Portable, Low-Cost, and Low-Power mmWave Radar Imaging for Concealed Weapon Detection

Trinity Term 2021-22

Author: David Grigorovich Khachaturov
Supervisor: Professor Kasper Rasmussen
Degree: MSc in Advanced Computer Science
Date: 30/08/2022

Abstract

This project proposes an innovative portable, low-cost, and low-power method of mmWave radar imaging, and explores its concealed weapon detection and classification capabilities. A 3D radar scan of a concealed object is produced using a scanning device that combines mmWave radar sensor for imaging purposes and a smartphone for position tracking. Classification is performed on these 3D radar scans using a resource-efficient convolutional neural network. A physical prototype implementing this method was built and evaluated. The prototype achieves a high 85%+ classification accuracy amongst three object classes, demonstrating the viability of the proposed method.

Contents

Chapter 1

Introduction

Millimetre wave (mmWave) radar refers to a class of radar technologies that operate in the $30 - 300\,\text{GHz}$ frequency range. The technology has been around for well over a century, but has seen a recent resurgence of academic and commercial interest [1]. Nowadays, mmWave technologies are used in a wide range of areas, including 5G cellular networks and airport security scanners [2, 3].

This project will explore the use of mmWave as a means of radar imaging. As a form of imaging technology, it has a few distinct advantages over its alternatives (discussed in Section 2.1):

- The high operating frequency means that the physical antennas can be small, reducing the hardware footprint and making the sensor compact,
- The large bandwidth of the sensors, often in the region of $4\,\text{GHz}$, provides a higher transmission rate and better interference resistance, as well as finer imaging resolution,
- The high frequencies mean that multiple sensors configured to a short-range scan can operate on the same frequency without interfering with one another,
- The nature of mmWave signals means that they easily penetrate clothes, cardboard, and other organic-based materials, but are reflected off of metallic objects, making it an ideal technology for security imaging purposes [4].
- There have been recent advances in developing efficient low-power mmWave sensors [5].

mmWave radar has a lot of potential, especially due to its compact antenna size and low-power sensors. However, existing imaging methods that utilise mmWave technology, such as the ones used in the aforementioned airport security scanners, are often very limited – the devices are bulky and expensive, requiring an uninterrupted power supply to operate.

While more portable imaging methods exist, they have major caveats when it comes to their operation. For example, infrared cameras do not work reliably when imaging objects of similar temperatures, such as weapons in a crate.

As such, there is a distinct niche to be filled with a *portable*, *low-power*, and *low-cost* radar imaging method. This dissertation proposes such a method and demonstrates its viability via a prototype.

While radar imaging has a wide variety of applications, ranging from industrial use for quality control on a factory production belt, to object detection for autonomous vehicles, I will focus on its applicability to *concealed weapon detection* (CWD). This is an area of research that focuses on detecting and potentially classifying weapons in environments whereby the weapon is hidden or obstructed from view by some means. Examples include detecting firearms concealed on-person, or screening containers for illegal weapons.

The rest of the chapter discusses the motivation behind this research, followed by a high-level description of the proposed radar imaging method. The chapter then concludes with an explanation of the structure of this dissertation.

1.1 Motivation

As will be discussed in more detail in Section 2.1, existing forms of portable imaging technologies are very limited in their capabilities. One of the most common forms of "imaging" in high-security places are static or portable metal detectors. These are very crude devices that are unable to identify the nature of the detected metallic object, merely signalling to the operator that some metal was found on-person or in a piece of luggage. These leaves the operator vulnerable due to lack of awareness.

More advanced technologies such as infrared cameras are able to provide higher-fidelity imaging, but are limited to distinguishing between objects with temperature differences. This is useful when attempting to detect weapons concealed on-person but fails in common security scenarios such as being able to identify weapons in a crate.

The latter scenario can be solved by using X-Ray technologies. However, X-Ray scanners often require a stable and high-voltage power supply, and are mostly static by nature. While there have been demonstrations of portable X-Ray scanners for military use, these carry a high radiation risk to the operator and surrounding environment, limiting their applicability in civilian or commercial environments [6].

On the other hand, mmWave frequencies are considered non-ionising, making them safe for human exposure [7]. As previously mentioned, mmWave scanners are also low-powered in stark contrast to X-Ray scanners. This makes mmWave a natural choice to explore as a form radar imaging technology.

1.2 Real-World Applications

mmWave radar imaging has a lot of potential use-cases. One such case is being considered by Stofl, a UK-based startup currently engaged in developing infrastructure and solutions that combine Blockchain and Low-Power-WAN technologies [8]. Stofl is partnering with *anti-poaching* charities in South Africa to assist their efforts, specifically with weapon detection. The project requires a *low-power*, and *low-cost* method of detecting and classifying concealed metallic objects to assist in anti-poaching exercises.

Existing methods are not very well suited to a Savannah environment, where the weapon detection systems need to be transported around in light vehicles and where there is a distinct lack of constant power sources due to the mobile nature of the operations. These efforts are also mainly volunteer-run, with subsequently very low funding [9].

South Africa has the largest rhinoceros population in the world and has been at the forefront of rhino conservation. Unfortunately, from 2007-2014, it has seen a $9,000\%$ increase in rhino poaching [10]. Elephant poaching in South Africa is also a source of major concern – roughly $40,000$ elephants are killed a year for their ivory [11]. This alarming rise in numbers is a call to action.

While poachers often do not pose a direct threat to rangers, who monitor and protect the animals and their natural habitats, the poachers are often well-trained and armed. They often use weapons – guns and knives – to hunt their pray and extract products such as ivory. Rangers hence constantly risk their lives during interventions as they often have insufficient information about the exact intentions and capabilities of the poachers. This is where CWD proves to be invaluable: it allows rangers to assess the situation at a distance to form an appropriate plan of action and to gauge the threat level in case back-up is needed.

1.3 Proposed Method

The mmWave radar imaging method proposed in this project aims to tackle the pitfalls of the existing CWD and radar imaging methods that were outlined above. Namely, their cost, high power requirements, and static nature. The method proposed below is portable in nature and can be put into practice using cheap off-the-shelf components, as will be discussed in Section 1.3.1 and Chapter 3.

The proposed method is as follows: a mmWave radar sensor is used in combination with a position and orientation tracking device to produce 3D radar scans of concealed objects. The 3D scan is formed by combining multiple 2D radar scans taken from different points-of-view. To combine the 2D scans into a single 3D image, the position and orientation of the radar scanner at the time of each scan is needed.

As previously mentioned, mmWave signals easily penetrate clothes, cardboard, and other organic-based materials, but are reflected off of metallic objects. This means that the produced radar scans highlight the rough 3D shape of any concealed metallic object. Classification is performed on these 3D scans via a neural network to identify the exact nature of the concealed object.

This method is developed further in subsequent chapters and is put into practice via a physical prototype. A brief comparison between the produced prototype and existing commercial imaging methods is drawn in the subsection below.

1.3.1 Comparison

The prototype uses a radar sensor with a unit price of 358.80 USD, and an Android phone retailing at roughly 200 USD for tracking purposes. This places the total unit cost at just over 550 USD. As will be shown in Section 6.2, the prototype achieves a high classification accuracy of 85%+.

In comparison, a mmWave body scanner used in airports retailed for around 170,000 USD in 2010 (230,000 USD, adjusted for inflation) [12]. A basic metal detector retails for around 1,000 USD. In both cases, the prototype is *significantly* cheaper and offers comparable or better features – a metal detector cannot classify the detected objects, for example.

This serves as a testament to the advancement in low-cost radar imaging technology that this project develops.

1.4 Dissertation Structure

The rest of the dissertation is structured as follows:

- **Background** Chapter 2 covers the existing technical information needed to understand and appreciate the contributions of this dissertation. The chapter starts with an overview of existing CWD methods to highlight pitfalls and areas where they can be improved. The theory behind mmWave radar is then discussed together with an introduction to Visual Inertial Odometry (VIO), which is the position tracking method of choice for this project. Finally, the fundamentals of Deep Learning and its applicability to this project is discussed.
- **Scanner Design** Chapter 3 derives the problem statement and the subsequent scope of this project. The chapter then goes on to discuss the design decisions behind the proposed method, based on the project requirements outlined. These design decisions also include the prototype design, the choice of mmWave radar, tracking, and classification method.
- **Methodology** Chapter 4 provides a description of the algorithms used in the proposed method. This chapter also describes the experiment setup for demonstrating the prototype's capabilities, and the data post-processing pipeline employed for the classification task.

- **Implementation** Chapter 5 details the produced work as an implementation of the proposed method, including the hardware and software components of the prototype.
- **Evaluation** Chapter 6 describes the steps performed to verify the correctness of implementation of the method, and presents the prototype's experimental results. This is followed by a discussion of said results and the prototype as a whole.
- **Related Work** Chapter 7 contrasts the findings and proposed method of this project to previous work done in the field of CWD using mmWave radar; and how this project improves upon existing methods.
- **Conclusion** Chapter 8 summarises and draws conclusions from the project, and suggests potential extensions to the proposed method and prototype.

Chapter 2

Background

This chapter will cover existing technical information that is needed to understand the subsequent work that will be presented in this dissertation. A variety of weapon detection mechanisms and their pros and cons will be first discussed in Section 2.1. This will be taken as an opportunity to highlight the niche that is being targeted with the imaging method being proposed.

Section 2.2 will talk about the theory behind mmWave radar sensors and discuss the data they are able to produce. A short introduction to Visual Inertial Odometry (VIO) as a method of position tracking will be presented in Section 2.3. This tracking method will be used in the project to estimate the position and orientation of the mmWave radar device for the purposes of producing a 3D radar scan.

Finally, Section 2.4 will introduce Deep Learning and Convolutional Neural Networks (CNNs) as classification algorithms. 3D variants of traditional CNNs will be used in this project to perform the classification of the obtained 3D radar scans of concealed objects.

2.1 Weapon Detection

Concealed Weapon Detection (CWD) has significant real-world security applications, ranging from airport security to counter-terrorism. It is a well-studied subject with a wide variety of approaches. The choice of method usually depends on the constraints of a particular environment and the usable budget.

This section will cover these various methods together with the context in which they are commonly used and their specific advantages and disadvantages.

2.1.1 Metal Detectors

Metal detectors are one of the most common forms of weapon detection. They are nearly ubiquitously present in airports and other high-security locations such as embassies and nightclubs.

They perform a very basic function – namely detecting the presence of metallic objects. Metal detectors do not usually perform any classification, but are cheap and effective as an initial screening test. For example, if the metal detector rings in an airport you're often invited to a further scan via a mmMave body scanner. Metal detectors are often used for screening for on-person objects and are often accompanied by X-Ray machines for scanning luggage and other containers. Hand-held metal detectors are also used, but require very close proximity to the subject to work properly, making leaving the operator in a potentially vulnerable position.

Even given a metal detector's simplicity, this method can miss smaller metallic objects such as needles and does not detect non-metallic weapons, such as blow-darts and ceramic knives. Metal detectors also require co-operation on behalf of the individual being screened, so will not work very well in an adversarial environment.

2.1.2 X-Ray and Computed Tomography

X-Ray machines are also a relatively ubiquitous way of imaging objects to peer inside them without invasive methods. They are used in both the medical sphere and in security scenarios when scanning containers for dangerous or forbidden objects. For example, they are used in airports to screen carry-on bags for any hazardous objects [13].

A more modern approach to this is Computed Tomography (CT), which uses the same X-Ray wavelength to 'see through' solid material, but uses higher-energy waves for better penetrative properties. A CT machine takes scans from multiple angles to provide a 3D image of the object being imaged. This technology is also used in both the medical sphere and in high-security areas like airports to screen checked-in luggage and, increasingly, carry-on luggage [14, 15].

These systems work quickly and do not damage the contents of the container, including sensitive objects like film, due to the low intensity of radiation used during the scans. The setups are however not very portable, require large amounts of energy to run and again assume the explicit cooperation of the containers' owners to perform the scans. The scanning machines need the container to be small enough to fit within the scanner, limiting their usage when it comes to larger and heavier crates.

2.1.3 Terahertz Spectroscopy

Terahertz Spectroscopy uses 10^{11} to 10^{13} Hz light to perform object imaging. The idea behind THz spectroscopy is that different materials have very distinct absorption and reflection properties when exposed to different wavelengths of THz waves. The approach relies on material surface properties, meaning that THz spectroscopy is not limited to metallic objects, being able to also distinguish between different organic and inorganic materials [16].

THz spectroscopy is considered a reliable and tested imaging method, capable of detecting a large number of different objects including traditional metallic weapons, as well as non-metallic hazards such as bio-hazards and explosives [17, 18]. Unfortunately, THz spectroscopy instruments are expensive and not widely-used, mainly due to lack of commercial scaling and predominantly scientific use [16]. The alternative is often considered to be the previous-introduced Computed Tomography, which is more than sufficient for most security scenarios, limiting the development of THz spectroscopy.

2.1.4 Infrared Cameras

Infrared (IR) cameras work in the infrared ($700 - 1400\,\text{nm}$) spectrum, rather than the usual visible-light ($380 - 750\,\text{nm}$) spectrum which human eyes are perceptive to. All objects have a surface temperature above $0\,\text{K}$, and hence will emit heat radiation, which is detectable in the IR spectrum. Metallic objects usually differ in temperature to the surrounding biological matter and its derivatives due to metal's lower specific heat capacity. This makes them stand out against an organic background when being viewed through an IR camera.

It has been shown that IR cameras can successfully be used to both detect and classify concealed weapons at a distance, usually when carried on-person. However, the approach relies on there being a temperature difference between the weapon and the surrounding environment, and so works well for detecting weapons concealed on-person, but less-so for weapons hidden in luggage or other containers [19, 20].

2.1.5 mmWave Radar

mmWave radar is used in most airports in full-body scanners as an additional security check after a cruder metal detector sweep. The mmWave radar waves are bounced off the human body to give an outline of it and any metallic or solid objects being concealed on-person. These often feature a "Automatic Threat Recognition and Detection" system whereby the system will highlight possible threat areas. A security officer then determines whether a further physical search is required. The sophistication of these scanners vary from giving simple 'threat areas' to classifying the detected objects [3].

The full-body scanner is a great example of commercialised and wide-spread use of mmWave radar for weapon detection in static form, but it is often lacking in classification functionality and requires the co-operation of the individual being scanned to work properly – scanning process requires the screened individual to be still for a couple of seconds.

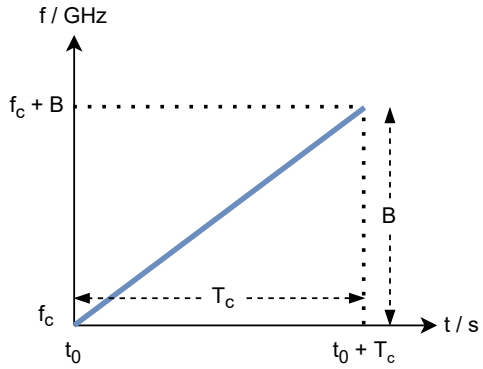


Figure 2.1: A single chirp, with start frequency f_c, signal bandwidth B, and signal duration T_c.

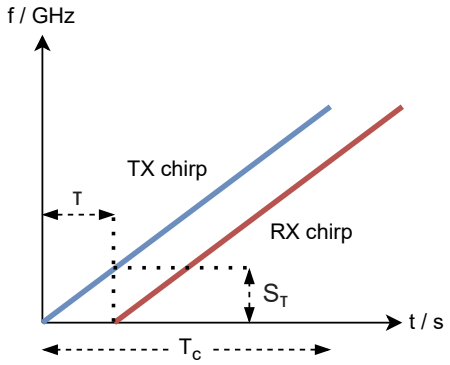


Figure 2.2: A single transmitted (TX) chirp and its corresponding received (RX) chirp, with a time delay τ and intermediate frequency S_τ.

2.2 mmWave FMCW Radar

As introduced in Chapter 1, the imaging technology used in this project will be mmWave radar.

Millimetre wave (mmWave) as a form of radar technology uses electromagnetic waves with a wavelength in the millimetre range (i.e. in the frequency range $30-300\,\text{GHz}$). This is a 'short' wavelength, which is advantageous as the required antennas to transmit and receive signals of this frequency are relatively small. The short wavelength also allow for higher accuracy in terms of range resolution. These radar systems send out signals that objects then reflect with varying absorption coefficients.

Frequency-modulated continuous wave (FMCW) is a special class of mmWave technology that differs from more traditional pulsed-radar systems. FMCW transmits a continuous frequency-modulated signal to measure both range, angle, and velocity of objects [21].

The rest of the section will cover the theory behind mmWave FMCW radar, followed by an overview of the available commercial sensors. Finally, the radar sensor's outputs will be discussed in the context of their applicability to this project.

2.2.1 Theory

FMCW radar uses a special class of signals where frequency increases linearly with time. This class of signals is called a *chirp*. These signals can be used to both measure distance and relative angle to an object.

A *chirp*, as shown in Figure 2.1, can be characterised by three main variables: a start frequency f_c, signal bandwidth B, and signal duration T_c. Common value for a Texas Instrument FMCW radar would be $f_c = 77\,\text{GHz}$, $B = 4\,\text{GHz}$, $T_c = 40\,\mu\text{s}$.

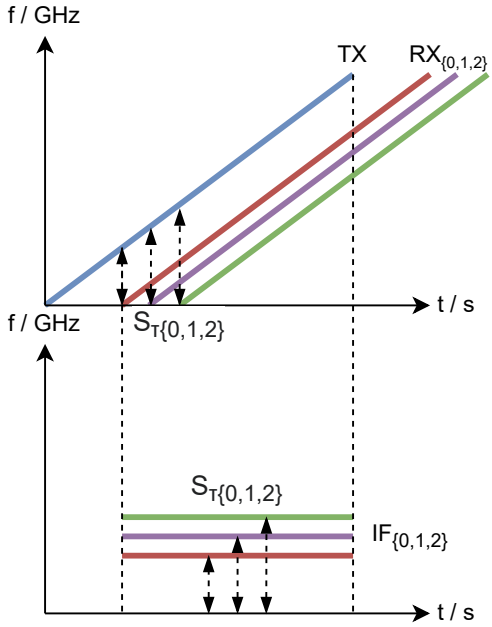


Figure 2.3: A single TX chirp with multiple RX chirps received from different objects, with respective intermediate frequencies $S_{\tau_{\{0,1,2\}}}$.

