}_i|} \cdot \sum_{S_k \in \mathcal{P}_i} S_k$$

The I/Q offset IQO is calculated as the mean of the constellation cloud centers $\hat{P}_i$:

$$IQO = \frac{1}{4} \cdot \sum_{i=1}^{4} \hat{P}_i$$

Since the I/Q phase imbalance has the effect of "skewing" the constellation diagram in diagonal direction, the I/Q skew feature IQS is quantified by the difference in amplitude between the two diagonals:

$$IQS = \frac{|\hat{P}_2| + |\hat{P}_4|}{2} - \frac{|\hat{P}_1| + |\hat{P}_3|}{2}$$

The shape of the constellation clouds is captured by their extend in two directions, the magnitude and the phase. Since the clouds in opposite quadrants are experimentally found to have similar shape, they are averaged together. Let

$$\begin{aligned} \text{mag}(\mathcal{P}_i) &= \{|S_k| : S_k \in \mathcal{P}_i\} \\ \text{phase}(\mathcal{P}_i) &= \{\angle S_k : S_k \in \mathcal{P}_i\} \\ \text{diff}(\mathcal{P}) &= \max \mathcal{P} - \min \mathcal{P} \end{aligned}$$

then the four CCS features are defined as:

$$CCS_{M13} = \frac{1}{2}\left(\text{diff}(\text{mag}(\mathcal{P}_1)) + \text{diff}(\text{mag}(\mathcal{P}_3))\right)$$
$$CCS_{M24} = \frac{1}{2}\left(\text{diff}(\text{mag}(\mathcal{P}_2)) + \text{diff}(\text{mag}(\mathcal{P}_4))\right)$$
$$CCS_{P13} = \frac{1}{2}\left(\text{diff}(\text{phase}(\mathcal{P}_1)) + \text{diff}(\text{phase}(\mathcal{P}_3))\right)$$
$$CCS_{P24} = \frac{1}{2}\left(\text{diff}(\text{phase}(\mathcal{P}_2)) + \text{diff}(\text{phase}(\mathcal{P}_4))\right)$$

The Error-Vector Magnitude is defined in IEEE 802.15.4 as the root mean square amplitude of the error vectors of the individual symbols. To abstract away from the I/Q offset and I/Q phase imbalance which are already captured in separate features, the error vector $\vec{E}_k = S_k - \hat{P}_i$ is calculated with respect to the constellation cloud center $\hat{P}_i$ instead of the ideal constellation point P_i, and the average magnitude of the constellation cloud centers $\hat{P}_i$.

$$EVM = \sqrt{\frac{\frac{1}{N}\sum_{k=1}^{N}|E_k|^2}{\frac{1}{4}\sum_{i=1}^{4}|\hat{P}_i|}}$$

Figure 4.14 visualizes the constellation-based features on the constellation diagram obtained for the frame used as an example throughout this section.

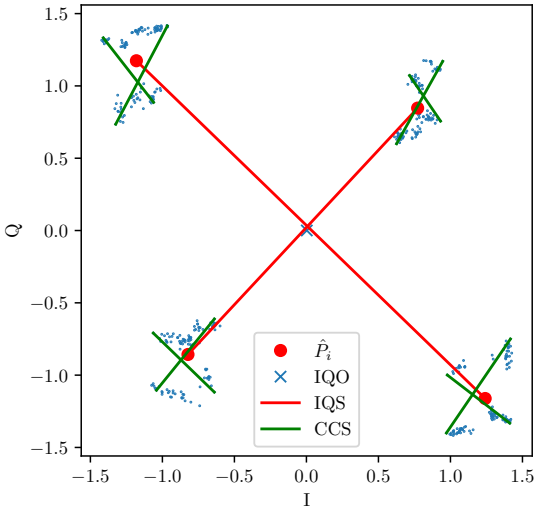


Figure 4.14: I/Q plot of time-synchronized samples with visualizations of all constellation-based features.

4.3 Stage 3: Fingerprint Classifier

As for any radiometric fingerprinting system, the third and last stage of ORF corresponds to classifying the extracted features in order to identify the transmitter devices. Most commonly, this part is realized using an ML model suitable for classification (cf. Section 2.1). It is not a primary goal of this work to optimize the classifier performance, but rather to show the general feasibility of on-board classification. Therefore, only a single ML classification model is considered for ORF: A Random Forest classifier. It is an ensemble learning method which combines the output of several decision trees, which are independently trained on a random subset of the features and the training data, a process referred to as *bootstrap aggregating*, or *bagging* for short. In that way, they avoid *overfitting* to the training set and generally outperform simple decision trees. [30]

For all experiments, the input of the classifier is preprocessed in two steps: First, outliers are removed for each feature and transmitter device individually, where samples with absolute *z-scores* (which are defined as the difference to the feature mean value in multiples of the feature standard deviation) above three are considered to be outliers. In a second step, *min-max feature scaling* is used to bring all values per feature into the range $[0, 1]$.

The Random Forest classifier is trained in Python using `scikit-learn` [31] and converted to C99 code using `emlearn` [32]. The limited ROM space available on the SoC restricts the model complexity, which depends on a variety of model hyperparameters such as the number of decision trees that form the Random Forest. Lower model complexity means less code, but also potentially a loss in performance. Section 5.4 therefore jointly investigates the classifier performance in terms of accuracy together with the model size for different combinations of hyperparameters, finally leading to a decision for the hyperparameters used in ORF.

5 Evaluation

The performance of the complete ORF system implementation can be quantified using its classification outcome, i.e., the ratio of frames that are attributed to the correct transmitter. This final metric however depends on a multitude of factors, which are discussed one after another in the following sections:

- ***Evaluation Dataset.*** The dataset used for evaluation purposes has a significant impact on the evaluation outcome. First of all, the probability that two different devices have a similar fingerprint due to similar hardware characteristics, effectively hindering distinction between those devices, increases according to the number of considered transmitter devices. Especially devices of the same type are more likely to expose similar characteristics due to their common hardware design. In turn, it follows that increasing the device type variety within the dataset can be expected to improve the evaluation outcome. Apart from that, signal distortions introduced by the wireless transmission channel can severely impair correct classification [8]. The experimental setup realized to obtain the data used for evaluation purposes is explained in detail in Section 5.1.
- ***Raw Data Quality.*** Previous work in radiometric fingerprinting relies on expensive and specialized hardware for I/Q sampling (cf. Section 2.1). In contrast, ORF obtains I/Q samples directly on a COTS device. Since it is difficult to assess the I/Q sample quality directly due to the automatically applied gain control, the data quality is only evaluated indirectly using the overall system performance.
- ***Implementation Details.*** The concrete implementation of each step naturally influences the performance of the following steps in the fingerprint-extraction and classification pipeline. Bad performance of an early step in the pipeline is thus magnified in the final outcome. On the other hand, it is hard and time-consuming to evaluate the impact of a specific step on the final classification when working with real hardware. In order to allow for faster evaluation cycles without resorting to the hardware device, the Python implementation has been used throughout the development. Section 5.2 shows that there is no significant difference between the ORF pipeline implementations in Python and C.
- ***Feature Engineering.*** The system performance also largely depends on the selected features. If the features fail to capture the individual transmitter hardware characteristics well, the classification deteriorates quickly. An intuitive measure of feature quality and significance is their variation among and between classes:

A "good" feature has little variation between frames of a single transmitter, but differs significantly from transmitter to transmitter. Section 5.3 investigates the quality of the features defined for ORF.

- ***Classifier Training Hyperparameters.*** Same as for all machine learning models, the performance of the Random Forest classifier used in ORF highly depends on so-called *hyperparameters*. This includes model parameters such as the number of subtrees and training parameters such as the size of the training set. On the other hand, those parameters also influence the resulting classifier model size, which is another critical factor for on-board deployment. Section 5.4 thus presents a joint evaluation of the classifier performance and ROM memory usage.