Distance Measurement

A FMCW radar continuously transmits chirps and captures the signals reflected by various objects along its path. Figure 2.2 shows such a received signal, with a time delay τ between the TX and RX chirps. τ is a function of distance d to the object reflecting the signal:

$$\tau = \frac{2d}{c} \tag{2.1}$$

where c is the speed of light. By re-arranging the equation to $d = \tau c/2$, we are able to calculate the distance to the reflecting object.

Consider Figure 2.3, where a single TX chirp results in multiple RX chirps that are reflected off different objects. The figure also shows the resulting intermediate frequency (IF) tones, which are proportional in frequency to the signal time delay $\tau_{\{0,1,2,\}}$ and hence to the distance to the objects. The received IF signal (depicted in the bottom half of Figure 2.3) consists of these multiple tones (denoted in various colours). One can separate out these tones from the combined signal via a Fourier transform. The resulting frequency spectrum will have have peaks corresponding to the presence of an object at a specific distance.

An important factor to consider is the *resolution* with which we can perform these multi-object range measurements. The *range resolution* of a radar system is the minimum distance between two objects at which they are still distinguishable as separate objects. According to Fourier transform theory (FTT), the resolution is proportional to the length of the IF signal (S_τ in Figure 2.2). This length is in turn proportional to the signal bandwidth B.

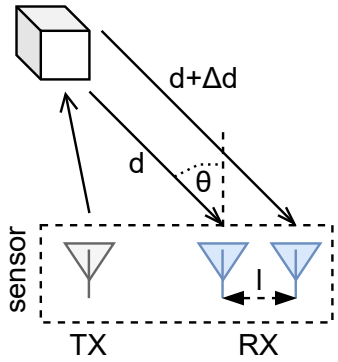


Figure 2.4: Angle-of-arrival estimation with 2 RX antennas.

FTT further states that an observation window of length T can distinguish frequency components separated by at least $\frac{1}{T}$Hz. Hence, one can resolve two IF tones as long as their $\Delta f = |f_1 - f_2|$ satisfies:

$$\Delta f = \frac{1}{T_c} \tag{2.2}$$

where T_c is the signal duration/observation interval as shown in Figure 2.1. Given the definition $\Delta f = 2S\Delta d/c$, where $S = B/T_c$ is the slope of the chirp and Δd is the distance separation of the two objects, we can show that:

$$\Delta d > \frac{c}{2ST_c} = \frac{c}{2B} \tag{2.3}$$

Hence, we have shown that the range resolution d_{res} depends solely on the bandwidth B of the chirp:

$$d_{\text{res}} = \frac{c}{2B} \tag{2.4}$$

Angle Estimation

An array of RX antennas can be used to estimate the *angle of arrival* θ of the reflected signal. This works on the principle that the difference in distance Δd between the reflecting object and the receiving antennas leads to a phase change in the received signal. Namely, the FFT peak of the received signals will have a phase offset. Figure 2.4 shows this with a two-RX-antenna array.

The phase change $\Delta\Phi$ can be derived mathematically, under the assumption that the object is sufficiently far enough that $\Delta d = l\sin(\theta)$ where l is the distance between the antennas (i.e. one can assume that the received signals are parallel to one another):

$$\Delta\Phi = \frac{2\pi\Delta d}{\lambda} \tag{2.5}$$

where λ is the wavelength of the carrier signal.

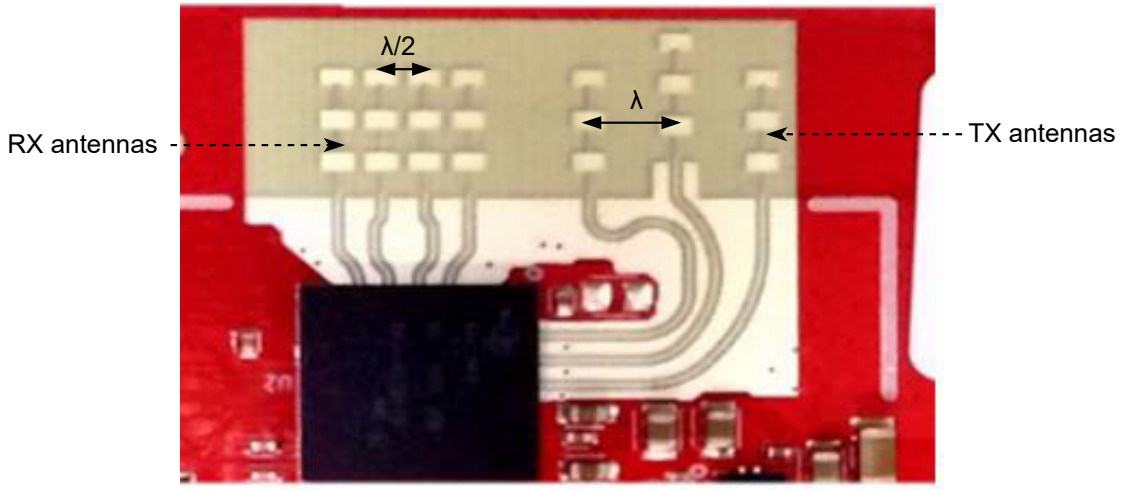


Figure 2.5: Photograph of the IWR1443BOOST's PCB, showing the physical TX and RX antennas in a 2D arrangement, allowing for 3D angle estimation. Photo provided courtesy of Texas Instruments [23].

The angle of arrival θ can hence be shown to be:

$$\theta = \sin^{-1}\left(\frac{\lambda\Delta\Phi}{2\pi l}\right) \tag{2.6}$$

Angle measurement ambiguity is introduced if $\Delta\Phi > \pi$. Hence, the maximum unambiguous angular field of view can be shown to be:

$$\Delta\Phi_{\max} = \frac{2\pi\Delta d_{\max}}{\lambda} = \frac{2\pi l \sin(\theta_{\max})}{\lambda} = \pi \tag{2.7}$$

Hence the angular field of view $\theta_{\max}$ is:

$$\theta_{\max} = \sin^{-1}\left(\frac{\lambda}{2l}\right) \tag{2.8}$$

Which implies that the angular field of view is maximised to be $\theta_{\max} = \pm\pi$ when the spacing between the antennas is $l = \lambda/2$.

The angle estimation can be performed in 3D, given a 2D antenna arrangement. Figure 2.5 shows a photograph of such a 2D arrangement present on the IWR1443BOOST evaluation board. The spacing between the horizontal antennas is $l_{\text{hor}} = \lambda/2$ and $l_{\text{vert}} \approx 2\lambda$ for the vertical antennas. This arrangement gives a vertical field of view of roughly $\pm 15°$ [22].

The theory presented in this section has been adapted from Iovescu and Rao [21].

2.2.2 Commercial Sensors

There are two main companies that produce mmWave sensors: Infineon and Texas Instruments (TI). The products are distinguished by their operational frequencies, usually 24 GHz, 60 GHz, or 77 GHz, and chirp bandwidth. Texas Instruments provides better documentation for their sensors and they are more readily available to consumers. Hence, only TI sensors will be considered from here on.

In terms of sensors for industrial (rather than automotive use), Texas Instruments produces two sensors in the $60-64$ GHz range: IWR6443 and IWR6843; as well as three in the $77-81$ GHz range: IWR1443, IWR1642, IWR1843. All 5 sensors have a bandwidth of 4 GHz. We have shown in Equation (2.4) that the imaging resolution solely depends on the available chirp bandwidth B. Hence, a chirp bandwidth of 4 GHz translates to a range resolution of roughly 3.75 cm.

Some of the sensors, such as the IWR1642 and IWR1843, have on-chip DSP. An integrated DSP module allows for implementation of high-level algorithms such as object tracking and simple classification on-chip, but significantly increases the power draw. The IWR1443 sensor lacks a DSP core but still provides high-accuracy radar measurements.

2.2.3 Sensor Outputs

The TI mmWave sensors feature a variety of different outputs. These range from the raw FFT radar signal, to a Range-Azimuth heatmap that combines angle and distance estimation as covered above.

There are a number of other outputs that are not relevant to this project, such as the Range-Doppler heatmap that tracks object speeds, and a 3D object detection output that attempts to cluster individual object heatmaps into a single point with a 3D position and velocity vector attached.

The evaluation module that will be used for this project is the IWR1443BOOST board. This dictates the specifics of the background material presented below. The justification for this choice will be discussed in Section 3.3.1.

Raw FTT Signal

The IWR1443BOOST evaluation module used does not support outputting its raw data via the default UART-to-USB connection, owing to a $1\,\mathrm{Mbit\,s^{-1}}$ limit on data transmission. This limits output data to pre-processed outputs, including the radar heatmap that will be discussed in the following sub-section.

The sensor's raw data can be accessed via its LVDS interface using an additional board such as the DCA1000EVM. Future extensions of this project would likely focus on using the full raw data to improve accuracy and performance.

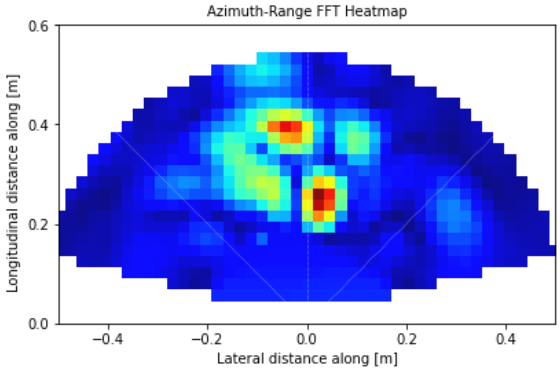


Figure 2.6: An example visualisation of a Range-Azimuth heatmap obtained from a IWR1443BOOST evaluation board, capturing data between 0.18 m and 0.53 m away from the sensor.

Radar Heatmap

There are two main heatmap types that are accessible via the on-board UART connection: the Range-Doppler and the Range-Azimuth heatmaps. Since this project is focused on scanning static objects, the Doppler coordinates of the radar scans are irrelevant. Hence, when referring to the radar heatmap from here on, I will implicitly refer to the Range-Azimuth heatmap.

The heatmap is a radar cube matrix with a zero Doppler index (i.e. only static objects), across all range bins and all antennas. It shows the intensity of the reflected signal at each physical location. It can be visualised as a 2D semi-circular sweep, with the intensities averaged across the elevation dimension [24]. An example of such a heatmap is presented in Figure 2.6. The heatmap essentially combines angle and distance estimation, as described in Section 2.2.1, into a single output.

Due to the UART-to-USB connection's bandwidth limitations, the radar heatmap output requires dropping the sensor update rate to 4 frames-per-second.

The raw data is received as a 2D FFT array in the range direction, with shape `numRangeBins` × `numVirtualAntAzim`. The `numRangeBins` is the total number of range bins, set by the configuration of the radar. A range bin's index can be directly converted into an actual range by multiplying it by a set factor also determined by the radar's configuration. For example, `numRangeBins` = 256 with `rangeIdxToMeters` = 0.0353 implies that the data at range bin 10 corresponds to a physical location roughly 0.35 m away from the sensor.

`numVirtualAntAzim` corresponds to the number of "azimuth" virtual antennas. This is usually (`numTxAnt` − 1) ∗ `numRxAnt` where `numTxAnt` is the number of transmission antennas and `numRxAnt` – the number of receiving antennas. One of the transmission atennas is used for elevation estimation and hence does not count towards the *azimuth* virtual antennas [25].

2.3 Visual Inertial Odometry (VIO)

As was described in Section 1.3, a position and orientation tracking method is required to estimate the physical pose of the radar sensor. This is necessary to be able to combine an array of individual radar heatmaps into a single 3D radar scan. The method of choice was Visual Inertial Odometry. This choice is justified in Section 3.3.2.

Visual Inertial Odometry (VIO) is a method of enhancing the accuracy of visual odometry (VO) using an inertial measurement unit (IMU). In turn, VO is the process of determining the 3D position and orientation of an object by virtue of analysing its associated camera images.

VO often works via feature-based methods. An image's feature points, i.e. distinguishing features such as points of high contrast, sudden colour change, or distinct patterns, would be extracted and tracked on a frame-by-frame basis [26]. One can also apply a "direct" method which track pixel intensity values inter-frame instead [27].

An IMU sensor on its own can be used to estimate position and orientation. The orientation is often relatively accurate if using a dedicate gyroscope, but the position estimation often suffers from high error rates introduced during the double integral of converting between acceleration (provided by an accelerometer) and displacement [28].

Combining VO and IMU data allows one to minimise the error rates, providing incredibly accurate tracking capabilities [29]. These technologies are actively being used in practice and are accessibly via a simple API call to either Apple's ARKit or Google's ARCore [30]. These APIs apply the feature extraction methods described above to the phone's camera output and combine the position and orientation data estimated with the data read from the phone's IMU sensors – usually a gyroscope and accelerometer. This provides accurate pose estimates, as will be subsequently verified in Section 6.1.1.

2.4 Deep Learning

The 3D radar scans obtained via the method proposed in Section 1.3 need to be classified to determine the nature of the concealed object. A deep learning approach via 3DCNNs is used for this project due to recent developments in lightweight and efficient networks [31]. A further discussion on the choice of classification algorithm is discussed in Section 3.3.3.

The rest of this section is going to give a gentle introduction to neural networks, starting with the theory behind generic supervised learning. Basic neural networks are then going to be discussed, followed by an introduction to convolutional neural networks as a means of processing multi-dimensional data. Finally, the problem of overfitting to training data, as a result of the combination of complex models and small datasets, will be discussed and a few potential solutions outlined.

2.4.1 Supervised Learning

Supervised learning can be defined as the process of learning some function $f : x \to y$ that takes an input x and assigns it a class label y:

$$f(\mathbf{x}) = y \tag{2.9}$$

$\mathbf{x} \in \mathbb{R}^N$ is usually taken to be some real-valued N-dimensional vector, and $y \in \mathbb{Z}$ is an integer in some range $0 \leq y \leq M$ which maps onto the desired class label [32]. Binary classification is a special subset of supervised learning where $M = 2$.

A *training dataset* $\mathbb{D}_T$, made up of input-output pairs $(\mathbf{x}, y) \in \mathbb{D}_T$, is used to learn f. The learning process consists of repeatedly sampling from $\mathbb{D}_T$ and comparing the resulting predicted output $f(\mathbf{x}) = \hat{y}$ to the true label y. However, rather than directly predicting $\hat{y}$, it is often more informative to learn a function g that outputs the likelihood $p^{\mathbf{x}}_{y'}$ of an input $\mathbf{x}$ belonging to a class y':

$$g(\mathbf{x}) = (p^{\mathbf{x}}_0, p^{\mathbf{x}}_1, \ldots, p^{\mathbf{x}}_{M-1}) = \hat{\mathbf{y}} \tag{2.10}$$

The classification function f is then taken to be:

$$f(\mathbf{x}) = \text{argmax}(g(\mathbf{x})) = \text{argmax}(\hat{\mathbf{y}}) = \hat{y} \tag{2.11}$$

The comparison between the predicted $\hat{y}$ and true y is performed numerically via a *loss* function, whereby the model's training goal would be to minimise this resulting loss. In the case of an M-class (multiclass) classification problem, *cross entropy loss* is often used:

$$\mathcal{L}(\hat{\mathbf{y}}, y) = -\sum_{m=0}^{M} \mathbb{1}_{m=y} \log(\hat{\mathbf{y}}_m) = -\sum_{m=0}^{M} \mathbb{1}_{m=y} \log(p^{\mathbf{x}}_m) \tag{2.12}$$

where $\mathbb{1}_{m=y}$ is an indicator function that returns 1 if $m = y$ and 0 otherwise [33].

2.4.2 Neural Networks

Neural networks (NNs) are common models used for handling classification tasks. A simple, single-layer NN is also often called a *perceptron*. A perceptron f_p has two main variables: a weight vector $\mathbf{w}$ and a scalar bias term b [34]. Perceptrons can be chained together to form multi-layer perceptrons (MLPs) to improve their classification abilities. Binary classification is then performed on some vector input $\mathbf{x}$ as follows:

$$f_p(\mathbf{x}) = \mathbb{1}_{\mathbf{w}\cdot\mathbf{x}+b>0} \tag{2.13}$$

where $\mathbf{w} \cdot \mathbf{x}$ is the dot product $\sum_{i=0}^{n} w_i x_i$, given $|\mathbf{w}| = |\mathbf{x}| = n$.

Alternatively, rather than using a step-function such as the one shown in Equation (2.13), one can use a generic *activation* function g_{act}:

$$f_p(\mathbf{x}) = g_{\text{act}}(\mathbf{w} \cdot \mathbf{x} + b) \tag{2.14}$$

Common activation functions include *sigmoid*:

$$g_{\text{act}}^{\text{sigmoid}}(z) = \frac{1}{1 - e^z} \tag{2.15}$$

tanh:

$$g_{\text{act}}^{\tanh}(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{2.16}$$

and Rectified Linear Unit (ReLU) [35]:

$$g_{\text{act}}^{\text{ReLU}}(z) = z^+ = \max(0, z) \tag{2.17}$$

Such activation functions are essential as they allow multi-layer NNs to solve non-trivial problems by introducing non-linearities into the network [36].

2.4.3 Training

As mentioned in the previous subsection, each node/perceptron has two main variables associated with it: a weight vector $\mathbf{w}$ and a scalar bias term b. These variables need to be tuned to improve the classification ability of the neural network, i.e. to minimise the previously-defined *loss*.

This is nearly ubiquitously done via *back-propagation*. This involves computing the gradient of the loss with respect to the weights and biases of the neural network for a single input-output pair $(\mathbf{x}, y) \in \mathbb{D}_T$. The main innovation of this methods stems from its efficiency: the gradient of the loss function with respect to each weight is computed via the chain rule one layer at a time. This process starts from the last output layer of the NN and iterates backward to avoid unnecessarily calculating the intermediate terms in the chain rule. This efficiency allows the use of gradient-based methods for training deep (multi-layer) neural networks [37].

One of the most commonly-used gradient-based methods is Stochastic Gradient Descent (SGD). A standard gradient descent method would optimise the neural network's weights w according to the following formula:

$$w_{t+1} = w_t - \eta \nabla Q(w) = w_t - \frac{\eta}{n} \sum_{i=1}^{n} \nabla Q_i(w) \tag{2.18}$$

where n is the size of the training dataset, ν is the learning rate, $Q(w)$ is the empirical risk, and $Q_i(w)$ is the *loss* for sample $\mathbf{x}_i$.

While evaluating the whole sum is feasible for smaller datasets, in practice it is often too computationally expensive due to large training dataset size and non-trivial gradient calculation. Hence, SGD approximates the true $\nabla Q(w)$ as the gradient $\nabla Q_i(w)$ of a single sample $\mathbf{x}_i$:

$$w_{t+1} = w_t - \eta \nabla Q_i(w) \tag{2.19}$$

The training dataset would be randomly shuffled and the simple-sample gradient step shown in Equation (2.19) would be performed iteratively for each data sample.

In practice, mini-batch SGD is more common. Here, the true $\nabla Q(w)$ is approximated as:

$$\nabla Q(w) \approx \sum_{i=1}^{m} \nabla Q_i(w) \tag{2.20}$$

where $m < n$ is the mini-batch size. m is often taken to be a power of 2 to improve data retrieval efficiency [38].

2.4.4 Convolutional Neural Networks

While neural networks in the form discussed above work well for one-dimensional data, there is a lot of value to being able to work with two-dimensional data such as images. Traditionally, MLPs were used for classification of 2D images, but due to the curse of dimensionality their application was limited.

Convolutional neural networks (CNNs) employ N-dimensional kernels to efficiently analyse N-dimensional data. While CNNs are also applicable to 3D input data (which is important given that this project needs to classify 3D radar scans), for the sake of simplifying the mathematics let us consider the $N = 2$ case. Kernels are usually small (3×3 or 5×5) learnable matrices. The size of the kernel used dictates the size of the image feature it can detect. The learned kernels are able perform feature detection with translation invariance, which makes CNNs incredibly powerful at performing image classification.

During the forward pass of the CNN, the kernel k is convolved with the input to produce an activation map g:

$$g(x, y) = k \star f(x, y) = \sum_{dx=0}^{w-1} \sum_{dy=0}^{h-1} k_{dx,dy} f(x + dx, y + dy) \tag{2.21}$$

where $f(x, y)$ is the pixel value at the respective co-ordinates of the input image, $g(x, y)$ is the resulting activation map value for those co-ordinates, and w, h are the dimensions of the kernel k.

The size of the resulting activation map depends on the length of the kernel's *stride*. This dictates by how much the kernel window moves over the input image at each step. A larger stride is often used for computational efficiency or to downsample. A stride of > 1 is often preferred for larger kernels as they capture signification portions of the underlying image [39].

A lot of the base CNN models used in this project have large strides as they are designed to deal with high-resolution images. Since the 3D radar scans produced are just $16 \times 16 \times 16$, a large stride and kernel size would lead to very coarse activation maps, which would negatively affect classification performance. As such, as will be discussed in Section 5.3.4, the strides and kernel sizes of the base models used were adjusted to suit the smaller data dimensions.

2.4.5 Overfitting

The models that will be used in this project are designed to be trained on large datasets with millions of examples to hone their classification abilities. However, in the context of this project, it is unfeasible to collect and process a large dataset due to time constraints. As such, the models become susceptible to *overfitting*.

Overfitting is refers to the phenomenon where a model fits the training data almost exactly, to a point of deteriorating performance on unseen data. An overfitted model's performance generalises poorly to unseen data. This usually occurs when a model is too powerful, in terms of complexity or number of learnable parameters, compared to the dataset that is being fitted to [40].

There are a number of ways to prevent overfitting. A simple way of doing so is by increasing the amount of training data. It was not possible to collect enough unique data for this project due to time constraints, but the collected data was augmented to generate more of it without significant loss of diversity. This process is described in Section 4.3. However, this approach alone is still not sufficient to prevent overfitting due to the limited underlying data. Hence, two additional approaches can be taken:

1. *Early stopping* – the model's performance is periodically evaluated on a validation dataset entirely disjoint from the training set. Models with the best validation set performance are saved and training stopped when the validation accuracy begins deteriorating. This stops training before the model begins to memorise (rather than learn) the training set.

2. *Dropout* – during training, a Dropout layer can be used to randomly ignore ("drop out") the output of the preceding layer. This regularisation technique is effective at reducing overfitting in NNs [41].

As will be discussed in Section 5.3.4, these two approaches were successful in improving model generalisation.

Chapter 3

Scanner Design

This chapter discusses the various design decisions that went into the the proposed method described in Section 1.3 and into designing the prototype radar scanner.

The chapter starts with Section 3.1 defining the problem statement that this project aims to tackle. The scope of the project is then derived from this problem statement in Section 3.2, followed by a requirements analysis of the project in Section 3.3. The requirements analysis weights up the pros and cons of the design decisions in the face of the problem statement. Specifically, the choice of mmWave sensor, position tracking method, and classification method are discussed. The chapter then concludes with Section 3.4 giving an overview of the final design.

3.1 Problem Statement

Existing methods of weapon detection, as described in Chapter 2, often rely on stationary and difficult-to-transport equipment and require the co-operation of the party being imaged. Hence, there is a niche to be filled by using a *portable* device. A portable device would be flexible in the way it can be used – it could can objects of varying shapes and sizes, which would be an improvement over imaging methods such as CT scans and X-Ray machines as they have a size limit dictated by the physical dimensions of the scanner.

Existing methods of weapon detection also often have significant power consumption requirements, needing to rely on a stable power supply. Hence, the scanner would benefit from a *low-power* radar module. I will define low-power to mean a power draw of below $5\,\mathrm{V}/3\,\mathrm{A}$. This is the Quick Charge 2.0 standard and can be met by many portable USB power banks. The standard capacity for such a power bank is roughly $10 - 20\,\mathrm{A\,h}$, which would be able to power such a device for at least $3 - 6$ hours.

Setting a *low-cost* requirement means that the method should use as many off-the-shelf components as possible. As discussed later, this meant that the final design uses an everyday mid-range Android phone for positioning purposes. The low-cost nature of the device increases its availability, which especially important for the motivating example given in Section 1.1, i.e. in wildlife preservation and anti-poaching programmes that are usually under- or poorly funded [9].

3.2 Project Scope

The most important challenge that this project needs to solve is to be able to *detect and classify concealed metallic objects*. To demonstrate these classification abilities, three household objects were selected. Namely, these were chosen to be: *a phone*, *a knife*, and *a screwdriver*. The objects are sufficiently different enough and pose an interest for detection: a knife and a screwdriver can be used as weapons, and a phone, if found under suspicious circumstances, can be searched to provide further context of the threat.

These objects have similar size measurements in certain dimensions – they are all around the same thickness for example; which means that the object scanning needs to be done in 3D to account for the full shape of the objects. While radar sensors perform a basic depth map, most are usually limited by the elevation angle range and the penetrative abilities of the radar. This means that objects need to be scanned from multiple points-of-view to obtain a complete picture. This, in turn, implies that one needs to know the position and orientation of the radar sensor, relative to the object being scanned.

This gives rise to three main design decisions: the choice of *mmWave sensor*, the way of performing *position tracking* of the sensor, and how to perform object *classification*.

3.3 Requirements Analysis

The design decisions presented in Section 3.2 can be analysed via the requirements imposed by the *Problem Statement*: portability, power, and cost. These requirements guide the design decisions.

3.3.1 mmWave Sensor

As mentioned in Chapter 2, Texas Instruments produces five sensors for industrial use, of varying operating frequencies: IWR6443, IWR6843, IWR1443, IWR1642, IWR1843. A single sensor needs to be selected to construct the prototype. These five sensors will be scrutinised in terms of scan resolution, power requirements, price, and availability.

Scan Resolution

The scan resolution of the sensor needs to be high enough to tell objects apart when they are imaged with this resolution. The three selected objects are all roughly the same thickness, but vary in size in other dimensions: a phone is roughly as long as a household screwdriver but is significantly wider; while a screwdriver and a knife are roughly the same width but have different lengths.

As such, the imaging resolution needs to be at least equal to the minimum difference in dimensionality between these objects. In the case of the objects used for the experiments, detailed in Section 4.2.1, this was found to be 4.5 cm. This constituted the difference between the length of the screwdriver and knife, as shown in Figure 3.1. The second largest difference was between the width of the phone and knife at 5.4 cm. Hence, the scan resolution needs to be at least 4.5 cm.

Figure 3.1: Size comparison of the objects to be classified.

As discussed in Section 2.2, TI produces sensors that operate in both the $60-64$ GHz and $77-81$ GHz frequency ranges. Equation (2.4) shows that the imaging resolution depends exclusively on the available chirp bandwidth. All the sensors have the same chirp bandwidth of 4 GHz which translates to a range resolution of roughly 3.75 cm. The aforementioned scan resolution requirement of 4.5 cm requires only a bandwidth of 3.33 GHz. Hence, all the TI sensors have a sufficiently large bandwidth to meet the scan resolution requirement. The resolution is slightly lower in practice ($\sim$ 4 cm), as discussed in Section 4.1.1, but it still sufficiently high to meet the resolution requirement.

The sensors differ in operating frequency, which is relevant only when considering the regulations surrounding radar usage. These regulations differ depending on the country and application (e.g. automotive vs not). However, since the project is a prototype, it does not need to meet these regulations. As the resolution across the selection is the same, the choice lies with the most available and lowest power-consuming sensor.

Power, Price, and Availability

All the TI evaluation boards use a 5 V/2.5 A power supply, which meets the low-power requirements as set out in Section 3.1. However, the IWR1642, IWR1843, and IWR6843 sensors have an on-chip digital signal processor (DSP) which significantly increases the power draw [42]. The method utilises the Range-Azimuth heatmap output of the sensor, making the DSP unnecessary for this project.

This leaves just the IWR1443 and the IWR6443 sensors under consideration. The IWR6443 sensor does not have an evaluation board available, which greatly increase the difficulty and scope of this project as one would have to design a board to make the chip usable. The *IWR1443* is usable out-of-the-box via the IWR1443BOOST evaluation board and is hence the sensor of choice in this case.

The IWR1443BOOST evaluation board also retails at 358.80 USD, making it relatively affordable, especially compared to higher-end radar modules such as the MMWCAS-RF-EVM which retails at 1,318.80 USD.

Final Decision

The final decision was made to use the IWR1443 sensor via the IWR1443BOOST evaluation board, for the reasons discussed above. This board was purchased via University of Oxford's Computer Science Department.

3.3.2 Position Tracking

To combine multiple 2D radar heatmaps into a single 3D image, one needs to know the position and orientation of the radar at the time each 2D heatmap was recorded. There are multiple ways of doing so:

- Wi-Fi-based indoor positioning systems work by time-of-flight multilateration from multiple Wi-Fi hotspots. The distance estimation between the device and a hotspot, needed for positioning, is based on signal strength, which varies heavily depending on environment conditions, obstacles, and even device orientation. These approaches often need to rely on neural networks to achieve reasonable accuracies, though more traditional approaches can give room-level granularity reliably.

  This approach is not ideal as it does not give the device's orientation, and even the best methods have an error rate of around 0.22 m in an ideal environment. The setup also relies on setting up hotspots around the tracking location, which makes this approach unrealistic for use outside of a controlled space, such as an office [43]. The Bat System, described by Ward et al. [44], takes a similar approach but uses a combination of MHz radio for clock synchronisation purposes and ultrasonic pulses for time-of-flight estimation. It too is only usable in a controlled indoor space.

- Photogrammetry-based approaches rely on constructing a model of the tracking environment based on a large number of images of the scene. This is labour-intensive to construct, but the end result is a 3D model of the scene that can be used in tandem with a single photo (of the scene) to accurately predict both the 3D position and orientation of the camera when the photo was taken.

  Ghofrani et al. [45] propose such a method that achieves remarkable accuracy in both position and orientation estimation: the largest error rate for positioning was a mere 3.4 mm, and 0.0086° for orientation. This approach however is not very practical for our use-case as it relies on the pre-built environment model.

- Visual-Inertial odometry (VIO), as described in Section 2.3, has been used for various Augmented Reality (AR) and indoor tracking purposes. The approach is able to produce very accurate and stable results, with exact error-rates differing significantly depending on the device and specific method used. VIO can be performed is entirely on-device, without external installations or beacons [46]. VIO is also used in VR headsets such as the Oculus Quest for position and orientation tracking, however using a multi-camera setup to provide millimetre-level tracking accuracy. The error-rates for the constructed prototype were experimentally determined. These results are discussed in Section 6.1.1.

The first two approaches rely on external equipment to provide the positioning data, which makes them impractical for a *portable* setup. Hence, VIO is selected to be the position tracking method of choice.

VIO data can be obtained both via a smartphone using simple API calls on both Android (via Google's ARCore [30]) and on iOS (via Apple's ARKit); or on a custom-built device featuring an RGB camera(s) and IMU sensor. While the latter may be more cost-efficient in the long-run and more accurate, using a smartphone is easier for prototyping and they have been getting progressively cheaper and are widely available [47]. This makes VIO accessible and easily the most favourable approach, owing for both its high accuracy and ease of use. The Android ecosystem and its respective ARCore [30] library was the tracking solution of choice, due to ease of development and cheaper average prices compared to iPhones.

3.3.3 Classification

Classifying the obtained 3D radar images can be done both via a classical data science approach through feature selection or via an ML model. A feature-selection approach would involve manually determining which features of the data would be most informative to making the classification decisions. Once these features are selected, the model would be trained to determine the exact threshold values of these features to guide the classification process.

For example, one could select the dimensions of voxels with an above-mean intensity value as a feature of the image, i.e. filter out the data points with an intensity value below the mean and calculate the 3D shape of the resulting object (clustering if needs be). Alternatively, a feature could be the maximum intensity value in the image. This feature-based approach requires hand-crafting these features, which may reduce the generalisability properties of the classification model. It is also more labour intensive of an approach.

The alternative is to use neural networks, and specifically the CNN models described in Section 2.4.4. An ML approach is more hands-off and generally gives better generalisation properties. There have also been recent advances in efficient 3D classification networks which significantly speed up training and inference [31].

The 3D nature of the data used in this project means it is unfeasible to manually derive useful features. For example, the shape feature described above is not rotation-invariant, which reduces its usability outside of a controller scanning environment. Resource-efficient 3D CNNs seem like the most promising direction to go down in terms of classification, due to their accurate performance and lightweight nature. Classification performance will be evaluated on different model architectures.

3.4 Design Overview

Figure 3.2 gives a diagrammatic overview of the final prototype scanner design. The idea behind the project is to able to construct a 3D radar scan of a solid metallic object by combining multiple 2D radar scans taken from known positions and orientations. The 2D radar scans are obtained via the mmWave IWR1443BOOST evaluation board's heatmap read-out. The 3D position and orientation of the evaluation board is tracked using Visual-Inertial Odometry (VIO) data collected on an Android phone using Google's ARCore library [30]. These 3D radar scans roughly correspond to the shape of the object, with the intensities at each co-ordinate indicating the reflectance properties of the surface at that co-ordinate. 3DCNN neural network models are trained to classify the obtained 3D radar scans into predetermined object classes: phone, knife, and screwdriver.

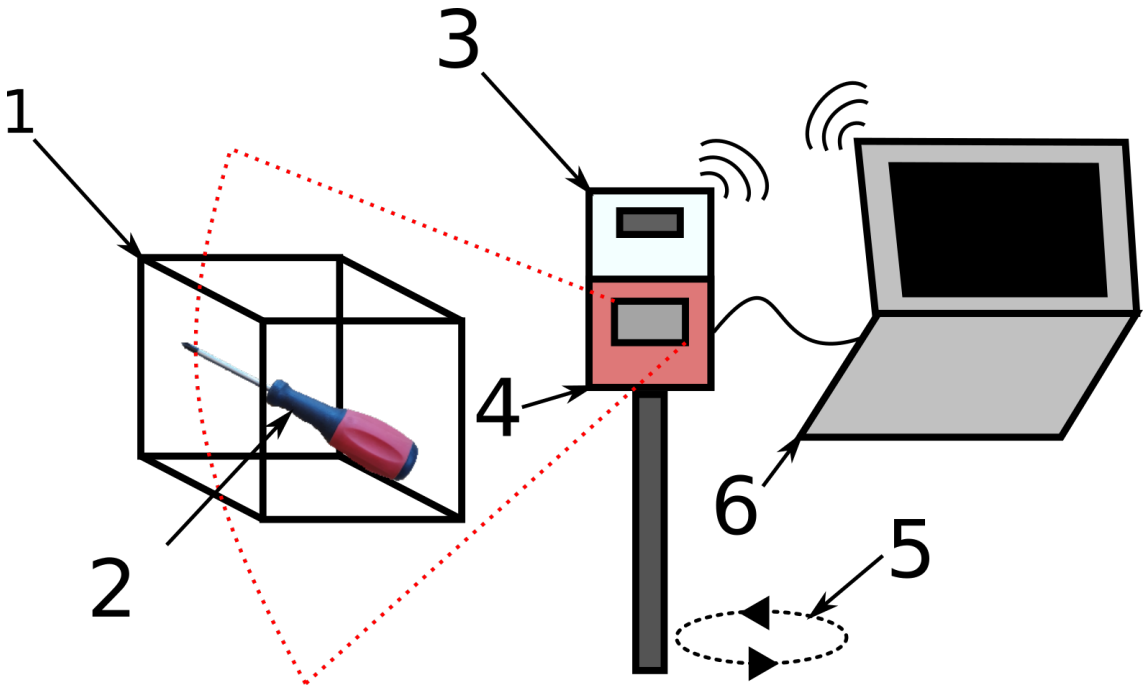


Figure 3.2: Diagrammatic overview of the various components of the prototype scanner. `(1)` is a solid box used to obscure an object `(2)` that is meant to be scanned and classified. Figures 4.8 and 4.9 show real-world photo examples of `(1,2)`. `(3)` and `(4)` together form the contraption used for producing 3D radar scans, with Figure 5.2 depicting a real-world photo of this device. `(3)` is an Android phone that is used for capturing Visual-Inertial Odometry (VIO) data. `(4)` is a mmWave radar module, namely, the IWR1443BOOST evaluation board. `(5)` depicts in abstract form the process of scanning an object, described in detail in Section 4.2.2. Finally, `(6)` is a laptop or PC that receives data from the mmWave radar module via a USB cable and the VIO data wireless from the Android phone. This PC is also responsible for running the data post-processing and machine learning pipeline, described in Section 4.3.