Although not directly connected to the classification accuracy, another important factor for deployment on embedded systems is investigated in Section 5.5: the resource consumption of the ORF implementation in terms of memory, time and energy. Section 5.6 concludes the evaluation with a brief look into the stability of the features over time.

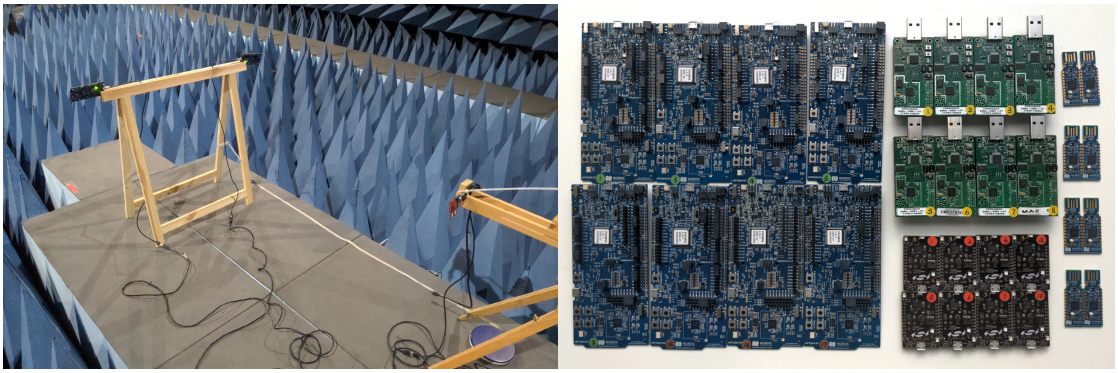
5.1 Experimental Setup

The data used to train and evaluate the ORF classifier is collected under ideal, fixed conditions in an anechoic chamber with direct line-of-sight transmission in order to avoid multipath, interference, Doppler and noise effects. It has been shown in [8] that it is possible to adapt the classifier to real-world scenarios by training it on an artificially augmented dataset simulating those wireless channel effects based on the ideal samples.

32 different devices of four different types are used to transmit 802.15.4 PHY frames depicted in Figure 4.2 with 12B of random payload, followed by 2B of CRC, at a rate of 50 frames/s. One nRF52833-DK board is placed at a distance of 1.3m and used as a receiver, capturing the raw I/Q samples of 1000 frames per transmitter at a sampling rate of 8 MHz. Table 5.1 gives an overview of the used devices.

Table 5.1: Devices used as receiver (Rx) and transmitter (Tx) during data collection, together with their respective tag used throughout the evaluation. The age of the devices can be estimated from their approximate year of purchase.

Device Type	Tx	Rx	Amount	Tag	Year (approx.)
nRF52833 DK [21], [33]		X	1		2023
nRF52840 DK [34], [35]	X		8	`dk*`	2020
nRF52840 Dongle [35], [36]	X		8	`dongle*`	2020
Thunderboard Sense 2 [37]	X		8	`gecko*`	2019
Tmote Sky [38]	X		8	`sky*`	2009



(a) Each transmitter (right) is placed 1.3m apart from the receiver (left) in an anechoic chamber.

(b) Transmitter devices included in the dataset.

Figure 5.1: Experimental setup for the data collection.

5.2 Off-Board and On-Board Implementation

As mentioned previously in Section 4.2.2, two implementations of the whole coherent-receiver, feature-extraction and classifier pipeline have been developed for ORF: A Python version for fast off-board evaluation on an ordinary x86-64 Linux PC, as well as a C version for on-board deployment on the SoC. Both versions implement the same algorithms and use single-precision floating point storage and computation compliant with the IEEE 754 standard[1].

However, the trigonometric functions used in several steps of the pipeline are implemented differently on the two systems: On one side, CPython and thereby `numpy` uses the highly-optimized, native C math library for those functions. In contrast, the on-board C implementation uses the functions provided by `CMSIS-DSP`, which are implemented as a combination of table lookup using 512 values and linear interpolation [26]. The precision of the outcome is thus expected to be lower on the SoC than off-board.

This section shows that the overall results obtained using the Python implementation are nevertheless very similar to the ones obtained on-board, justifying the usage of Python for evaluation purposes. To that end, the result of individual pipeline steps are first compared for a single frame. Then, the outcome of the whole pipeline is compared over the whole dataset.

In any case, the results reported in the remainder of the evaluation are obtained from the features extracted using the on-board C pipeline.

[1] using `np.float32` in Python and `float32_t`/`float` in C

5.2.1 Individual Pipeline Steps

For this part, a single frame out of the dataset is processed both off board and on board. The individual sample values after each pipeline step are compared and a relative error value is calculated as

$$\epsilon_k = \left| \frac{S_k^{\text{off-board}} - S_k^{\text{on-board}}}{S_k^{\text{off-board}}} \right|.$$

Table 5.2: Average error of I/Q samples obtained via on-board C processing of a single example frame compared to the off-board Python pipeline.

	$\bar{\epsilon_k}$ [%]
Normalization	0.0001
Matched Filter	0.0001
Frequency Synchronization	0.2496
Phase Synchronization	0.2496
Phase Ambiguity Resolution	0.2496
Time Synchronization	0.1573

Table 5.2 shows the average error $\bar{\epsilon_k}$ after each pipeline step. Overall, the average difference between samples obtained using the off-board and on-board pipeline is very low. The largest difference is introduced at the frequency synchronization step, because even a slightly different carrier frequency offset estimate leads to an increasing difference in frequency-corrected sample values over the length of the frame. The remaining pipeline steps introduce no further difference in samples, and the overall error is smaller after the time synchronization step because the number of samples is lower.

5.2.2 Whole Feature Extraction Pipeline

Instead of looking at the I/Q sample values after individual pipeline steps for a single frame, the whole dataset is processed both off board and on board, up to and including the feature extraction step. For each frame and feature f, the difference is quantified as

$$\epsilon_f = \left| \frac{f^{\text{off-board}} - f^{\text{on-board}}}{f^{\text{off-board}}} \right|.$$

Table 5.3: Average error of features obtained via on-board C processing of the whole dataset compared to the off-board Python pipeline.

Feature	$\bar{\epsilon_f}$ [%]
Carrier Frequency Offset CFO	0.0006
I/Q offset IQO-I	9.7894
I/Q offset IQO-Q	25.0879
I/Q skew IQS	0.4447
Constellation Cloud Shape CCS_{M13}	0.6764
Constellation Cloud Shape CCS_{M24}	0.6500
Constellation Cloud Shape CCS_{P13}	0.6287
Constellation Cloud Shape CCS_{P24}	0.8943
Error-Vector Magnitude EVM	0.8805

Table 5.3 shows ϵ_f for each feature, averaged over all frames available in the dataset. As expected based on the previous experiment which showed that the time-synchronized samples only differ marginally between the on-board and off-board pipeline implementation, most of the constellation-based features differ in less than 1 %. Only the IQO features present higher differences, which is mainly a mathematical effect of them having values very close to zero for most transmitters (cf. Figure 5.2 in the next section). The relative error value is thereby amplified but not of importance in practice.

5.3 Feature Quality

A decisive factor for the performance of any classifier is the selection of features used as input. This part investigates the quality of the features defined for ORF in Section 4.2.1 using the dataset described in Section 5.1.

The process of selecting optimal features is considered to be quite hard in general. A multitude of metrics has been proposed for statistical and empirical evaluation of feature quality and importance [39], [40]. However, since it is not the main goal of ORF to optimize for the best possible classifier performance, a relatively straightforward approach to feature selection is used, mainly based on visual and statistical assessment of feature variance.

"Good" features have small variance among data of a single class, but significant variance between different classes. A mathematical method to capture this property is the *one-way analysis of variance (one-way ANOVA)*, which performs a statistical test whether two or more class means are equal [41]. The resulting *f-value* is higher for larger differences in class means.

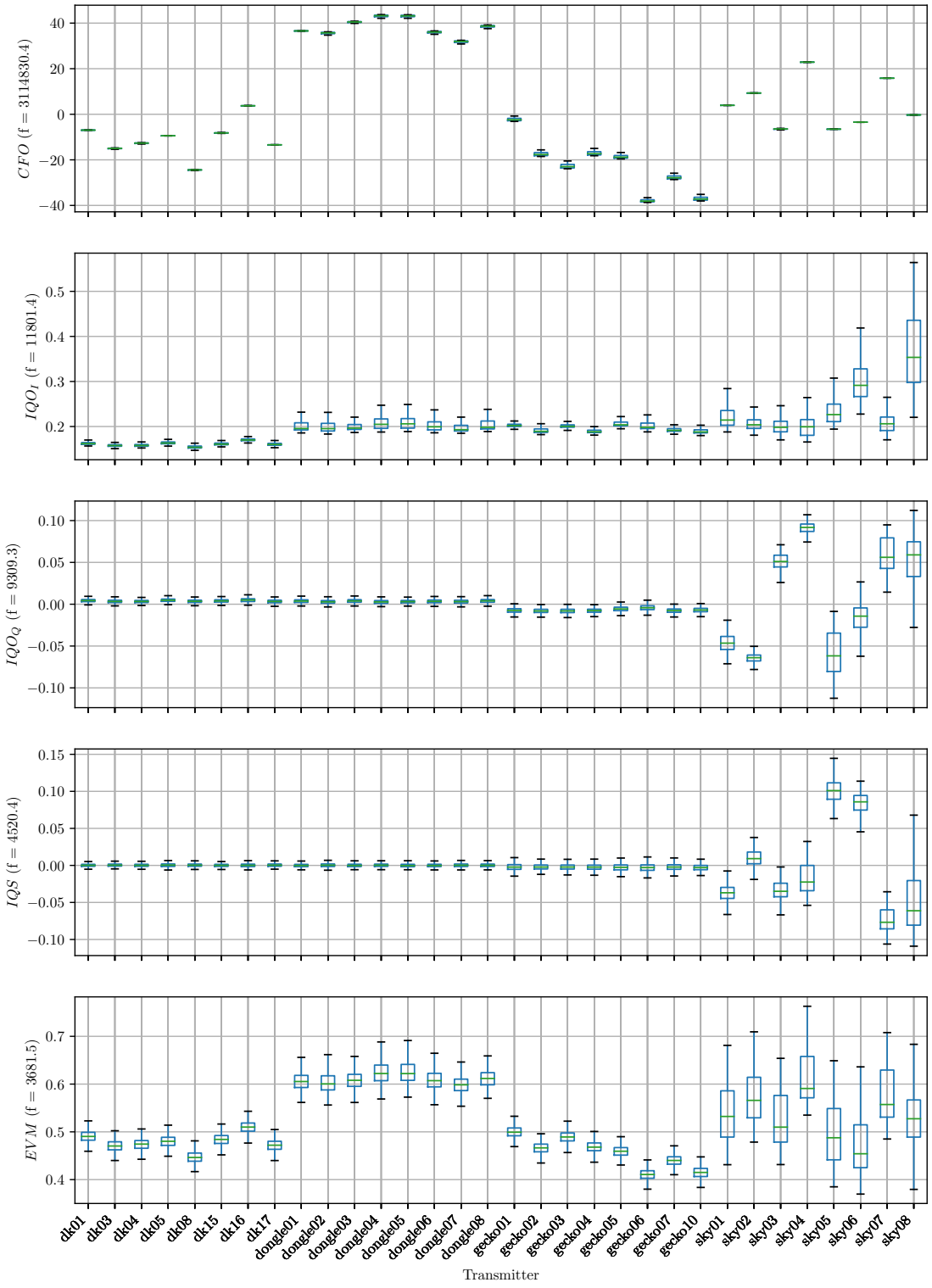


Figure 5.2: Variance of the CFO, EVM, IQO and IQS feature over the dataset. The boxes extend from Q1 to Q3, while the whiskers indicate the 1.5 IQR. Outliers are omitted.

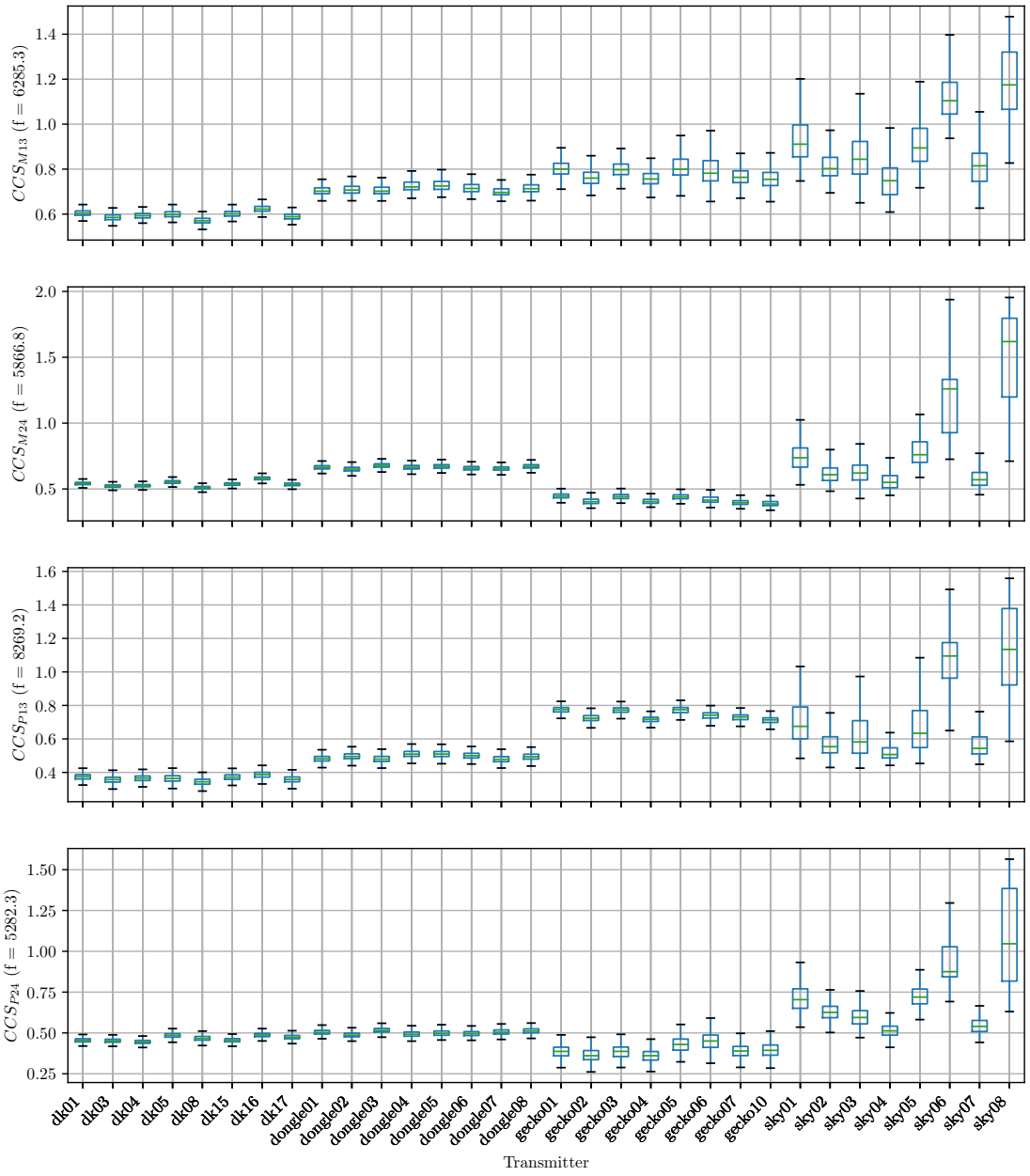


Figure 5.3: Variance of the CCS features over the dataset. The boxes extend from Q1 to Q3, while the whiskers indicate the 1.5 IQR. Outliers are omitted.