Chapter 4

Methodology

This chapter describes the various algorithms used in the method proposed in Section 1.3. The chapter also describes the experimental procedures used to test the capabilities of the produced prototype.

There are three sections to this chapter. Section 4.1 gives the detailed methodology behind producing 3D radar scans of concealed objects. Section 4.2 details the approach taken to conducting the data-collection experiments. Section 4.3 details the data post-processing procedures as well as the data-centric machine learning techniques employed to classify the obtained 3D radar scans.

4.1 3D Radar Scan

This section describes the procedure behind obtaining 3D radar scans of concealed objects. The scanning method relies on a IWR1443BOOST evaluation board for obtaining the radar measurements, and an Android phone for the Visual-Inertial Odometry (VIO).

4.1.1 Reading Data

The first step in producing a radar scan is to be able to interact with and read data from the evaluation board. Python was used as the main programming language due to the wide availability of data processing libraries.

The board connects to a computer using a UART-to-USB connection via a micro-USB port on the sensor. The single USB connection exposes two main serial ports: the data port and command-line interface (CLI). These have different software paths and baud rates. Python's `serial` library was used to communicate with the board.

After opening the two serial ports, the board's configuration file needs to be read-in and sent to the device via the CLI port. The configuration file is also parsed into a usable dictionary format to allow for access to the configuration variables to ease data decoding.

The file specifies the operating frequency of the radar, the range resolution, maximum unambiguous range, the output data types, and other settings. The file is generated using TI's online mmWave Demo Visualiser[1] tool. Deviations from the default configuration of the radar were as follows:

- All 3 TX antennas (as opposed to the default 2) were enabled to allow for elevation estimation, as described in Section 2.2.1.
- Range resolution was maximised by being set to its lowest value of 0.04 m.
- The Range-Azimuth heatmap output was enabled, and all other outputs disabled.
- The output rate of the sensor was dropped to 4 frames-per-second due to the heatmap's large data size and the data port's baud rate limitation.

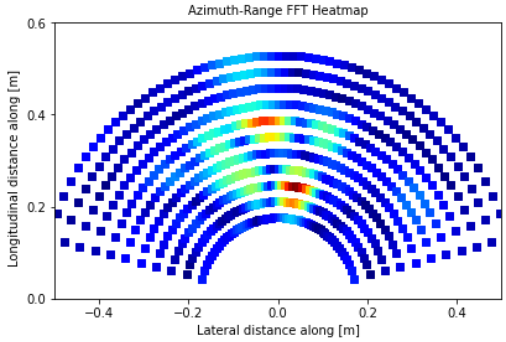
Enabling the heatmap output requires the sensor to be reset to apply these changes, and then have the settings resent. This reset needs to be done by physically depressing an on-board button, so a user prompt was written in the main program, described in Section 5.3, to ensure this is done.

Starting and stopping the radar scan was done by sending a respective `sensorStart` or `sensorStop` message on the CLI port. Once started, data was being outputted via the data port. This port was periodically read from and a data buffer was built up. Within this buffer, the start of a packet could be identified by looking for a *magic word*, which is a specific 8-element array of `uint8`s.

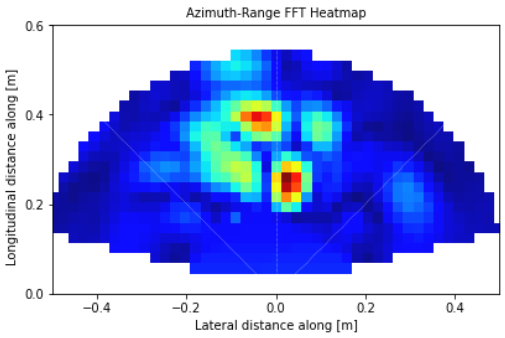
Once the start of a packet was identified, its expected packet length was decoded to ensure that the whole packet could be processed in one go. Otherwise, the program would wait for the remainder of the packet to be read-in before continuing. Once the whole packet was contained within the data buffer, its frame header was read. This contains the protocol version number, the expected packet length, the sensor type, as well a frame number and a relative timestamp. It also contains the number of TLVs (type-length-value) that the packet contains. The types of TLVs included in the packet are set by the configuration file and include the range-azimuth heatmap output. Other unused TLV types include detected points, range profile, noise floor profile, range-doppler heatmap, and performance and temperature statistics.

Each TLV in turn has its own header which identifies its type and length. The TLV type determines which parsing function to use. The parsing function takes the data portion of the TLV packet as well as the sensor configuration file as input. I chose to only decode the range-azimuth heatmap TLV as it was the only relevant one for the project.

[1] `https://dev.ti.com/gallery/view/mmwave/mmWave_Demo_Visualizer/ver/2.1.0/`



(a) An example of a raw radar heatmap.



(b) (a)'s radar heatmap with linear interpolation.

Figure 4.1: An example radar heatmap obtained from the evaluation board.

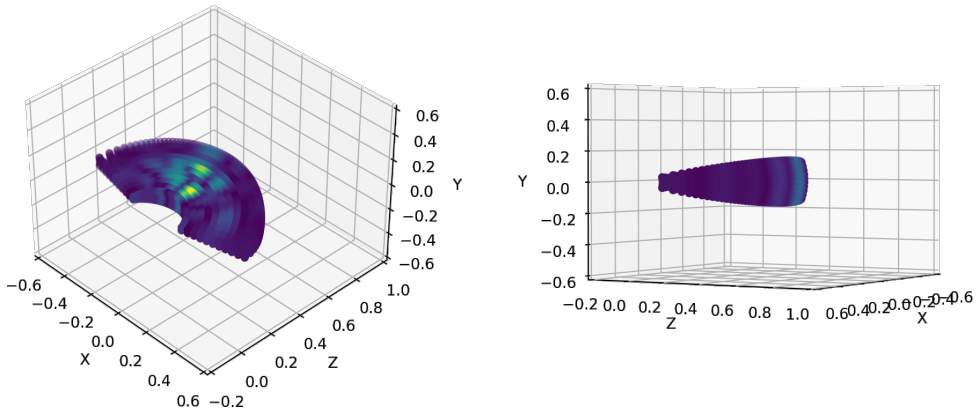


Figure 4.2: Two perspectives of Figure 4.1's radar heatmap presented in 3D space, given the evaluation board's ± 15 deg elevation angle of detection. The 2D heatmap contains the average intensities within this elevation range, so the heatmap data is simply duplicated while being pivoted around the origin to obtain the 3D scan.

4.1.2 2D Radar Heatmap

As mentioned in Section 2.2.3, the Range-Azimuth heatmap in raw format is a 2D FFT array in the range direction, with shape `numRangeBins` $\times$ `numVirtualAntAzim`, where `numRangeBins` is the total number of range bins, and `numVirtualAntAzim` is the number of "azimuth" virtual antennas used, both set by the configuration. The complex numbers are presented as pairs of real and imaginary `shorts` (`int16`). Technical references by Texas Instruments [25] and the sensor's documentation were used as reference for parsing the raw data.

The resulting heatmap had a range of roughly 8 m, which was far beyond what was required for the project. Hence, the final FFT data was cropped to match a predetermined scan range. This was chosen to be $0.18 - 0.53$ m (inclusive) to both avoid radar noise that appears closer to the sensor and to ignore potential background objects. During the imaging process, the sensor was kept within this range of the object being scanned. The distance felt natural while conducting the experiments. Figure 4.1 shows an example of the processed range-azimuth heatmap.

The algorithm for decoding the Range-Azimuth heatmap is best described via the produced code. It is very readable, with some knowledge of Python's `numpy`:

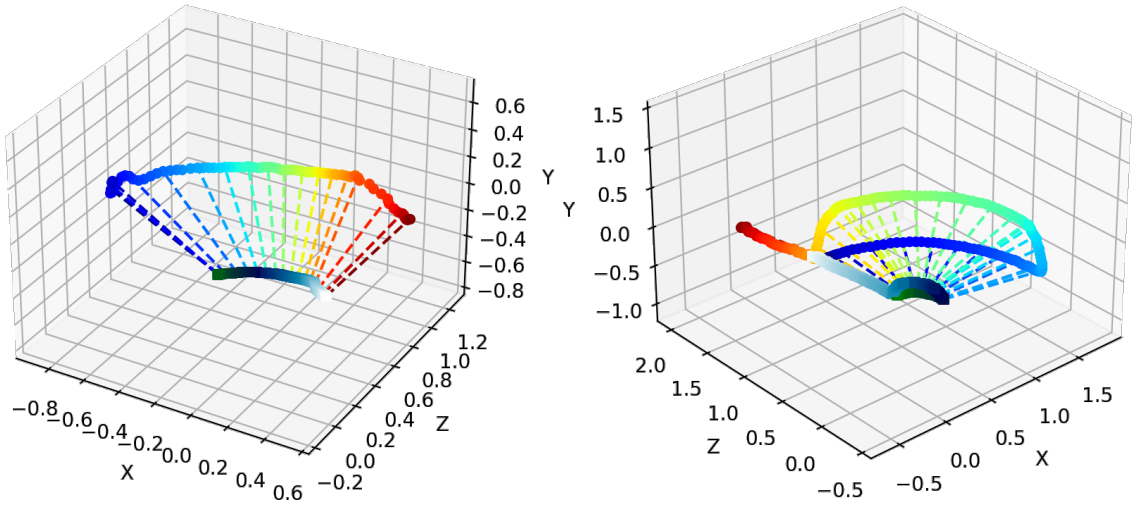
```
def azimuth_heatmap(header, data, config):
    # number of virtual azimuth antenna is (txAnt-1) * rxAnt
    num_virtual_antenna = (config['numTxAnt'] - 1) * config['numRxAnt']

    # 2 bytes per real/im short
    n = data.size // 2
    # unpack as an n-long little-endian list of shorts
    a = struct.unpack(f"<{n}h", data)
    # form array of complex numbers
    a = np.array([complex(a[i], a[i+1]) for i in range (0, len(a), 2)])
    a = a.reshape(config['numRangeBins'], num_virtual_antenna)

    # do FFT transform + shift
    a = np.fft.fft(a, config['angleBins'])
    a = np.abs(a)
    # put left to centre, put centre to right
    a = np.fft.fftshift(a, axes=(1,))
    # cut off first angle bin
    a = a[:,1:]

    # rotate 180 degrees to get correct perspective
    a = np.flip(a, axis=1)

    # crop to scan range
    sr = config['scan_range']
    return a[sr[0]:sr[1]+1,:]
```

4.1.3 3D Radar Heatmap

The 2D heatmap obtained from the sensor is post-processed 3D range-azimuth data. As discussed in Section 2.2.1, the IWR1443BOOST board contains an antenna for estimating elevation, with field of view of roughly $\pm 15°$. Due to the UART-to-USB bandwidth limitations, the raw radar data is inaccessible. Hence, in its transmitted form each data point in the 2D heatmap is actually the average intensity within the elevation angle of detection [24].

There is information loss here, so when constructing a 3D heatmap from the 2D one, there are few alternatives other than to duplicate the obtained 2D heatmap repeatedly. The vertical sweep range of the scan is taken to be in the range -15 to $15°$ to cover the field-of-view. A fixed number of 10 angles are selected within this range[2], including the extremities, and the 2D heatmap is rotated around its central origin in the vertical direction to align it with these angles. The result is a solid 3D 'slice of pie' that represents the approximate 3D range-azimuth radar data produced by the sensor. Figure 4.2 shows such a 3D heatmap from two perspectives.

[2]Via `np.linspace(-angular_range, angular_range, 10)`



(a) A horizontal pan from left to right.

(b) A longer recording, first panning from left to right, then panning up and back to the original angle, followed by a forward motion.

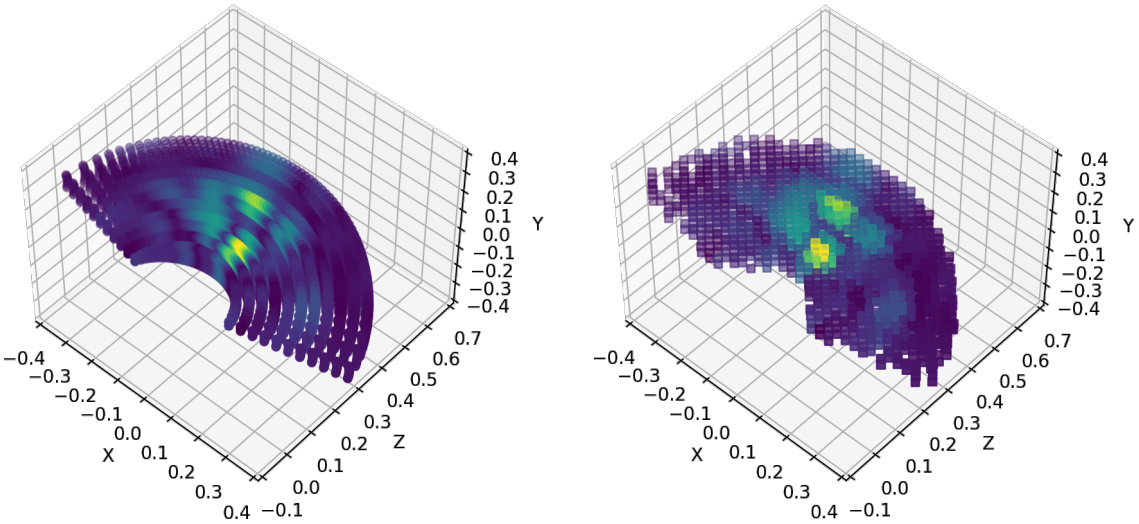
Figure 4.3: Visualising the recorded VIO data. The location of the phone is visualised as square markers, with a colour gradient[a] turning from white (at the recording start time) to blue to green (at the recording finish time) to indicate the associated timestamp. The rotation of the phone is visualised as the rainbow-coloured rays emanating from the location markers. These rays are also colour-coded with respect to the associated timestamp, starting with blue and ending with deep red[b].
[a]matplotlib's `ocean` colour map [b]matplotlib's `jet` colour map

4.1.4 Visual-Inertial Odometry (VIO)

The position and orientation tracking of the radar was done via Visual-Inertial Odometry (VIO), as discussed in Section 3.3.2. Google provides an easy-to-use API to achieve this via their ARCore library [30].

ARCore handles the position and orientation tracking automatically, storing these transforms in a `Pose` object. `Pose` describes a rigid transformation from one coordinate space to another. Specifically, it describes the transformation of an object's local coordinate space into world coordinate space. This transformation can be defined as a quaternion rotation (orientation) followed by a translation around the origin (position).

The application designed to extract this data is described in Section 5.3.1. The final UI of the Android application can be seen in Figure 5.3. An example of data collected via the application is presented in Figure 4.3, with the tracking accuracy being verified in Section 6.1.1.



(a) The original 3D radar heatmap as described in Figure 4.2.

(b) The voxelised version of the 3D heatmap presented in (a).

Figure 4.4: The effect of voxelising at a resolution of 32 voxels per meter (i.e. a voxel's side is 3 cm). As described in Section 4.1.5, max pooling is used to determine which intensity value to use at each voxel.

4.1.5 Voxelisation

As can be seen in Figure 4.2, the resulting 3D heatmap is dense and appears to be continuous. While this makes for easier visualisation, it makes working with the resulting 3D heatmap computationally expensive due to the large amount of data points. As such, I employ a down-sampling technique I call *voxelisation*.

This reduces the amount of data points in the 3D scan by pooling together intensity values from a number of data points. Another way of thinking about this technique is to liken it to rasterisation of a 2D vector image, but in 3D (hence voxels rather than pixels). This technique saves space, but also turns the 3D scan into a 3D *image* with concrete dimensionality, which is necessary as a pre-processing step before ML classification and training.

Max pooling is employed for down-sampling – the *maximum* intensity value is taken from a pre-defined area of space to represent that space as a voxel. The voxelisation resolution was taken to be 32 voxels per meter, making a single voxel's side roughly 3 cm. This scale was chosen with the radar's hypothetical range resolution of 3.75 cm in mind. Figure 4.4 shows the effect of voxelising a 3D heatmap at this resolution.

The voxelisation algorithm can be summarised with the following code, which is part of a larger class in the final implementation:

```
# xyz is a list of 3D co-ordinates with shape [n, 3]
# hm is a list of intensity values with shape [n]
def voxelise(xyz, hm):
    # quantise floats to a given voxel resolution (32)
    xyz_v = np.floor(xyz * VOXEL_RES).astype('int')

    # create a list of unique indices
    unique_xyz = np.unique(xyz_v, axis=0)
    unique_hm = np.zeros(unique_xyz.shape[0])

    # max pooling operation for voxel values
    for i,v in enumerate(unique_xyz):
        # find all duplicate indices
        condition = (xyz_v[:,0]==v[0]) & (xyz_v[:,1]==v[1]) & (xyz_v[:,2]==v[2])
        dup_indices = np.where(condition)[0]
        # perform max pooling with the intensity values
        unique_hm[i] = np.max(hm[dup_indices])

    return unique_xyz, unique_hm
```

4.1.6 Combining Radar and VIO

Combining the radar and VIO data first requires one to match data points of the two individual data streams together. This is discussed below, followed by the actual method for combining the data.

Timings

As mentioned in Section 4.1.1, the radar sensor outputs heatmaps at 4 frames-per-second. The VIO data returned by the Android application comes in at a frequency between 30 to 60 FPS depending on the device. This means that data comes in from both devices are different rates and cannot be perfectly matched up. Even though efforts have been made to start both the radar and VIO data collection at roughly the same time, the exact start point will differ due to lack of formal synchronisation.

To match each heatmap with a VIO data frame, one needs to get a relative (from the start of recording) timestamp for both.