For ORF, classes refer to the individual transmitter devices. Figures 5.2, 5.3 show box plots for the different features together with their f-values, analyzed over the whole dataset.

First of all, several features differ quite a lot between devices of different type, and, on the other hand, fall into a similar range for devices of the same type. This matches the expectation that the hardware design contributes to the transmitter-specific characteristics. It also explains why device type identification is an easier task than individual device classification as shown in Section 5.4.

Out of all features, it is apparent that the Carrier Frequency Offset is a very valuable one, since it is almost constant for all frames from a given transmitter device and, at the same time, differs a lot between different transmitters.

In general, the Tmote Sky devices present a very large variance for all constellation-based features. A possible explanation for their less constant behavior is their age: The hardware design and manufacturing quality has presumably improved during the 10 years difference in age. Also, hardware aging effects may manifest themselves in unstable characteristics.

5.4 Classification Performance

This part investigates the performance of the final step in the ORF pipeline, the Random Forest classifier. As for all ML algorithms, apart from the quality of the input data, the final performance also heavily depends on a set of hyperparameters used during model training. Model and training set size in general tend to correlate with the performance of an ML model and the model size of a Random Forest classifier increases with the number of decision subtrees and the training set size (if no maximum depth per decision tree is specified).

To find a good compromise between classifier accuracy and memory usage, a number of Random Forest classifier models are trained with a varying number of subtrees N_{trees} and a varying training set size. The training sets for this experiment are balanced and contain N_{train} samples per transmitter device. The same test set is used for all configurations. Other hyperparameters affecting the performance of the classifier are left unchanged. This experiment is meant to give a feeling of the potential classifier performance, and does not aim to find the absolutely optimal parameters for small model size and high accuracy.

Figure 5.4 shows the memory footprint of the on-board classifier for different configurations, compared to the respective average and worst-case precision obtained using those configurations (the average and worst-case recall values are very similar to those reported values for all the experiments). As expected, the model size in ROM correlates roughly linearly both with the number of subtrees and the number of training samples, and the average and worst-case precision tend to improve with higher numbers of the

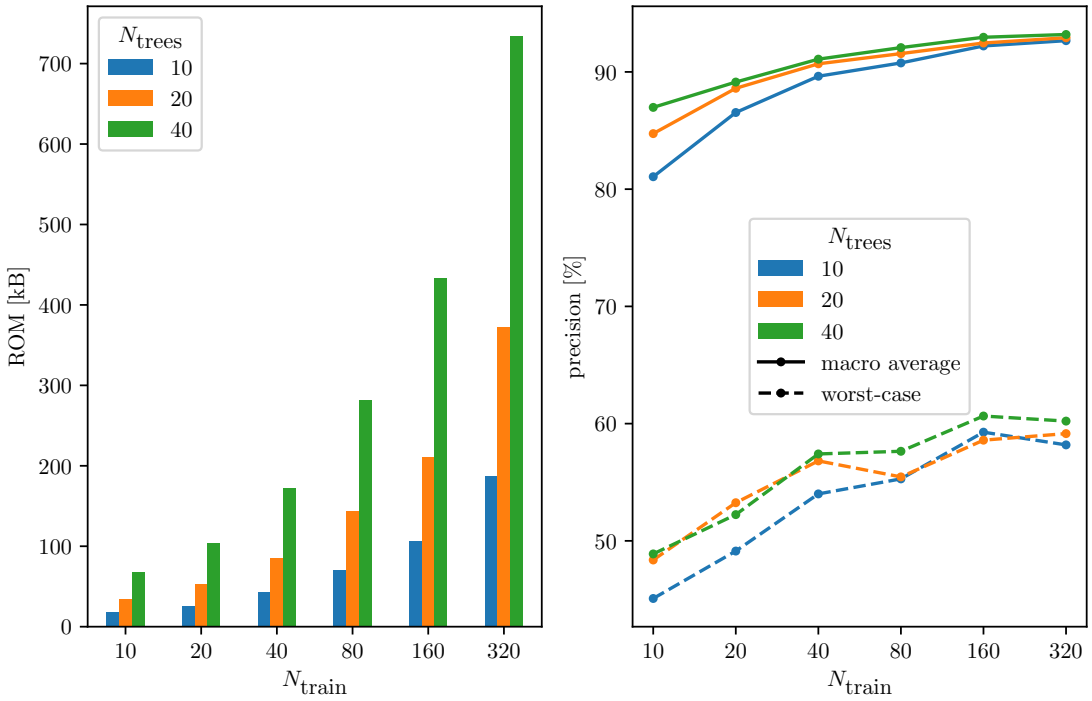


Figure 5.4: Memory footprint and performance of Random Forest classifiers trained with different hyperparameters.

hyperparameters, too. However, while the number of subtrees has a high impact on the model size, the performance improvements are relatively small, especially for a higher number of training samples. At the same time, the average precision seems to converge to a maximum around 93 % with larger training sets, suggesting that the input data quality, i.e., the considered features, impose an upper bound on the performance.

To gain better insight into the classifier performance for individual devices, two different configurations are taken as examples and analyzed in more detail:

- $N_{\text{trees}} = 40, N_{\text{train}} = 20$, which achieves an average precision of over 90 % while occupying only slightly more than 100 kB of ROM
- $N_{\text{trees}} = 20, N_{\text{train}} = 160$, which results in an improvement of 1.65 % (4.11 %) in average (worst-case) precision, while roughly doubling the memory footprint.

The confusion matrices of the two models in Figure 5.5 are normalized to the true values (rows) and show that, in both cases, most transmitters are correctly classified to 100 %. Only some device pairs such as `dongle04` and `dongle05` with similar fingerprints (cf. Figure 5.2) are misclassified among each other by both models. However, the percentage of misclassification for these device pairs decreases with a larger model size.

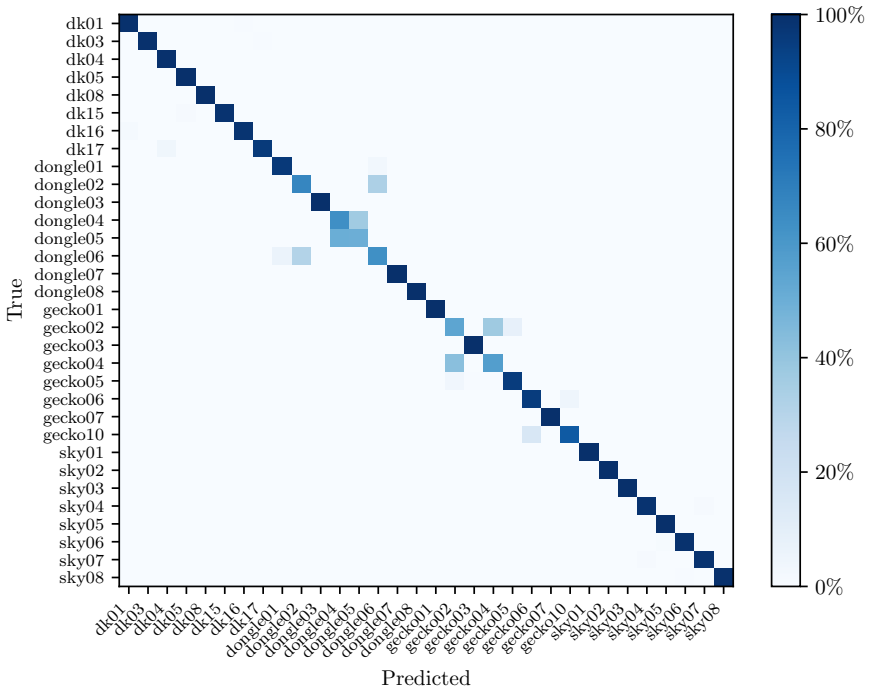
Excluding `dongle02`, `dongle04` and `gecko02` (being one out of two similar devices, respectively) from the dataset results in more than 99 % average precision and recall over the reduced dataset. This suggests that the classification is very accurate as long as all considered transmitters have sufficiently distinct radiometric fingerprints. Table 5.4 compares the classification performance for the two selected configurations according to the metrics defined in Section 2.4, before and after excluding the three devices.

Table 5.4: ORF system performance in terms of classifier evaluation metrics for the two example configurations. The values in parenthesis are obtained on a reduced dataset where `dongle02`, `dongle04` and `gecko02` are omitted.

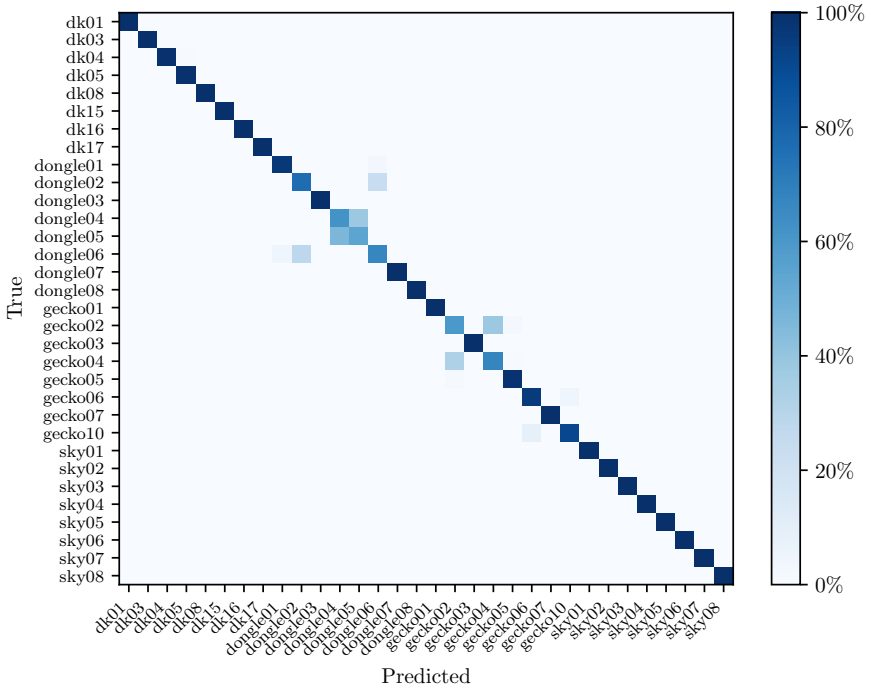
	$N_{\text{trees}} = 40, N_{\text{train}} = 20$ [%]	$N_{\text{trees}} = 20, N_{\text{train}} = 160$ [%]
$\text{precision}_{\text{macro}}$	90.9244 (97.8945)	92.6734 (99.0599)
$\text{recall}_{\text{macro}}$	90.9392 (97.8692)	92.6893 (99.0564)
accuracy	90.9499 (97.8718)	92.7005 (99.0567)
$\text{precision}_{\text{wc}}$	53.5789 (86.9776)	57.6835 (92.7294)
$\text{recall}_{\text{wc}}$	49.7886 (82.3654)	54.2184 (92.4411)

As expected due to the variance in features among different device *types*, both models never misclassify a transmitter as another device type. That means that ORF achieves correct classification in 100 % of the cases if only the device type is of interest.

Random Forest classifier models have the property that the importance of individual features for the classification can be conveniently deduced from the subtrees. Figure 5.6 shows the reported feature importance for the two example configurations. As expected



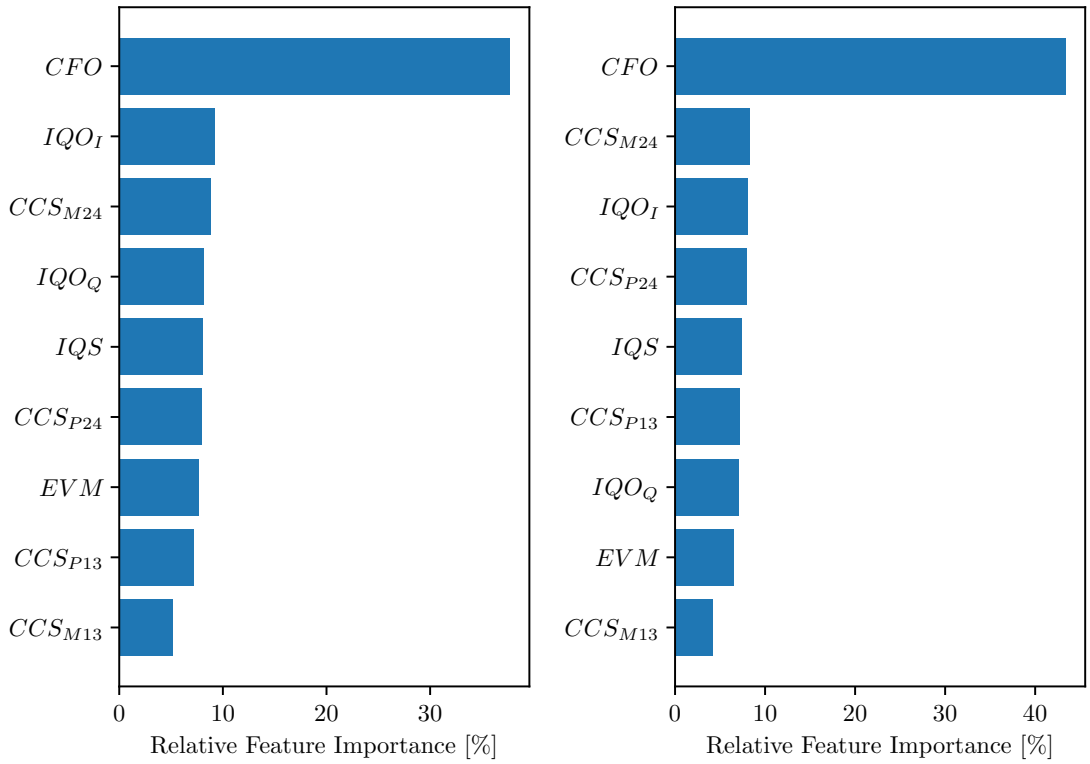
(a) $N_{\text{trees}} = 40, N_{\text{train}} = 20$



(b) $N_{\text{trees}} = 20, N_{\text{train}} = 160$

Figure 5.5: Confusion matrices for classifiers trained for the two example configurations.

from the feature analysis in Section 5.3, the CFO feature is by far the most important one throughout all models, while the remaining features contribute roughly by the same amount.



(a) $N_{\text{trees}} = 40, N_{\text{train}} = 20$ (b) $N_{\text{trees}} = 20, N_{\text{train}} = 160$

Figure 5.6: Relative feature importance for Random Forest classifiers trained for the two example configurations.

5.5 Resource Consumption

As for any embedded system with hardware constraints, the resource requirements of ORF in terms of memory, energy and execution time are important to consider. The following two subsections investigate this resource consumption for the configuration that is also used throughout the rest of the evaluation, with an I/Q sampling duration of 480 µs corresponding to 15 B of an IEEE 802.15.4 frame or roughly 3800 I/Q samples.

5.5.1 Memory Usage

ORF is implemented with memory constraints in mind, but is not majorly optimized for minimal memory consumption.