The procedure for doing so for a heatmap frame is as follows. The exact system time (in milliseconds) of the recording start is noted as `start_time`. The system time when the first timestamp is received is recorded as `first_frame_ts`. The actual relative-to-the-start-of-recording timestamp of this first frame is then taken to be:

```
first_frame_relative_ts = first_frame_ts - self.start_time - 100
```

The additional 100 ms is deducted as the sensor documentation lists the transmission time of the range-azimuth *and* range-Doppler heatmaps to be 200 ms. It is assumed that just the range-azimuth heatmap takes half this time.

The data transmission is assumed to be at a constant speed from then on, so every subsequent frame with frame number i (included in the frame header), can be taken to have a relative timestamp of:

```
relative_timestamp[i] = first_frame_relative_ts + i * 250
```

where 250 ms is the frame duration at 4 FPS.

The VIO data has timestamps attached to it, so the timestamp of the first VIO data frame received is taken to be the time base for calculating the relative timestamps of each subsequent frame.

Once the relative timestamps are calculated, each radar heatmap frame is matched with a VIO frame by smallest absolute difference:

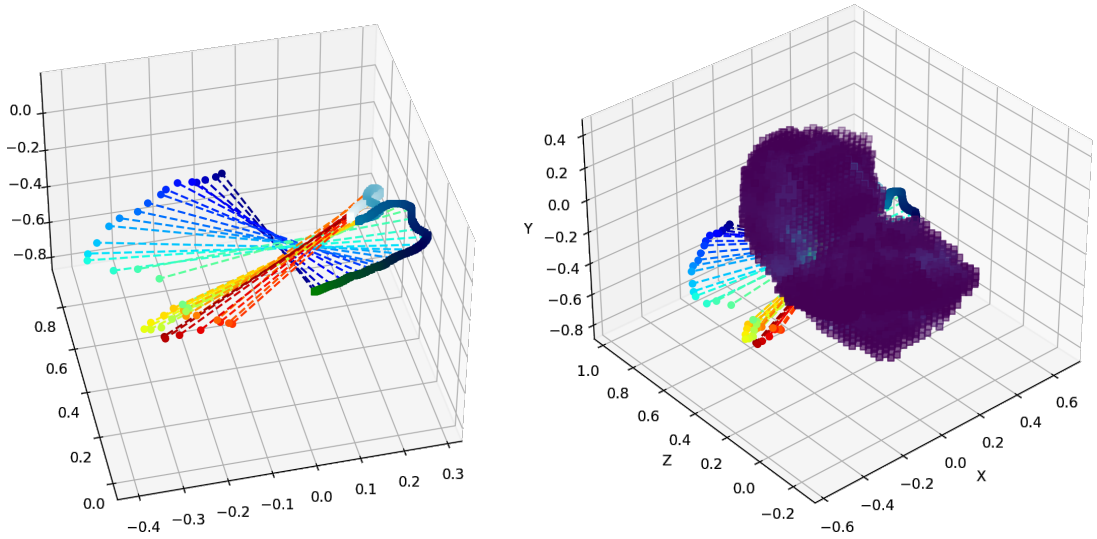
```
# returns indices for each heatmap timestamp with
# the closest vio entry in terms of relative timestamps
def find_closest_matches(heatmap_timestamps, vios):
    heatmap_timestamps = np.array(heatmap_timestamps)
    vio_relative_ts = np.array([x['relative_timestamp'] for x in vios])
    matches = []
    for ts in heatmap_timestamps:
        a = np.abs(vio_relative_ts - ts)
        arg = np.argmin(a)
        matches.append(arg)
    return matches
```

Combining

During the VIO post-processing step, the relative displacement of each VIO frame is also calculated, taking the position of the first frame to be the origin. This *relative* displacement is then used when combining VIO data with radar data.

Once each heatmap frame is matched with VIO data, one can perform the necessary transformations to bring the 3D heatmap into stable world co-ordinates:

```
xyz_list, hm_list = None, None
for radar_i, vio_i in enumerate(matches):
    # transform 2d heatmap into a 3d one
    xyz, hm = prepare_3d_heatmap(heatmap_data[radar_i], reader.config)
    # rotate + position the heatmap according to VIO data
    v = vio_data[vio_i]
    xyz_ = v['quat'].apply(xyz)
    xyz_ = xyz_ + v['relative_pos']

    xyz_list = xyz_ if xyz_list is None else np.concatenate([xyz_list, xyz_])
    hm_list = hm if hm_list is None else np.concatenate([hm_list, hm])
```



(a) Visualisation of VIO data points that match the collected radar heatmap timestamps.

(b) The combined radar and VIO data.

Figure 4.5: Visualising the combined radar and VIO data of a real scan. Sub-figure (a) shows just the VIO data: one can see how the radar was kept pointing at the object while the completing the scan. The colour scheme is identical to that in Figure 4.3. Sub-figure (b) shows the result of combining the VIO data with. its matching radar data. Note how the heatmap "pie" rotates to match the orientation of the scanner.

This combination procedure is followed by the voxelisation step described in Section 4.1.5 to produce a radar "world":

```
1 voxel_xyz, voxel_hm = voxelise(xyz_list, hm_list)
```

Figure 4.5 shows the result of voxelising the combined radar and VIO data, i.e. the radar "world".

4.1.7 Radar Cube

The resulting radar "world" shown in Figure 4.5 has an irregular shape, due to hand-held scanning, which would make any classification difficult. To properly train a classifier, one needs images of a fixed size. An option considered was to resize the radar world into some fixed dimensionality, but this would lead to information loss, especially keeping in mind that the object of interest would usually make up a small portion of the overall radar world.

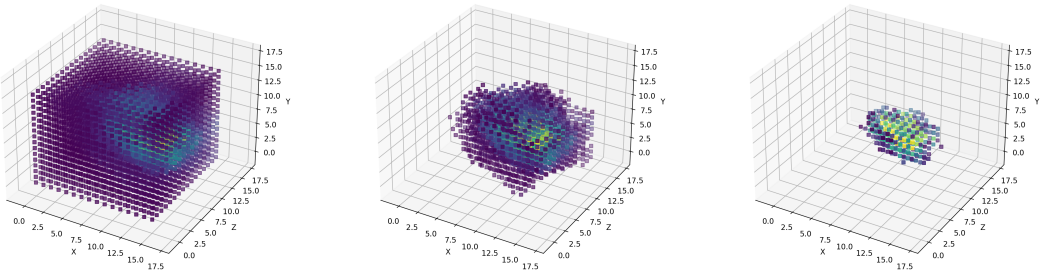
Hence, it was decided to crop a fixed-size cube out of the radar world. The fixed size of the cube was chosen to be $16 \times 16 \times 16$ voxels, which corresponds to a physical dimensionality of $0.5 \times 0.5 \times 0.5\,\mathrm{m}$. This was large enough to encompass all objects being tested, while being small enough to make the dimensionality tractable for a ML model to learn.

The cube was centred around the point of highest radar intensity, and any empty voxels in the cube were filled with the minimum radar intensity value found in the radar world. This filling-with-minimum-value was done to make the image more natural, as the radar scans themselves never had an exact-0 value due to radar noise. This radar cube was then treated as the final 3D image of the object being scanned, and was used for training and classification purposes.

The algorithm for producing the radar cube is again best summarised as code:

```
def get_normalised_radar_cube(world, crop=16):
    # get the voxel indices and heatmap intensities
    (xv, yv, zv), hmv = world.get()
    # get the index of the highest intensity value
    intense_ind = np.argmax(hmv)
    hxv, hyv, hzv = xv[intense_ind],yv[intense_ind],zv[intense_ind]

    # crop to size
    x_range = [hxv - crop//2, hxv + crop//2]
    y_range = [hyv - crop//2, hyv + crop//2]
    z_range = [hzv - crop//2, hzv + crop//2]

    cond = (x_range[0] <= xv) & (xv < x_range[1]) &\
           (y_range[0] <= yv) & (yv < y_range[1]) &\
           (z_range[0] <= zv) & (zv < z_range[1])
    xv, yv, zv = xv[cond], yv[cond], zv[cond]
    hmv = hmv[cond]

    # normalise indices to make sure all are >= 0
    xv, yv, zv = xv - np.min(xv), yv - np.min(yv), zv - np.min(zv)

    # fill image with min value in radar cube
    fill_value = np.min(hmv)
    img = np.full((crop, crop, crop), fill_value)
    img[xv,yv,zv] = hmv

    return img
```

Figure 4.6 shows the radar cube produced by applying the above method to the radar "world" presented in Figure 4.5. The sub-figures have progressively higher levels of filtering to reveal the rough outline of the object being scanned: a phone in this case.



(a) No filtering. (b) Filtering $< \mu$. (c) Filtering $< 5\mu$.

Figure 4.6: The final $16 \times 16 \times 16$ radar cube, centred around the point of highest intensity. The three subfigures (a-c) show the radar cube at increasingly higher levels of filtering with respect to the mean intensity μ, ultimately revealing the object of interest: a flat rectangular cuboid. This is in line with what we expect to see: the radar cube here is the result of a scan of a phone in a cardboard box, as described in Section 4.2.2.

4.2 Experiment Setup

This section will discuss the approach taken to conducting the classification experiment, including object class selection and the procedure for data collection. This experiment was performed to demonstrate the viability of the proposed method via the prototype described in Section 3.4.

4.2.1 Object Classes

As discussed in Section 3.2 and due to the time-consuming nature of data collection, I restricted the number of classes to three. The object classes were chosen to be: *phone, knife, screwdriver*. Figure 4.7 shows photographs of the objects used throughout the project.

(a) *Phone* (LG G5) (b) *Knife* (c) *Screwdriver*

Figure 4.7: Photographs of the objects used for conducting the experiments.

Figure 4.8: A knife placed in the cardboard box before being scanned using the built radar contraption. The front flap folds downwards and closes, completely obscuring the insides of the box.

Figure 4.9: The scanning setup. The white box (Figure 4.8 shows a close-up) containing the object to be classified has a red arrow pointing to it. It is situated on top of two empty cardboard shoe boxes.

4.2.2 Data Collection

This section describes the data collection procedure used to collect the training, validation, and test data sets.

The objects described in Section 4.2.1 were placed inside a cardboard box before being scanned to conceal their appearance. Figure 4.8 shows a photo of a knife in the aforementioned box. The scanning setup is shown in Figure 4.9 and involves the cardboard box being positioned on top of two empty cardboard shoe boxes, to raise it above the ground. Cardboard boxes are used as they do not have any metal components. Metal could cause interference with the radar scan and lead to degraded performance.

The following data collection procedure was employed:

1. The contraption c, described in Section 3.4 is switch on and calibrated (this involves moving it around in the environment to calibrate the VIO application).
2. An object $o \in \{\text{phone, knife, screwdriver}\}$ is selected.
3. o is placed inside the white cardboard box shown in Figure 4.8 in some random position and orientation.
4. The box with o inside – $[o]$ – is placed on some non-metallic elevated surface. As shown in Figure 4.9, this involved two more cardboard boxes.

5. c is positioned pointing at $[o]$ and the data collection is started.
6. c is waved around $[o]$ in a coherent fashion, tracing a path around and over the box, at a distance of roughly 20-30cm.
7. The data collection is stopped after roughly $5 - 10s$.
8. Steps 3 to 7 are repeated 55 times in total for the same object o. 40 of these recordings will be used for the training set, 5 for the validation set, and 10 for the test set. Note, the test set recordings were done on entirely separate occasions and in a different environment.
9. The procedure was repeated from Step 2 onward, selecting a different object o until all three objects were scanned 55 times in total.

4.3 Data Pipeline

This section describes the steps taken to post-process the obtained data to improve the ML classifiers' performances. The actual method behind the neural networks fits under the Implementation chapter due to mostly exploiting existing methods and is as such discussed in Section 5.3.4.

As briefly gestured at by the discussion in Section 2.4.5, ML models require a lot of data to not overfit and to perform well. Data collection is a very time-consuming and resource-intensive task. Hence, I was realistically unable to collect more than 55 scans per item. This, however, is not usually a sufficient amount of data to train a neural network on.

This was mitigated by *augmenting* the collected data. The training and validation sets were augmented independently to prevent data leakage between them. The augmentation happened after the radar and VIO data was combined (as described in Section 4.1.6), but before the voxelisation and formation of the radar cube. The following augmentations were randomly performed:

- The point cloud was rotated by a random unit quaternion around the point of highest radar intensity.
- The radar intensity values of the point cloud were randomly multiplied element-wise by a factor uniformly sampled in the range $[0.8, 1.2]$.

These augmentations were run 99 times per collected scan to produce the training and validation datasets. This brought the total number of items in the training and validation test sets to $12,000$ ($4,000$ per object class) and $1,500$ (500 per object class) respectively. The test set was not augmented as it is meant to closely represent the real-world experiment set-up. As such, the test set contained 30 items (10 per object class).

Finally, the data was normalised before being fed into the neural network. This was done by deriving the mean and standard deviation of the training set and normalising the entire dataset according to this. The normalisation fit the data to a Gaussian with mean 0 and standard deviation 1. The normalising constants were found to be $\mu = 819.54$ and $\sigma = 1658.33$. This normalisation technique is often employed to allow the neural network to converge faster and in a more stable fashion [48].

Chapter 5

Implementation

This chapter will present the practical work carried out for the project. An overview of the software components produced for the project and how they are interconnected is present first. The rest of the chapter then goes into implementation details of both the hardware and software components involved.

5.1 Project Structure

Below is the file tree of this project's repository, with Figure 5.1 visually summarising the below structure and its interconnectivity:

- `android_app` contains the code for the Android application, described in Section 5.3.1, used to collect the VIO data to be used in conjunction with the main data-collection program described in Section 5.3.2.
- `data` contains the collected training, validation, and test data, as well as the notebook to perform the post-processing.
  - `data.ipynb` is the notebook used for post-processing the collected raw data. This serves as the implementation of the methodology outlined in Section 4.3.
  - `train_val` contains the training and validation data sets.
  - `test` contains the test data set.
- `dev_data` contains miscellaneous data collected during the development process of the contraption to test its capabilities and to debug the code.
- `ver_data` contains the data collected for purposes of verifying the accuracy and correctness of implementation of the prototype, as described in Section 6.1.

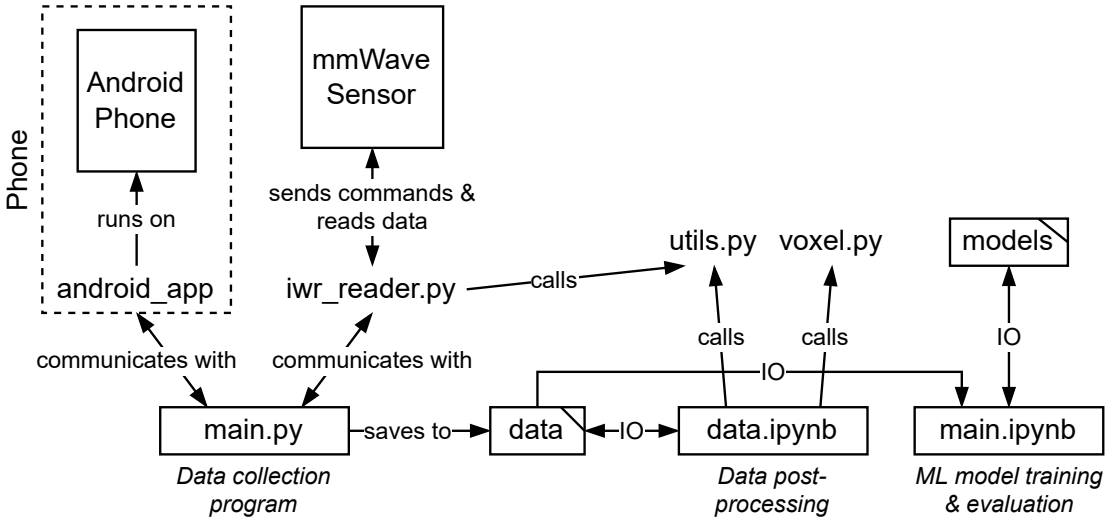


Figure 5.1: A visual summary of the project structure, as described in Section 5.1.

- `models` is the top-level folder containing ML-related materials, including the model source codes, model checkpoints, and the notebook used for running the models.
  - `{resnet,mobilenet,mobilenetv2,shufflenet,shufflenetv2}.py` contain the source code for the 3DCNN models described in Section 5.3.4, adapted from Köpüklü et al. [31].
  - `saved_models` contains the trained model checkpoints which were used for evaluating performance on the test data set.
  - `main.ipynb` is the main notebook used for training and evaluating the models.
- `notebooks`
  - `testing` contains miscellaneous notebooks that were used to test, in isolation, the various software components of the project during development.
  - `{end_to_end_test,serial_test,test_data_read}.ipynb` are notebooks which were used for respectively: testing the data post-processing code and visualising the collected data; for testing the interface intended for communicating with the radar; for the initial development and testing of the methodology described in Section 4.1 with regards to combining the radar heatmap and VIO, as well as the voxelisation process.
  - `verification_exp.ipynb` contains the code for analysing the results in the `ver_data` folder, obtained after conducting the verification experiments as described in Section 6.1.

- `src`
  - `iwr_reader.py` contains the source code for the `DataReader` class and its various helper functions. This class is at the core of the communication protocol between the radar and PC. It handles starting and stopping the radar scans, and decodes the raw radar data stream into more usable data formats. This latter functionality is described in Section 4.1.2.
  - `utils.py` contains various helper functions such as:
    - `find_closest_matches` which finds the best pairing between the radar heatmap data and the VIO data (which are received at different times with mismatched timestamps).
    - `process_vio` which processes the raw VIO packets into a more usable dictionary format.
  - `voxel.py` contains the source code for the `PersistentVoxels` class. This class is responsible for voxelising (quantising) the obtained 3D heatmap data. The methodology behind this class is described in Section 4.1.5. This file also contains the `prepare_3d_heatmap`, `to_radar_world`, and `get_normalised_radar_cube` functions:
    - `prepare_3d_heatmap` takes the raw 2D heatmap data and maps it into a 3D world, keeping in mind the technical specifications of the radar described in Section 4.1.3.
    - `to_radar_world` handles the process of combining the radar and VIO data into a stable voxelised 3D world using the `PersistentVoxels` class. This is described in Section 4.1.6. This function also handles the data augmentation described in Section 4.3.
    - `get_normalised_radar_cube` converts the voxelised 3D world of `PersistentVoxels` into a fixed-size ($16 \times 16 \times 16$) radar cube, via the method described in Section 4.1.7.
- `main.py` is the main entry point of the project and contains the source code for the data-collection program described in Section 5.3.2.