The selected hardware platform provides 512 kB of ROM (for code and static data) and 128 kB of RAM memory space. Table 5.5 gives a comprehensive overview of the space in memory occupied by each step in the ORF sampling, receiver and feature-extraction pipeline. The numbers for RAM memory consumption are taken from the discussion in Section 4.2. The ROM space needed per pipeline step is obtained by selectively enabling only one pipeline step at a time and comparing the binary size to a baseline compilation with the entire pipeline disabled. The exact numbers depend on the concrete implementation and should not be taken too seriously. Rather, they are meant to show the order of magnitude of the memory requirement for each pipeline step.

Table 5.5: Memory footprint of (individual parts of) the ORF on-board C implementation. The reported total numbers do not correspond to the simple sum of the entries.

	ROM [B]	RAM [B]
I/Q sampling	682	15200
Conversion to `float`	672	30400
Normalization	1941	0
Matched filter	1704	64
Frequency synchronization	66617	32768
Phase synchronization	4352	12000
Phase ambiguity resolution	4904	1136
Time synchronization	2300	0
Feature extraction	3529	0
Total	106465	63168

The reported total ROM space does not match the sum of the individual pipeline steps for two reasons: First, several pipeline steps make use of the same library functions which only need to be included once in the final binary, decreasing the total binary size. Second, the Zephyr build system unconditionally adds code for bootup and other system functions, in turn increasing the binary size.

The frequency synchronization step alone occupies the most amount of ROM, since it uses a large amount of static data for the Gaussian window and during FFT computation. All other pipeline steps have a negligible ROM memory usage below 5 kB each.

Since the memory content of previous pipeline steps, apart from the I/Q sample buffer itself, is not needed in subsequent ones, the total RAM usage can be estimated as the sum of the I/Q sample buffer of 30.4 kB and the maximum RAM usage of the pipeline,

32.768 kB; resulting in 63.168 kB. ORF itself thus occupies less than half of the available space in RAM, allowing for real-world deployments alongside other device functionality.

In terms of ROM, the memory usage of the ORF pipeline up to the classification step amounts to a bit more than 100 kB. The classifier can thus at most occupy 400 kB of additional ROM. Out of the different classifier configurations discussed in Section 5.4, $N_{\text{trees}} = 20, N_{\text{train}} = 160$ with 210.37 kB is selected for all remaining experiments, leaving enough free ROM space for combination with other device functionality.

5.5.2 Energy and Time

Two small benchmarking experiments are used to investigate the delay and energy consumption of a single classification: First, the delay and energy is investigated for the individual steps of the ORF pipeline, after frame reception and I/Q sampling. Second, the energy consumption overhead incured by I/Q sampling during frame reception is investigated.

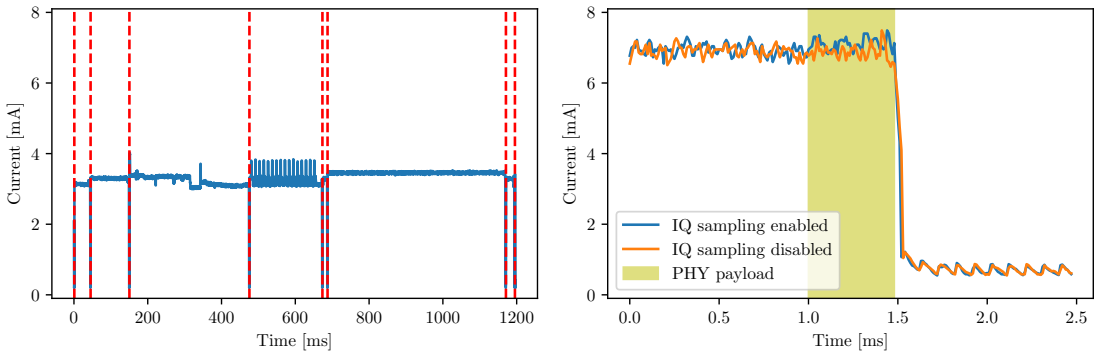
For the first experiment, one frame out of the dataset is processed 100 times on board, marking the start of the different pipeline steps using a pulse on a GPIO pin. The GPIO output and current is sampled at 10 kHz ($\Delta T = 0.1$ ms) using the nRF Power Profiler Kit II in Source Meter mode with a constant voltage supply of $U = 3$ V [42].

Figure 5.7a shows the current measurements $I(t)$, together with the GPIO pulses, for a single ORF pipeline run as an example. The current measurements are transformed to energy consumption, attributed to the individual pipeline steps and averaged over the 100 measurements. Table 5.6 summarizes the average energy consumption and delay of each pipeline step after the I/Q sampling.

Table 5.6: Delay and energy consumption per pipeline step, averaged over 100 measurements.

	Delay [ms]	Energy [uJ]
Conversion to `float32_t`	5.1	51.1
Normalization	43	404.9
Matched Filter	104.2	1030.3
Frequency Synchronization	324.1	3142.7
Phase Synchronization	196.7	1908.5
Phase Ambiguity Resolution	13	126.8
Time Synchronization	483.2	5003.7
Feature Extraction	23.4	227.3
Classifier	0.4	2.7
Total	1193	11898.2

The total delay for processing a single frame with ORF is 1.2 s, of which the most time is spent during the time synchronization step. This can be explained by the fact that this is the only step where the samples need to be processed one after another. The second most time-consuming step is the frequency synchronization where a computationally demanding 4096-FFT is calculated. The remaining steps of the receiver synchronization pipeline together make up for most of the total time consumption. In contrast, the final classification step takes barely 370 µs.



(a) Comparing different steps of the feature extraction pipeline.

(b) Comparing frame reception with and without I/Q sampling.

Figure 5.7: Current measurements for different parts of the ORF pipeline.

Since the average current drain is between 3 and 3.5 mA throughout all the pipeline steps, the energy consumption calculated as $E = \sum_t I(t) \cdot U \cdot \Delta T$ is roughly proportional to the processing delay.

The second experiment investigates the energy consumption overhead caused by I/Q sampling during frame reception as follows: The current drain of the receiver is sampled at 100 kHz during the reception of 100 frames, with and without I/Q sampling enabled. The duration of the I/Q sampling is marked by a GPIO pulse between the `ADDRESS` and `END` event (cf. Section 4.1.2). Figure 5.7b shows the measurements for two example frames with and without I/Q sampling, respectively. The GPIO pulse length of 480 µs exactly matches the on-air transmission time for the PHY header and payload part of the frame. It can be seen that in both cases there is no big difference in current drain between the time during which the radio is actively listening on the channel (left of the GPIO pulse) and the actual frame reception period. As soon as the radio is disabled and the microprocessor goes into idle mode (right of the GPIO pulse), the current drain drops significantly.

However, when considering only the frame reception, the current drain appears to be slightly higher when I/Q sampling is performed. Averaging over the 100 received frames, this difference can be confirmed and quantified: The average energy consumption for a single frame reception without I/Q sampling is 9.9131 mJ. When I/Q sampling is

performed, the average energy consumption goes up to 10.3194 mJ, indicating that the I/Q sampling itself (including storing the samples to RAM) consumes 406.3 µJ for the selected frame length.

5.6 Feature Stability

An interesting aspect of features capturing the radiometric fingerprint of a transmitter is their stability over time: How long does it take for a fingerprint to change drastically enough to deteriorate the classifier performance?