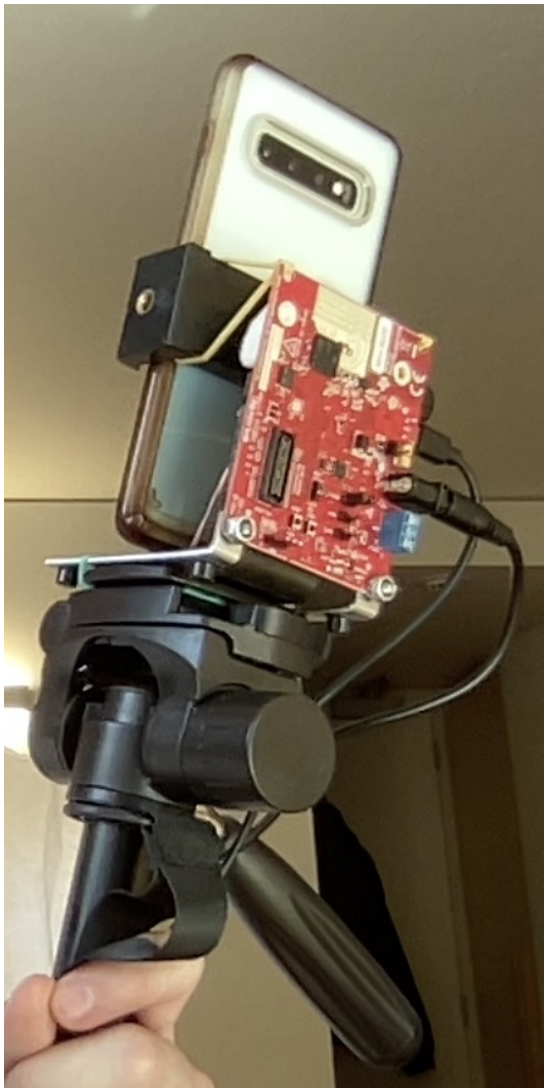


Figure 5.2: The final constructed device to be used for data collection, using the IWR1443BOOST evaluation board (red circuit board), and an Android phone.

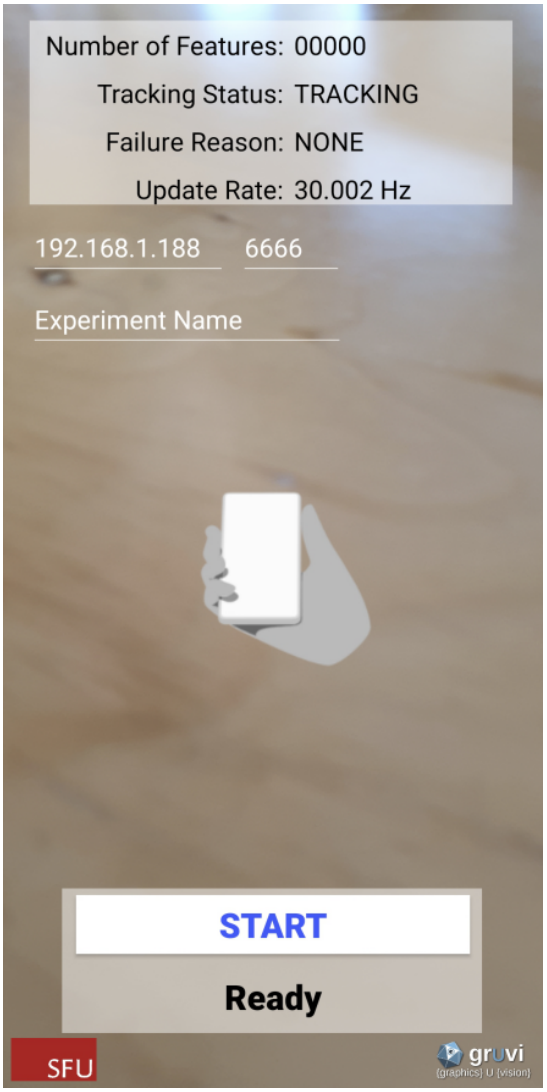


Figure 5.3: Screenshot of the Android application for collecting Visual-Inertial Odometry data, as described in Section 5.3.1.

5.2 Hardware

A photograph of the produced prototype can be seen in Figure 5.2. As described in Section 3.4, a camera tripod with a smartphone mounting point was used as a base to hold the various hardware components together. A Samsung Galaxy S10+ was used as to run the VIO Android application. The phone was securely placed in its designated mounting point, with the IWR1443BOOST evaluation board mounted in front of the it and secured using rubber bands and metal brackets included in the evaluation board's kit.

The phone communicates wirelessly with the main data-collection program `main.py`, described in Section 5.3.2, running on a laptop, while the evaluation board connects to the same laptop via a micro-USB cable and is powered via a wall socket.

5.3 Software

In addition to the physical device described in the section above, a variety of software was written to drive the hardware and analyse the produced data. While there was existing off-the-shelf software provided by Texas Instruments for obtaining data from the radar board, it did not allow me the flexibility I required – namely, I needed interactivity between the Android VIO application and the radar sensor.

While the Android application uses Java, the remainder of the software was written in Python. This includes the radar data parser and sensor communication interface.

5.3.1 Android Application

An Android application for obtaining the Visual-Inertial Odometry (VIO) was created. This was based on existing open-source software by Kim [49], modified to send data back to a laptop for processing. The application utilises Google's ARCore API [30]. An on-device screenshot of the application is presented in Figure 5.3.

A few changes were made to the open-source app:

1. The app would originally record pose obtained via the `Frame.getAndroidSensorPose()` function, which returns the pose of the smartphone's IMU sensor without accounting for screen rotation. After conducting some experiments, it was found that this pose did not properly align with the "expected" orientation of the phone. The final pose readings were obtained via the `Camera.getPose()` function that returns the pose of the physical camera in world space.

2. The app would record and save VIO data to a text file on-device. This made the process of extracting the data slow and time-consuming, so the app was modified to stream the data via to a UDP server in real-time as the data collection took place. The UDP server was hosted on the laptop that the radar was connected to, via the `main.py` program described in Section 5.3.2 below.

3. The "Start/Stop" button in-app was modified to send a start/stop packet to the aforementioned UDP server. The server would then simultaneously start recording the incoming VIO data *and* instruct the mmWave radar to, respectively, start or stop the scanning process.

   This allowed the operator of the scanner to control both the VIO data collection and the mmWave radar using just the Android application, which improved ease-of-use and greatly sped up data collection.

4. Editable text boxes were added to allowing the user to change the IP address and port of the receiving server in-app. The experiment's name could also be specified – this was sent together with the start packet to serve as the filename for the saved data.

Data is received from the application in the following format:

```
1 timestamp, q_x, q_y, q_z, q_w, t_x, t_y, t_z
```

where `timestamp` (in nanoseconds) indicates when this particular VIO frame was captured, without a specific time base. $q_{\{x,y,z,w\}}$ defines the orientation quaternion of the VIO frame, and $t_{\{x,y,z\}}$ – the position of the frame, in metres.

5.3.2 Data Collection Program

The main program (aptly named `main.py`) acts as a bridge between the Android application supplying the VIO data and the mmWave sensor. The program runs a UDP server that the Android application communicates with. It also runs the sensor communication interface via `iwr_reader.py`.

The program initialises the mmWave sensor with the provided configuration file as described in Section 4.1.1. A new software thread is then started to managed the UDP server. The program continuously scans the UDP socket for any oncoming packets, while the main program periodically probes the mmWave sensor's data port for any new data, which is only produced if the sensor is started.

Once a *start* packet, sent by the Android application, is received, the UDP server simultaneously starts recording the VIO packets being received and sends an instruction to the mmWave sensor to start the scanning process. Both the VIO and radar data are stored in temporary variables.

If a *stop* packet is received, the program signals to the radar sensor to pause recording. The VIO and radar data are then saved to a single file. The VIO data is lightly pre-processed into a dictionary format before saving via the `process_vio` function. This saved file can then be post-processed into its final usable radar cube form, as described in the following section.

5.3.3 Data Post-Processing

`data.ipynb` is the notebook used for post-processing the collected raw data. This serves as the implementation of the methodology outlined in Sections 4.1 and 4.3.

Namely, the notebook first combines the radar and VIO data and then augments it with random rotations and permutations of the intensity values:

```
from scipy.spatial.transform import Rotation as R

# xyz_list and hm_list are arrays of the
# combined radar and VIO data, as described in Section 4.1.6
def augment_radar_world(xyz_list, hm_list):
    # rotate by a random unit quaternion around the
    # point of highest radar intensity
    random_rot = R.random()

    intense_ind = np.argmax(hm_list)
    intense_pos = xyz_list[intense_ind]

    xyz_list = xyz_list - intense_pos
    xyz_list = random_rot.apply(xyz_list)
    xyz_list = xyz_list + intense_pos

    # multiply radar intensity values of the point cloud by
    # a factor in the range [0.8, 1.2]
    rand_intensity_factors = np.random.uniform(0.8, 1.2, hm_list.shape)
    hm_list = hm_list * rand_intensity_factors

    # voxelise the rotated heatmap all at once
    voxel_xyz, voxel_hm = voxelise(xyz_list, hm_list)
    return voxel_xyz, voxel_hm
```

As described in Section 4.3, this step is repeated 99 times per collected scan and is hence time-consuming. As such, the radar world is serialised and saved to disk after this augmentation to prevent data loss in case of hardware failure or other unforeseen errors. This step took roughly 20 hours for training and validation datasets.

After the augmentation generation is finished, the saved radar worlds are converted into radar cubes via the method described in Section 4.1.7:

```
radar_world_files = glob.glob("train/radar_world/*/*.npy")
for file_name in radar_world_files:
    # load the saved (augmented) radar world
    world = PersistentVoxels(file_name)
    # convert it into a radar cube
    img, _ = get_normalised_radar_cube(world)

    # save to file
    np.save(file_name.replace("radar_world", "radar_cube"), img)
```

After performing these augmentation and conversion steps, both the training and validation data sets are ready for use by the ML classifiers. As previously stated in Section 4.3, the test data set does not undergo the augmentation step, but is still converted into radar cube form.

5.3.4 ML Classifiers

There is an abundance of 2D convolutional neural networks (CNNs) that are used very successfully to classify and segment 2D images. However, the data we are working with is 3D in nature. 3DCNNs exist, but are usually used to classify time-series 2D data, such as video streams. As such, some work was needed to make these existing models applicable to this project.

As a starting point, I took the 3DCNN models described in Köpüklü et al. [31] and modified them to suit the needs of the project. The models presented in the paper are 3D adaptations of popular resource-efficient networks. Many of these models are small enough to be run on mobile devices. Namely, the following models were used for training and evaluation: ResNet18 [50], MobileNet [51], MobileNetV2 [52], ShuffleNet [53], ShuffleNetV2 [54].

The networks above are designed for classifying 100s of different classes and being trained on large datasets such as ImageNet with 3.2 million images in total [55]. The collected and subsequently augmented $12,000$ training data items and respective 3 classes do not compare. As such, these networks are prone to overfitting on the small radar cube dataset I collected, even with the data augmentation [56].

This was resolved by modifying the network architectures to include a `Dropout` layer after each ReLU activation. As discussed in Section 2.4.5, this is effective at reducing overfitting. The exact dropout rate was modified to suit the specific architecture it was applied to, but generally was in the range of $0.1 - 0.3$.

Other modifications include adjusting the CNN's kernel stride length to be 1 (down from 2 for most networks) to better capture the finer-grained features of the 3D scans. Smaller $3 \times 3 \times 3$ (down from $7 \times 7 \times 7$) kernels were also introduced for ResNet18 owing to the smaller input sample size. Lastly, the initial `MaxPool` layer (usually following the first 3D convolution) was removed again due to the small input size. This layer serves to reduce the dimensionality of the intermediate data in the network. This is useful when considering large inputs but leads to unnecessary information loss for small input sizes such as the one considered here.

`main.ipynb` (not to be confused with the data collection program `main.py`) is the main notebook used for training and evaluating the models. The optimiser used for training is Stochastic Gradient Descent (SGD), with the initial learning rate set to 0.05 and momentum to 0.9 [57]. This is described in Section 2.4.3. The models were trained for a maximum of 600 epochs where a single epoch is defined to be a full run-through of the training dataset. That said, most models required under 100 epochs to reach their best validation accuracy.

Early stopping was also used as a way of preventing overfitting: after every epoch, the validation set was evaluated and if the prediction accuracy exceeded the last best model checkpoint's, a new model checkpoint would be saved. Once all 5 models were trained to their best validation accuracy, they were used to make predictions on the test set. These results are presented in Section 6.2.

The training and evaluation was done using Google's Colab Pro platform[1] due to it's ease of use and ability to run Jupyter Notebooks. The platform also provided access to fast GPUs such as P100s and V100s, which reduced training time.

[1] `https://colab.research.google.com/`

Chapter 6

Evaluation

This chapter presents the results of the project and verifies the correctness and accuracy of implementation of the prototype. The results of the experiments described in Section 4.2 are also presented and discussed. Finally, the performance of each ML model tested is discussed in the context of its complexity, training speed and parameter count.

6.1 Verification

This section covers the various experiments performed to verify the correctness and accuracy of the approaches chosen to implement the proposed method. That is, the design decisions presented in Chapter 3 are scrutinised.

6.1.1 Pose Tracking

Google's ARCore library was used for position and orientation tracking. The library is extensively tested and actively maintained by its developers, its tracking capabilities were tested independently for this project. The two main factors to test were the position and orientation capabilities.

Position Accuracy

The position accuracy was tested as follows:

1. The VIO application was opened and calibrated.
2. A 30 cm ruler was placed on a table, and the back of the phone was aligned with starting mark on the ruler. The phone was held perpendicular to the surface of the table, facing the direction of the ruler.

3. The tracking was started and the phone was moved in the direction of the ruler until the back of the phone aligned with the end mark, i.e. the 30 cm mark. At this point the tracking was turned off.

4. The resulting relative displacement was calculated and compared against the expected 30 cm displacement. This error was recorded.

5. The above steps were repeated a total of 5 times in varying lighting and environment conditions to provide repeated measurements.

The results of the position accuracy measurement are as follows. The mean error was found to be 2.25 mm (0.75 % of the distance) with a standard deviation of 5.35 mm. These results show that the position tracking capabilities of ARCore are more than sufficient for the purposes of this project, and are actually in the same ballpark as the photogrammetry-based approach described in Section 3.3.2 despite the latter requiring significantly more setup.

Orientation Accuracy

The orientation accuracy was tested as follows:

1. Two perpendicular lines were drawn using a thick marker on a A3 sheet of paper, using a protractor to ensure the angle between them was exactly 90°.

2. The VIO application was opened and calibrated.

3. The phone was placed on the point of intersection of the two lines and oriented such that one of the lines was exactly in the middle of the screen.

4. The tracking was started and the phone was rotated such that the other perpendicular line was now centred on-screen. This constitutes a 90° rotation. At this point the tracking was turned off.

5. The resulting change in quaternion angle was calculated and compared against the expected 90° rotation. This error was recorded.

6. The above steps were repeated a total of 5 times in varying lighting and environment conditions to provide repeated measurements.

The results of the orientation accuracy measurement are as follows. The mean error was found to be $-0.192°$ (-0.21 % of the rotation) with a standard deviation of 0.615°. The orientation tracking provided by ARCore is hence found to also be accurate and more than sufficient for the purposes of this project.

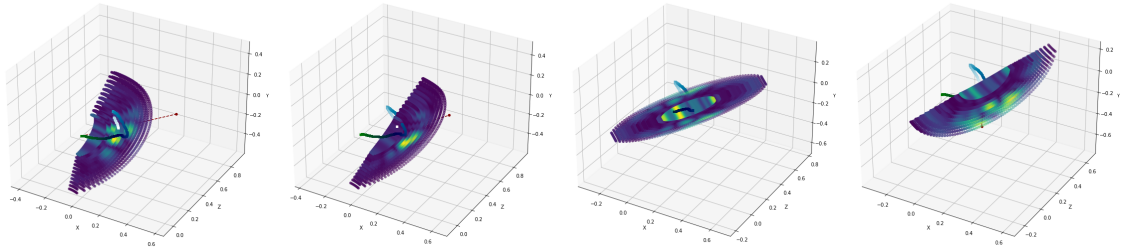


Figure 6.1: Visualising the intermediate steps of combining VIO and radar data.

Discussion

Both experiments show good performance with only minute errors. While the experiments only cover displacement and rotation in a single dimension or axis, it would be significantly more difficult to design repeatable experiments to measure error when moving/rotating in multiple dimensions or axis. These results verify that the tracking capabilities of the VIO approach are good enough for this project, especially given the 4 mm resolution of the radar scanner.

6.1.2 Communication Protocol

The correctness of implementation of the algorithm used to read-in the data being outputted by the IWR1443BOOST evaluation board, as described in Section 4.1.1, was verified as follows. Even though the decoding algorithm was written with reference to TI's technical manual, it was critical to test the implementation to ensure the data was being parsed correctly.

A number of raw packets were captured from the IWR1443BOOST board via TI's online mmWave Demo Visualiser[1] tool. These raw data packets were then decoded both using my implementation, found in `iwr_reader.py`, and using existing open-source code. The `serial_test.ipynb` notebook simulates a live mmWave sensor using a file containing raw captured packets. The values of the decoded Range-Azimuth heatmap, 3D point cloud, and header data were compared to the decoded values outputted by *Pythonic mmWave Toolbox for TI's IWR Radar Sensors*[2] and the *IWR1443-Read-Data-Python-MMWAVE-SDK-1*[3] repositories. The data matched up in all test cases.

6.1.3 Combining Radar and VIO

The process of ensuring that the radar and VIO data are combined correctly was done by manually visually inspecting the produced result. The intermediate output (that is, before all the frames would be combined into a single radar world) was

[1] `https://dev.ti.com/gallery/view/mmwave/mmWave_Demo_Visualizer/ver/2.1.0/`
[2] `https://github.com/m6c7l/pymmw`
[3] `https://github.com/ibaiGorordo/IWR1443-Read-Data-Python-MMWAVE-SDK-1`

looked at to make sure that the position and orientation of the radar scan matched up with what was happening in real life.

A print-out of some intermediate steps in a scan are presented in Figure 6.1. During development a short scan was performed and visualised to ensure the axes and rotations of the VIO and radar data lined up.

6.2 Results

Figure 6.2 summarises the main findings of this project. The table lists the 5 models evaluated and their respective parameter count. It then goes on to list metrics for both the validation and test data sets. The three metrics presented are accuracy, precision, and recall. These are calculated using true positive (TP), true negative (TN), false positive (FP), and false negative (FN) outcome counts:

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{6.1}$$

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{6.2}$$

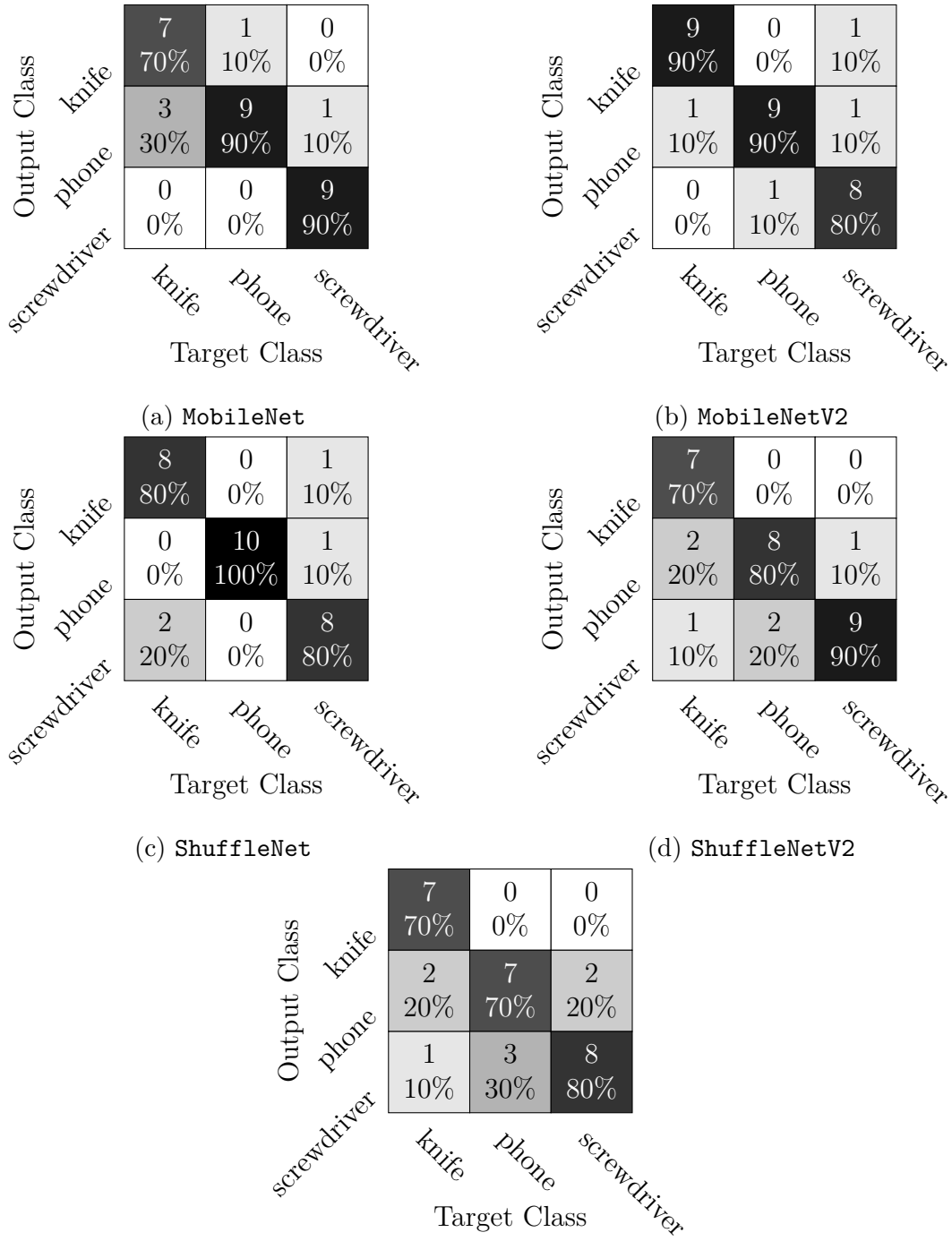
$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{6.3}$$

Both the validation and test datasets are balanced, so accuracy, as a metric, is representative. The precision and recall scores are *macro-averaged*, meaning that the metric is calculated separately for each class, and then averaged across classes with equal weighting.

		Accuracy [%]		Precision		Recall	
Model	Params	Validation	Test	Validation	Test	Validation	Test
ResNet18	32.97M	80.73	73.33	0.81	0.77	0.81	0.73
MobileNet	3.30M	82.13	83.33	0.82	0.86	0.82	0.83
MobileNetV2	2.36M	**85.00**	**86.67**	**0.85**	**0.87**	**0.85**	**0.87**
ShuffleNet	**0.95M**	81.47	**86.67**	0.83	**0.87**	0.81	**0.87**
ShuffleNetV2	1.30M	80.33	80.00	0.81	0.83	0.80	0.80

Figure 6.2: Results summary table, showing the validation and test set accuracies for the various models that were tested, as well as their parameter count. The best values in each category are highlighted.

There is value in knowing which of the objects are most often mis-labelled as others. The underlying assumption is that more similar objects, such as the elongated knife and screwdriver, will be mis-classified as one another more often than the objects with distinct characteristics, such as the wide flat phone. Hence, Figure 6.3 shows the confusion matrices of the five models evaluated. These results together with the results presented in Figure 6.2 will be interpreted in the following section. The confusion matrices for the validation data set can be found in Appendix A.



(a) `MobileNet` (b) `MobileNetV2`

(c) `ShuffleNet` (d) `ShuffleNetV2`

(e) `ResNet18`

Figure 6.3: Test dataset confusion matrices for the various models evaluated.

6.3 Discussion

Five different models were tested: ResNet18 [50], MobileNet [51], MobileNetV2 [52], ShuffleNet [53], and ShuffleNetV2 [54]. As shown in Figure 6.2, each has a different parameter count, which dictates its complexity and relative speed of training. The last four models (i.e. excluding ResNet18) are optimised for mobile processing owing to their lightweight and low-parameter nature.

ResNet18 was by far the longest model to train, which a $\times 10$ parameter count compared to the next largest model MobileNet. While its validation accuracies were roughly equal to ShuffleNetV2's, the performance did not transfer as well to the test set. This is likely due to the model overfitting on the training set due to the latter's small size and the former's high complexity. As such, this performance should not be used to immediately discount ResNet18 as a valid architecture to use for this classification task – given more training data and a larger variety of classes, it is likely to improve its performance transfer from validation to test set.

The smallest model, ShuffleNet, comparatively underperformed in validation accuracy but had the same performance metrics on the test set as the best-tested model MobileNetV2 but with $\times 2.5$ *fewer* parameters. The training process was also the quickest out of all tested models due to the simple architecture and very low parameter count. ShuffleNet is an "extremely efficient" CNN designed for mobile device. This makes it a very attractive candidate for the preferred model due to its efficiency and good performance. However, it is unclear whether this performance will scale up with more object classes.

ShuffleNetV2 performed relatively well achieving a 80% accuracy on the test set. However it did not match ShuffleNet's performance in both the validation and test sets. The performance transfer from validation set to test set was virtually one-to-one, which is encouraging in terms of controlling overfitting, with the comparatively low parameter count likely helping. The $\times 1.4$ increase in parameter count over ShuffleNet only marginally slows down training and evaluation, and would likely help the model perform better when more object classes are introduced.

MobileNet marginally underperformed compared to its newer MobileNetV2 version. Both models saw good performance transfer from validation to test set, which is again encouraging. MobileNetV2 was the best model in terms of both validation and test performance, getting the highest metric values in accuracy, precision, and recall. Training was quick on GPU due to the models being designed for mobile devices.

Overall, these results are very encouraging as they show that different model architectures are all able to learn to accurately classify the 3D radar scans produced by the prototype. As will be further discussed in Section 8.1, the good performance of mobile-designed networks paves a path for removing the laptop from the prototype's design altogether to make the device truly portable, moving the data-collection and ML classification entirely on to the smartphone.

Chapter 7

Related Work

This project demonstrates the feasibility of a *handheld* mmWave-based concealed weapon *detection and classification* method. As far as I am aware, this is the *first* method of concealed weapon detection using mmWave radar that manages to combine a device of hand-held nature and one with classification capabilities. However, this is not the first attempt at using mmWave technology for detection, classification, or tracking of objects, concealed or otherwise.

The academic research in this area differs significantly in the type of data used. While research by Gao et al. [58], Gupta et al. [59], Kim et al. [60], Zhao et al. [61] use pre-processed Range-Azimuth data similarly to what was done in this project, other works such as Liu et al. [62], Johnson and Chang [63] use raw radar data. As mentioned in Section 2.2.3, retrieving this raw data often requires the use of additional expensive equipment.

Gupta et al. [59], Kim et al. [60] use mmWave radar to detect and classify larger objects such as people, cars, drones, and cyclists. Zhao et al. [61] also use a similar approach to track, position, and identify multiple people in a room. While these papers show the viability of using radar in combination with deep learning, they are only tangentially related to the project at hand.

Liu et al. [62] achieve very impressive 2D radar imaging by leveraging a static scanning setup using a motorised 2D plane. However, the scans are performed from a very close distance (9 cm) to the target and require the use of an additional module (the DCA1000EVM evaluation board) to allow for real-time raw data capture. The scanning setup also requires very fine-grain control over the motorised movements, which makes the approach impractical for a truly handheld setup.

Gao et al. [58] use a cascading mmWave radar module (TIDEP-01012) to detect and classify concealed and open-carry objects on-person. This approach works well and utilises two additional cameras for depth perception. The setup is again however not very portable due to having a bulky design to house the large radar module and two cameras, and the need to be directly connected to a laptop.

While the radar module used provides better sensing capabilities but can be considered prohibitively expensive (1,318.80 USD for the MMWCAS-RF-EVM) compared to the IWR1443BOOST (358.80 USD) module used in this project.

Johnson and Chang [63] presents a portable method of mmWave radar imaging. The radar module used was custom-built. The paper demonstrates the feasibility of their approach in a static setup and makes a conjecture about potential hand-held operation using an inertial positioning system. This hypothesised extension has a striking similarity to the method proposed in this project. The paper however does not perform any classification of the obtained images, leaving this task in the hands of the device operator. This is likely due to the age of the paper, as it was published in 2001, before the widespread use of CNNs. The obtained images can be considered quite crude in nature, unless taken from a very close distance away, which makes this classification task hard even for humans.

Most of the methods described above rely on static setups, limiting their use in a real-world setting. The proposed method and the subsequent constructed prototype address both the cost factor of past approaches as well as the static nature of the setups, while providing accurate classification capabilities.

Chapter 8

Conclusion

This project proposes an innovative low-cost handheld method of mmWave radar imaging and explores its concealed weapon detection and classification capabilities. It leverages a mmWave radar sensor and a smartphone for position tracking. The proposed method was put into practice in the form of a physical prototype that was demonstrated to be able to achieve high (85%+) classification accuracy amongst three object classes. The design of the prototype was thoroughly scrutinised and all decision points weighted up. This was followed up by a number of experiments to verify the design claims.

The total unit cost of the prototype is just over 550 USD. As was discussed in Section 1.3.1, an airport mmWave body scanner, with detection and classification capabilities similar to that of the produced prototype, retails for over $170,000$ USD. The prototype's cost even beats a basic metal detector with no classification capabilities by a factor of two. I believe this comparison highlights the true *low-cost* nature of this method, making it competitive amongst existing technologies.

Furthermore, the produced prototype and accompanying methods are applicable in a wide range of scenarios. These include the anti-poaching exercises that served as an example of a real-world use-case of the technology. Possible use-cases in high-security environments include the handheld scanners being deployed in airports to replace the traditional handheld metal detectors. The approach can also be used to screen oversized cargo or as a quality control screening method on a factory line without needing to invest in large X-Ray machines.

The method and prototype developed in this project are a real advancement in low-cost radar imaging technology. There are however a number of ways in which this approach could be further improved. These are discussed below.

8.1 Potential Extensions

This section is broken into a number of sub-sections, each addressing different paths one could take if further developing the proposed method or if constructing a more advanced prototype.

8.1.1 On-Device Smartphone Classification

As alluded to in Section 6.3, the accurate classification results obtained via resource-efficient CNNs pave the road for removing the need for a laptop in the device setup altogether. That is, the smartphone that is used for obtaining the VIO data can perform the classification directly on-device. The neural network training would likely still be offloaded to a server, but performing inference on-device is very much within the realm of possibilities with the mobile-optimised networks. In fact, the best performing model was the aptly-named MobileNetV2.

This extension would require running software to read the mmWave sensor's data directly on the smartphone. While there are Python interpreters for the Android ecosystem, for efficiency's sake it would be worthwhile to rewrite the communication protocol in a more native language such as Java.

8.1.2 The Data Problem

One could increase the number of object classes that can be classified by the prototype. This requires no changes to the device itself, but would require significant data collection efforts. Though these efforts would be necessary to make the prototype useful outside of a research environment, regardless of whether the number of classified object classes were to increase. As discussed in Section 6.3, a larger dataset and more classes would also likely reshuffle the relative performance of the tested ML classification models.

8.1.3 Hardware Extensions

A relatively simple improvement to the prototype would be to use two radar modules fixed at a 90 degree offset from one another to improve the vertical field-of-view of the device, which is currently constrained by the low 15° elevation FoV. This would give the same FoV in both the vertical and horizontal directions, making it easier to scan larger objects and would improve the scanning speed overall.

A slightly more involved extension would involve using a custom-built radar array with even horizontal and vertical FoV and higher bandwidth to improve the imaging resolution. Custom silicon manufacturing is always a more involved task though, so this would be outside the scope of a single-year project and would likely need commercial sponsorship.

In the same light, one could build a custom multi-camera setup for improving the speed and reliability of the VIO tracking, especially in low-light and more homogeneous environments. One could look to a device such as the Oculus Quest VR headset that uses 4 wide-angle low-resolution cameras together with an IMU sensor to achieve tracking accuracies of around 1 mm, while being able to operate in low-light environments [64].

Finally, as discussed in Section 2.2.3, an additional DCA1000EVM board could be attached to the IWR1443BOOST evaluation board to allow it to output raw data. This would enable more accurate radar readings, but would require significantly more post-processing, likely limiting the viability of migrating to the smartphone-only setup described in Section 8.1.1. One could process the raw data using the method described in Liu et al. [62] and re-use the proposed method's way of combining multiple radar scans into a single 3D image to achieve very high-quality 3D radar scans.

References

[1] IEEE. Milestones:first millimeter-wave communication experiments by j.c. bose, 1894-96, Sept 2012. URL https://ethw.org/Milestones:First_Millimeter-wave_Communication_Experiments_by_J.C._Bose,_1894-96.

[2] ETSI. User equipment (ue) radio transmission and reception; part 3: Range 1 and range 2 interworking operation with other radios, Jul 2018. URL https://www.etsi.org/deliver/etsi_ts/138100_138199/138104/15.14.00_60/ts_138104v151400p.pdf.

[3] Gatwick Airport. Security scanners, Dec 2019. URL https://www.gatwickairport.com/at-the-airport/flying-out/security/security-scanners/.

[4] T. Teshirogi and T. Yoneyama. *Modern Millimeter-wave Technologies*. Wave summit course. Ohmsha, Limited, 2001. ISBN 9784274903823. URL https://books.google.co.uk/books?id=ijP52rsxTN8C.

[5] Khaled Khalaf, Vojkan Vidojkovic, John R. Long, and Piet Wambacq. *Low-power millimeter wave transmitters for high data rate applications*. Springer, 2019.

[6] David W Hagy. Portable x-ray systems for use in bomb identification, Dec 2007. URL https://www.ojp.gov/pdffiles1/nij/218586.pdf.

[7] Infineon Technologies AG. Health effects of mmwave radiation, Nov 2018. URL https://www.infineon.com/dgdl/Infineon-Health%20Effects%20of%20mmWave%20Radiation-PI-v01_01-EN.pdf?fileId=5546d46266a498f50166f1ada0520444.

[8] Stofl. Upending the status quo. URL https://stofl.net/.

[9] Richard Fynn and Oluwatoyin Kolawole. Poaching and the problem with conservation in africa (commentary), Mar 2020. URL https://news.mongabay.com/2020/03/poaching-and-the-problem-with-conservation-in-africa-commentary/.

[10] Poaching numbers: Conservation: Save the rhino international. URL https://www.savetherhino.org/rhino-info/poaching-stats/.

[11] Ming Yao. Wwf's ming yao on why china's elephant ivory trade ban matters, Sep 2018. URL https://www.worldwildlife.org/stories/wwf-s-ming-yao-on-why-china-s-elephant-ivory-trade-ban-matters.

[12] Security Magazine. Millimeter scanning at airports: Is it worth the cost?, Dec 2010. URL https://www.securitymagazine.com/articles/79508-millimeter-scanning-at-airports-is-it-worth-the-cost-1.

[13] Jeff Tyson and Ed Grabianowski. How airport security works, Jun 2021. URL https://science.howstuffworks.com/transport/flight/modern/airport-security4.htm.

[14] NIBIB. Computed tomography (ct), Jun 2022. URL https://www.nibib.nih.gov/science-education/science-topics/computed-tomography-ct.

[15] TSA. Computed tomography, 2022. URL https://www.tsa.gov/computed-tomography.