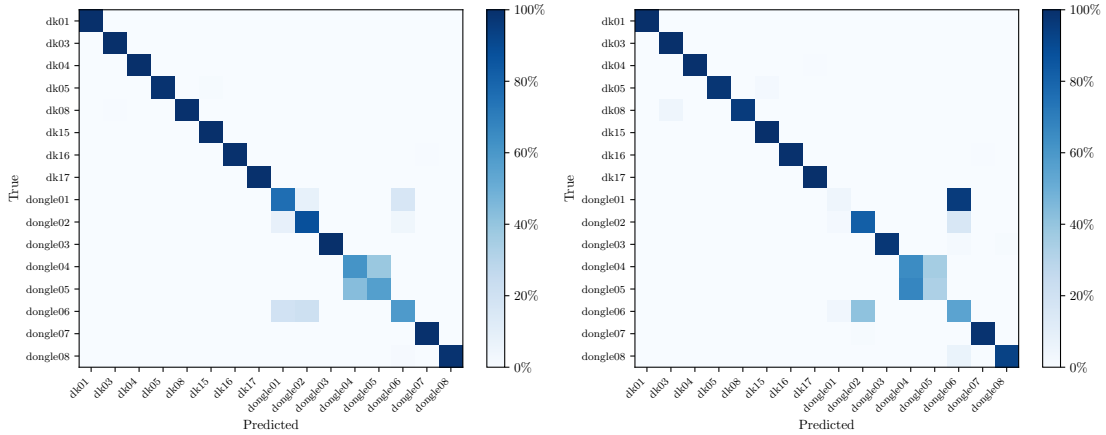
All previous evaluation findings are based on the dataset acquired using the experimental setup detailed in Section 5.1, where all 1000 frames per transmitter are captured back-to-back. However, a second data collection has been realized at a time difference of roughly two months with a subset of the used transmitter devices (only `dk*` and `dongle*` nRF52840 devices). Because only two device types are considered for this experiment and there are multiple `dongle*` devices with similar radiometric fingerprints (cf. Section 5.3), the classifier performance metrics differ from the results obtained over the whole dataset and reported in Section 5.4.

Figure 5.8 and Table 5.7 show the classification performance when training the classifier using the old dataset D_{old} and testing with frames from D_{new}. The macro precision and recall as well as the accuracy degrade slightly from 89 % to around 82 %, mainly due to some pair of devices of the same type (`dongle*`) being misclassified. This is also reflected by the worst-case metrics which drop significantly from around 57 % to 32 % and even 4 %. The misclassification already present on D_{old} due to very similar features is magnified when testing over D_{new} with slightly changed feature values.

On the other hand, the classification performance is stable for devices that are already reliably classified on D_{old}, suggesting that the ORF classifier is robust over time when the features for individual devices are sufficiently distinct. Further research is needed to obtain a better understanding of the feature stability over time.

Table 5.7: ORF classifier performance when trained with frames of D_{old} and tested with frames of D_{old} and D_{new}.

	$D_{test} \subseteq D_{old}$ [%]	$D_{test} = D_{new}$ [%]
precision_{macro}	89.6747	83.0400
recall_{macro}	89.6655	82.5326
accuracy	89.6096	82.5401
precision_{wc}	58.4816	31.432
recall_{wc}	56.6832	4.5596



(a) $D_{\text{train}}, D_{\text{test}} \subseteq D_{\text{old}}$ (b) $D_{\text{train}} \subseteq D_{\text{old}},\ D_{\text{test}} = D_{\text{new}}$

Figure 5.8: Confusion matrices obtained with different datasets on a classifier trained with frames of one of them.

6 Conclusions and Future Work

6.1 Conclusions

ORF is the first on-board radiometric fingerprinting system that can be deployed on COTS devices in a distributed network. It is realized by taking advantage of the I/Q sampling functionality available on the nRF52833 SoC. Although originally designed for BLE DFE, ORF shows that I/Q sampling on this SoC can be used to enable capturing other parts of frames with lengths of up to 500 µs, even using a different physical-layer protocol. The proof-of-concept implementation of ORF processes and classifies frames complying with the IEEE 802.15.4 2450 MHz PHY protocol and achieves an average accuracy of 92.7 % on a challenging dataset with 32 devices of four different types. One single classification decision takes roughly one second and consumes around 11.8 mJ of energy. The memory footprint is as low as 50 % of the available RAM and less than 75 % of the available ROM.

6.2 Future Work

By showing that on-board I/Q sampling is feasible with sufficient quality for a radiometric fingerprinting system, ORF opens the door for a multitude of interesting research directions:

First of all, the current implementation of ORF certainly leaves room for further improvement and optimization in terms of memory consumption, pipeline delay and classification performance. Methods for achieving higher accuracy could be: re-iterated feature engineering, i.e., defining and extracting better features; more careful feature selection; classifier hyperparameter optimization; or using a different classification algorithm altogether. Apart from that, on-board radiometric fingerprinting could also be explored for other physical-layer protocols. While IoT deployments are often built on IEEE 802.15.4-based protocols, BLE is another promising candidate for on-board radiometric fingerprinting [14].

The dataset used for evaluation purposes in this work has been captured under ideal conditions, largely ignoring wireless propagation effects such as multipath and noise. Combining the ORF approach with previous work on increased robustness to channel distortions such as [8] is expected to facilitate application to real-world scenarios. Another aspect which has only been briefly evaluated for ORF is the stability of features and

thereby of the classification as time progresses. Further investigation in this direction would help to assess the feasibility of real-world deployments.

Last but not least, having access to raw I/Q samples potentially enables applications other than radiometric fingerprinting which rely on on-board I/Q sampling.

References

[1] Ö. H. Tekbaş, O. Üreten, and N. Serinken, "Improvement of transmitter identification system for low SNR transients," *Electron. Lett.*, vol. 40, no. 3, p. 182, 2004, doi: 10.1049/el:20040160.

[2] D. A. Knox and T. Kunz, "Wireless Fingerprints Inside a Wireless Sensor Network," *ACM Trans. Sen. Netw.*, vol. 11, no. 2, pp. 1–30, Mar. 2015, doi: 10.1145/2658999.

[3] M. Woolley, "Bluetooth Direction Finding." Oct. 13, 2021. Available: https://www.bluetooth.com/wp-content/uploads/Files/developer/RDF_Technical_Overview.pdf

[4] Q. Xu, R. Zheng, W. Saad, and Z. Han, "Device Fingerprinting in Wireless Networks: Challenges and Opportunities." arXiv, Jan. 06, 2015. Accessed: Mar. 28, 2023. [Online]. Available: http://arxiv.org/abs/1501.01367

[5] O. Ureten and N. Serinken, "Wireless security through RF fingerprinting," *Canadian Journal of Electrical and Computer Engineering*, vol. 32, no. 1, pp. 27–33, 2007, doi: 10.1109/CJECE.2007.364330.

[6] V. Brik, S. Banerjee, M. Gruteser, and S. Oh, "Wireless device identification with radiometric signatures," in *Proceedings of the 14th ACM international conference on Mobile computing and networking*, in MobiCom '08. New York, NY, USA: Association for Computing Machinery, Sep. 2008, pp. 116–127. doi: 10.1145/1409944.1409959.

[7] T. D. Vo-Huu, T. D. Vo-Huu, and G. Noubir, "Fingerprinting Wi-Fi Devices Using Software Defined Radios," in *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, in WiSec '16. New York, NY, USA: Association for Computing Machinery, Jul. 2016, pp. 3–14. doi: 10.1145/2939918.2939936.

[8] W. Yan, T. Voigt, and C. Rohner, "RRF: A Robust Radiometric Fingerprint System that Embraces Wireless Channel Diversity," in *Proceedings of the 15th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, in WiSec '22. New York, NY, USA: Association for Computing Machinery, May 2022, pp. 85–97. doi: 10.1145/3507657.3528542.

[9] C. K. Dubendorfer, B. W. Ramsey, and M. A. Temple, "An RF-DNA verification process for ZigBee networks," in *MILCOM 2012 - 2012 IEEE Military Communications Conference*, Oct. 2012, pp. 1–6. doi: 10.1109/MILCOM.2012.6415804.