[16] K. Ajito, M. Nakamura, T. Tajima, and Y. Ueno. Terahertz spectroscopy methods and instrumentation. In John C. Lindon, George E. Tranter, and David W. Koppenaal, editors, *Encyclopedia of Spectroscopy and Spectrometry (Third Edition)*, pages 432–438. Academic Press, Oxford, third edition edition, 2017. ISBN 978-0-12-803224-4. doi: https://doi.org/10.1016/B978-0-12-409547-2.12092-X. URL https://www.sciencedirect.com/science/article/pii/B978012409547212092X.

[17] Lothar Moeller. Standoff detection of concealed weapons using a terahertz illuminator with an uncooled imager, Aug 2011. URL https://www.ojp.gov/pdffiles1/nij/grants/233347.pdf.

[18] Megan R. Leahy-Hoppa. Terahertz for weapon and explosive detection. *Critical Infrastructure Security*, page 207–220, 2012. doi: 10.2495/978-1-84564-562-5/13.

[19] Nashwan Jasim Hussein, Fei Hu, and Feng He. Multisensor of thermal and visual images to detect concealed weapon using harmony search image fusion approach. *Pattern Recognition Letters*, 94:219–227, 2017. ISSN 0167-8655. doi: https://doi.org/10.1016/j.patrec.2016.12.011. URL https://www.sciencedirect.com/science/article/pii/S016786551630366X.

[20] Siu-Yeung Cho and Nanda-Pwint Tin. Using infrared imaging technology for concealed weapons detection and visualization. In *TENCON 2010 - 2010 IEEE Region 10 Conference*, pages 228–233, 2010. doi: 10.1109/TENCON.2010.5685995.

[21] Cesar Iovescu and Sandeep Rao. The fundamentals of millimeter wave radar sensors (rev. a), Jul 2020. URL https://www.ti.com/lit/wp/spyy005a/spyy005a.pdf.

[22] Akash Gondalia, Dec 2017. URL `https://e2e.ti.com/support/sensors-group/sensors/f/sensors-forum/650639/iwr1443boost-about-elevation-estimation`.

[23] Texas Instruments. Iwr1443boost evaluation module mmwave sensing solution, May 2020. URL `https://www.ti.com/lit/ug/swru518d/swru518d.pdf`.

[24] Texas Instruments. mmwave demo visualizer, Apr 2020. URL `https://www.ti.com/lit/ug/swru529c/swru529c.pdf`.

[25] Texas Instruments. mmwave industrial toolbox: Understanding the out of box demo data output, 2020. URL `https://dev.ti.com/tirex/explore/content/mmwave_industrial_toolbox_4_10_0/labs/Out_Of_Box_Demo/docs/understanding_oob_uart_data.html`.

[26] Niko Sünderhauf, Kurt Konolige, Simon Lacroix, and Peter Protzel. Visual odometry using sparse bundle adjustment on an autonomous outdoor vehicle. *Autonome Mobile Systeme 2005*, page 157–163, Dec 2005. doi: 10.1007/3-540-30292-1_20.

[27] A.I. Comport, E. Malis, and P. Rives. Real-time quadrifocal visual odometry. *The International Journal of Robotics Research*, 29(2-3):245–266, 2010. doi: 10.1177/0278364909356601.

[28] Manon Kok, Jeroen D. Hol, and Thomas B. Schön. Using inertial sensors for position and orientation estimation. *Foundations and Trends® in Signal Processing*, 11(1-2):1–153, 2017. doi: 10.1561/2000000094. URL `https://doi.org/10.1561%2F2000000094`.

[29] Otto Seiskari, Pekka Rantalankila, Juho Kannala, Jerry Ylilammi, Esa Rahtu, and Arno Solin. Hybvio: Pushing the limits of real-time visual-inertial odometry, 2021. URL `https://arxiv.org/abs/2106.11857`.

[30] Google. Google arcore. URL `https://developers.google.com/ar/`.

[31] Okan Köpüklü, Neslihan Kose, Ahmet Gunduz, and Gerhard Rigoll. Resource efficient 3d convolutional neural networks. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 1910–1919. IEEE, 2019.

[32] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 edition, 2010.

[33] Iffat Zafar, Giounona Tzanidou, Richard Burton, Nimesh Patel, and Leonardo Araujo. *Hands-On Convolutional Neural Networks with TensorFlow: Solve Computer Vision Problems with Modeling in TensorFlow and Python*. Packt Publishing, 2018. ISBN 1789130336.

[34] F. Rosenblatt. The perceptron - a perceiving and recognizing automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory, Ithaca, New York, January 1957.

[35] Abien Fred Agarap. Deep learning using rectified linear units (relu), 2018. URL https://arxiv.org/abs/1803.08375.

[36] Ding-Xuan Zhou. Universality of deep convolutional neural networks, 2018. URL https://arxiv.org/abs/1805.10769.

[37] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*, chapter 6.5 Back-Propagation and Other Differentiation Algorithms. MIT Press, 2016.

[38] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2016.

[39] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks, 2015. URL https://arxiv.org/abs/1511.08458.

[40] Benyamin Ghojogh and Mark Crowley. The theory behind overfitting, cross validation, regularization, bagging, and boosting: Tutorial, 2019. URL https://arxiv.org/abs/1905.12787.

[41] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

[42] Akash Gondalia. Iwr1843boost: Iwr1843boost versus iwr1443, Jan 2020. URL https://e2e.ti.com/support/sensors-group/sensors/f/sensors-forum/873593/iwr1843boost-iwr1843boost-versus-iwr1443.

[43] Vladimir Bellavista-Parent, Joaquin Torres-Sospedra, and Antoni Perez-Navarro. New trends in indoor positioning based on WiFi and machine learning: A systematic review. In *2021 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*. IEEE, nov 2021. doi: 10.1109/ipin51156.2021.9662521. URL https://doi.org/10.1109%2Fipin51156.2021.9662521.

[44] A. Ward, A. Jones, and A. Hopper. A new location technique for the active office. *IEEE Personal Communications*, 4(5):42–47, 1997. doi: 10.1109/98.626982.

[45] Ali Ghofrani, Rahil Mahdian Toroghi, and Sayed Mojtaba Tabatabaie. Icps-net: An end-to-end rgb-based indoor camera positioning system using deep convolutional neural networks, 2019. URL https://arxiv.org/abs/1910.06219.

[46] Kejian Wu, Ahmed Ahmed, Georgios A. Georgiou, and Stergios I. Roumeliotis. A square root inverse filter for efficient vision-aided inertial navigation on mobile devices. In *Robotics: Science and Systems*, 2015.

[47] Laura Silver. Smartphone ownership is growing rapidly around the world, but not always equally, Aug 2020. URL https://www.pewresearch.org/global/2019/02/05/smartphone-ownership-is-growing-rapidly-around-the-world-but-not-always-equall

[48] Dalwinder Singh and Birmohan Singh. Investigating the impact of data normalization on classification performance. *Applied Soft Computing*, page 105524, 05 2019. doi: 10.1016/j.asoc.2019.105524.

[49] Pyojin Kim. Pyojinkim/arcore-data-logger: Android app to save arcore results (visual-inertial odometry) to a series of text files for offline use., 2019. URL `https://github.com/PyojinKim/ARCore-Data-Logger`.

[50] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. URL `https://arxiv.org/abs/1512.03385`.

[51] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017. URL `https://arxiv.org/abs/1704.04861`.

[52] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. 2018. doi: 10.48550/ARXIV.1801.04381. URL `https://arxiv.org/abs/1801.04381`.

[53] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices, 2017. URL `https://arxiv.org/abs/1707.01083`.

[54] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design, 2018. URL `https://arxiv.org/abs/1807.11164`.

[55] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

[56] Xue Ying. An overview of overfitting and its solutions. *Journal of Physics: Conference Series*, 1168:022022, feb 2019. doi: 10.1088/1742-6596/1168/2/022022. URL `https://doi.org/10.1088/1742-6596/1168/2/022022`.

[57] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1139–1147, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL `https://proceedings.mlr.press/v28/sutskever13.html`.

[58] Xiangyu Gao, Hui Liu, Sumit Roy, Guanbin Xing, Ali Alansari, and Youchen Luo. Learning to detect open carry and concealed object with 77ghz radar. 2021. doi: 10.48550/ARXIV.2111.00551. URL `https://arxiv.org/abs/2111.00551`.

[59] Siddharth Gupta, Prabhat Kumar Rai, Abhinav Kumar, Phaneendra K. Yalavarthy, and Linga Reddy Cenkeramaddi. Target classification by mmwave fmcw radars using machine learning on range-angle images. *IEEE Sensors Journal*, 21(18):19993–20001, 2021. doi: 10.1109/JSEN.2021.3092583.

[60] Jin-Cheol Kim, Hwi-Gu Jeong, and Seongwook Lee. Simultaneous target classification and moving direction estimation in millimeter-wave radar system. *Sensors*, 21(15), 2021. ISSN 1424-8220. doi: 10.3390/s21155228. URL `https://www.mdpi.com/1424-8220/21/15/5228`.

[61] Peijun Zhao, Chris Xiaoxuan Lu, Jianan Wang, Changhao Chen, Wei Wang, Niki Trigoni, and Andrew Markham. mid: Tracking and identifying people with millimeter wave radar. In *2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 33–40, 2019. doi: 10.1109/DCOSS.2019.00028.

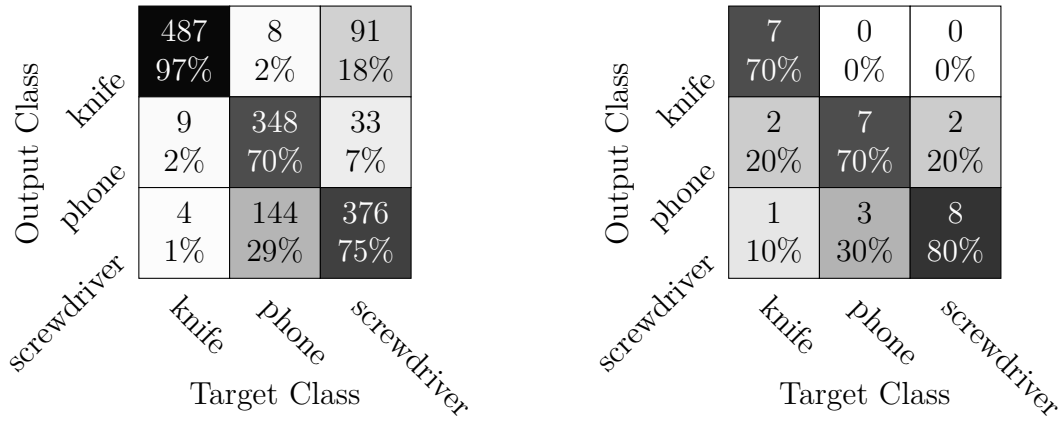
[62] Jie Liu, Kai Zhang, Zhenlin Sun, Qiang Wu, Wei He, and Hao Wang. Concealed object detection and recognition system based on millimeter wave fmcw radar. *Applied Sciences*, 11(19), 2021. ISSN 2076-3417. doi: 10.3390/app11198926. URL `https://www.mdpi.com/2076-3417/11/19/8926`.

[63] Michael A. Johnson and Yu-Wen Chang. Portable concealed weapon detection using millimeter-wave FMCW radar imaging. In Simon K. Bramble, Edward M. Carapezza, Lenny I. Rudin, Lenny I. Rudin, and Simon K. Bramble, editors, *Enabling Technologies for Law Enforcement and Security*, volume 4232, pages 134 – 141. International Society for Optics and Photonics, SPIE, 2001. doi: 10.1117/12.417525. URL `https://doi.org/10.1117/12.417525`.

[64] Valentin Holzwarth, Joy Gisler, Christian Hirt, and Andreas Kunz. Comparing the accuracy and precision of steamvr tracking 2.0 and oculus quest 2 in a room scale setup. 03 2021. doi: 10.1145/3463914.3463921.

Appendix A

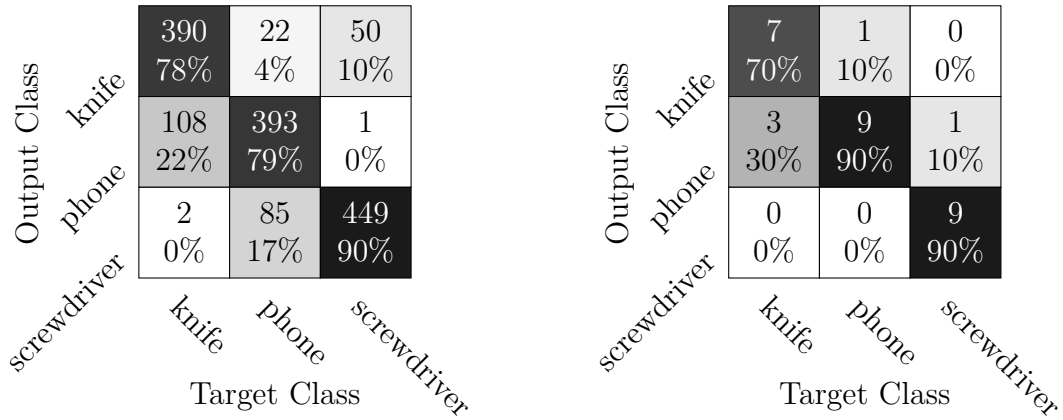
Confusion Matrices



(a) Validation set; accuracy is 80.73%.

(b) Test set; accuracy is 73.33%.

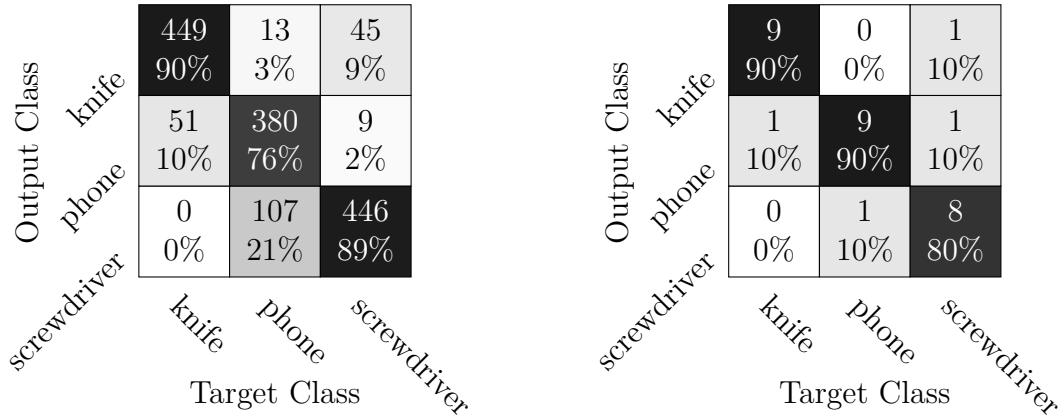
Figure A.1: `ResNet18` confusion matrices.



(a) Validation set; accuracy is 82.13%.

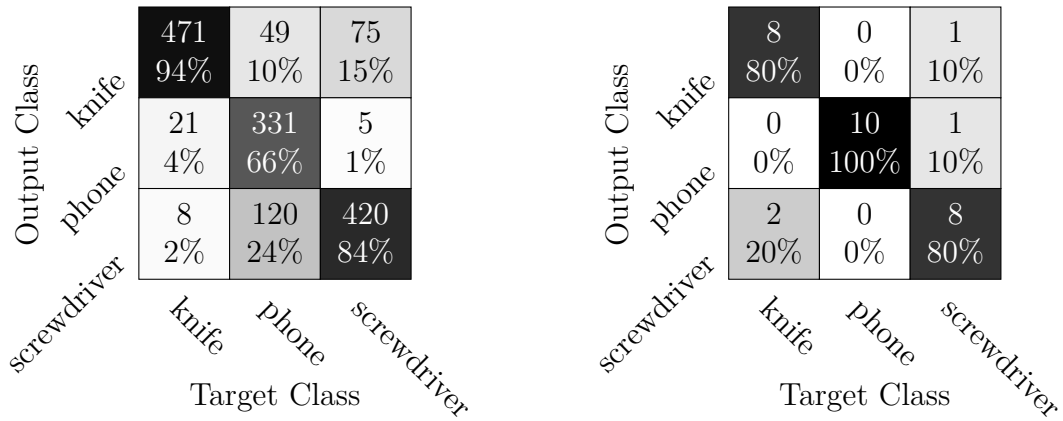
(b) Test set; accuracy is 83.33%.

Figure A.2: `MobileNet` confusion matrices.



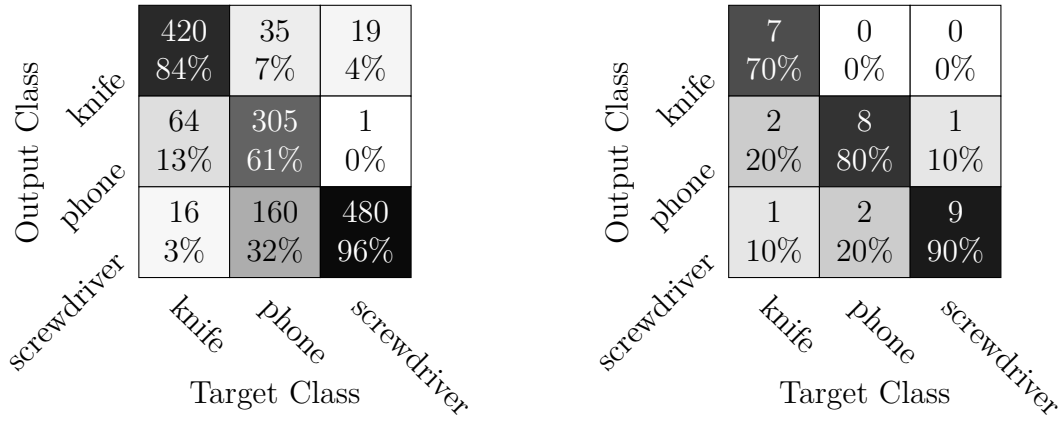
(a) Validation set; accuracy is 85.00%. (b) Test set; accuracy is 86.67%.

Figure A.3: `MobileNetV2` confusion matrices.



(a) Validation set; accuracy is 81.47%. (b) Test set; accuracy is 86.67%.

Figure A.4: `ShuffleNet` confusion matrices.



(a) Validation set; accuracy is 80.33%. (b) Test set; accuracy is 80.00%.

Figure A.5: `ShuffleNetV2` confusion matrices.