[10] L. Peng, A. Hu, J. Zhang, Y. Jiang, J. Yu, and Y. Yan, "Design of a Hybrid RF Fingerprint Extraction and Device Classification Scheme," *IEEE Internet of Things Journal*, vol. 6, no. 1, pp. 349–360, Feb. 2019, doi: 10.1109/JIOT.2018.2838071.

[11] X. Zhou, A. Hu, G. Li, L. Peng, Y. Xing, and J. Yu, "A Robust Radio-Frequency Fingerprint Extraction Scheme for Practical Device Recognition," *IEEE Internet of Things Journal*, vol. 8, no. 14, pp. 11276–11289, Jul. 2021, doi: 10.1109/JIOT.2021.3051402.

[12] P. Robyns, E. Marin, W. Lamotte, P. Quax, D. Singelée, and B. Preneel, "Physical-layer fingerprinting of LoRa devices using supervised and zero-shot learning," in *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, Boston Massachusetts: ACM, Jul. 2017, pp. 58–63. doi: 10.1145/3098243.3098267.

[13] Y. Jiang, L. Peng, A. Hu, S. Wang, Y. Huang, and L. Zhang, "Physical layer identification of LoRa devices using constellation trace figure," *J Wireless Com Network*, vol. 2019, no. 1, p. 223, Sep. 2019, doi: 10.1186/s13638-019-1542-x.

[14] D. Nilsson and W. Yan, "Identifying Bluetooth Low Energy Devices," in *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, in SenSys '21. New York, NY, USA: Association for Computing Machinery, Nov. 2021, pp. 375–376. doi: 10.1145/3485730.3492880.

[15] M. Rice, *Digital communications: a discrete-time approach*. Pearson Education India, 2009.

[16] E. W. Weisstein, "Harmonic Addition Theorem," *MathWorld–A Wolfram Web Resource.* https://mathworld.wolfram.com/HarmonicAdditionTheorem.html (accessed May 23, 2023).

[17] J. Huettner, S. Reinhardt, and M. Huemer, "Low complex IQ-imbalance compensation for low-IF receivers," in *2006 IEEE Radio and Wireless Symposium*, San Diego, CA, USA: IEEE, 2006, pp. 303–306. doi: 10.1109/RWS.2006.1615155.

[18] F. Zhuo, Y. Huang, and J. Chen, "Radio Frequency Fingerprint Extraction of Radio Emitter Based on I/Q Imbalance," *Procedia Computer Science*, vol. 107, pp. 472–477, 2017, doi: 10.1016/j.procs.2017.03.092.

[19] "IEEE Standard for Information technology– Local and metropolitan area networks– Specific requirements– Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (WPANs)." Sep. 07, 2006.

[20] Babcock University *et al.*, "Supervised Machine Learning Algorithms: Classification and Comparison," *IJCTT*, vol. 48, no. 3, pp. 128–138, Jun. 2017, doi: 10.14445/22312803/IJCTT-V48P126.

[21] Nordic Semiconductor, Ed., "nRF52833 Product Specification." Nov. 08, 2021. Accessed: Mar. 28, 2023. [Online]. Available: https://infocenter.nordicsemi.com/pdf/nRF52833_PS_v1.5.pdf

[22] Nordic Semiconductor, Ed., "nWP-036 Direction Finding White Paper." Aug. 24, 2020. Accessed: Jun. 02, 2023. [Online]. Available: https://infocenter.nordicsemi.com/pdf/nwp_036.pdf

[23] A. Candore, O. Kocabas, and F. Koushanfar, "Robust stable radiometric fingerprinting for wireless devices," in *2009 IEEE International Workshop on Hardware-Oriented Security and Trust*, San Francisco, CA, USA: IEEE, 2009, pp. 43–49. doi: 10.1109/HST.2009.5224969.

[24] Inc. The MathWorks, "Communications Toolbox," manual. Available: https://www.mathworks.com/help/comm/

[25] C. R. Harris *et al.*, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, Art. no. 7825, Sep. 2020, doi: 10.1038/s41586-020-2649-2.

[26] "CMSIS-DSP: Software Library." Arm Ltd. Accessed: May 23, 2023. [Online]. Available: https://arm-software.github.io/CMSIS-DSP/latest/

[27] G. Turin, "An introduction to matched filters," *IRE Transactions on Information Theory*, vol. 6, no. 3, pp. 311–329, Jun. 1960, doi: 10.1109/TIT.1960.1057571.

[28] Jonathan Olds, "Designing an OQPSK demodulator," *OQPSK demodulator design for 10.5kbps Aero signals*, 2016. https://jontio.zapto.org/hda1/oqpsk.html (accessed May 23, 2023).

[29] M. Gasior and J. L. Gonzalez, "Improving FFT Frequency Measurement Resolution by Parabolic and Gaussian Spectrum Interpolation," *AIP Conference Proceedings*, vol. 732, no. 1, pp. 276–285, Nov. 2004, doi: 10.1063/1.1831158.

[30] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct. 2001, doi: 10.1023/A:1010933404324.

[31] F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, no. 85, pp. 2825–2830, 2011, Accessed: May 23, 2023. [Online]. Available: http://jmlr.org/papers/v12/pedregosa11a.html

[32] J. Nordby, "emlearn: Machine Learning inference engine for Microcontrollers and Embedded Devices." Zenodo, Mar. 10, 2019. doi: 10.5281/zenodo.2589394.

[33] Nordic Semiconductor, Ed., "nRF52833 DK User Guide." Dec. 03, 2020. Accessed: May 23, 2023. [Online]. Available: https://infocenter.nordicsemi.com/pdf/nRF52833_DK_User_Guide_20201203.pdf

[34] Nordic Semiconductor, Ed., "nRF52840 DK User Guide." Dec. 03, 2020. Accessed: May 23, 2023. [Online]. Available: https://infocenter.nordicsemi.com/pdf/nRF52833_DK_User_Guide_20201203.pdf

[35] Nordic Semiconductor, Ed., "nRF52840 Product Specification." Nov. 30, 2021. Accessed: May 23, 2023. [Online]. Available: https://infocenter.nordicsemi.com/pdf/nRF52840_PS_v1.7.pdf

[36] Nordic Semiconductor, Ed., "nRF52840 Dongle User Guide." May 15, 2023. Accessed: May 23, 2023. [Online]. Available: https://infocenter.nordicsemi.com/pdf/nRF52840_Dongle_User_Guide_v2.1.1.pdf

[37] Silicon Labs, Ed., "UG309: Thunderboard Sense 2 User's Guide." Aug. 2020. Accessed: May 23, 2023. [Online]. Available: https://www.silabs.com/documents/public/user-guides/ug309-sltb004a-user-guide.pdf

[38] Moteiv Corporation, Ed., "Tmote Sky: Datasheet." Nov. 13, 2006. Accessed: May 23, 2023. [Online]. Available: http://www.crew-project.eu/sites/default/files/tmote-sky-datasheet.pdf

[39] V. Kumar, "Feature Selection: A literature Review," *SmartCR*, vol. 4, no. 3, Jun. 2014, doi: 10.6029/smartcr.2014.03.007.

[40] J. Li *et al.*, "Feature Selection: A Data Perspective," *ACM Comput. Surv.*, vol. 50, no. 6, pp. 1–45, Nov. 2018, doi: 10.1145/3136625.

[41] N. Omer Fadl Elssied, O. Ibrahim, and A. Hamza Osman, "A Novel Feature Selection Based on One-Way ANOVA F-Test for E-Mail Spam Classification," *RJASET*, vol. 7, no. 3, pp. 625–638, Jan. 2014, doi: 10.19026/rjaset.7.299.

[42] Nordic Semiconductor, Ed., "Power Profiler Kit II User Guide." Aug. 16, 2022. Accessed: May 30, 2023. [Online]. Available: https://infocenter.nordicsemi.com/pdf/PPK2_User_Guide_v1.0.1.pdf