Tampere University

Juhani Kupiainen

MULTI-MODAL DATA RECORDING SYSTEM

implementation and documentation

Master's thesis
Faculty of Information Technology and Communication Sciences
Examiners: Prof. Mikko Valkama
Prof. Bo Tan
June 2023

ABSTRACT

Juhani Kupiainen: Multi-modal data recording system
Master's thesis
Tampere University
Master's Programme in Information Technology
June 2023

With the growth in computing power and sensor capability, new possibilities are opened for human activity recognition, for which multiple applications have been proposed. Most notably, security and health care applications, such as shoplifting and vandalism detection, fall detection, and respiratory rate sensing. Some commercial applications are already available, such as the health and fitness applications found in smart phones and watches, that automatically detect certain types of exercise.

Typically, wearable sensors are used for human activity recognition. Using gyroscopes and accelerometers, key positions in the human body can be accurately tracked, which can be used for very high-performance activity recognition, given the number of sensors is sufficient. Wearable sensors have an inherent drawback, though. They can be cumbersome and unfashionable, and they require willful equipping. They also must be battery-powered to enable free movement, which makes them less reliable than a mains-powered solution.

In the context of human activity recognition, visible spectrum imaging (RGB camera) has traditionally been the most popular choice for remote sensing. Unfortunately, the RGB camera is very susceptible to obstructions and varying lightning conditions, which limits its usefulness. The RGB camera is also inherently privacy-invasive. Clearly, there is a demand for other means of remote sensing. The majority of available data sets for human activity recognition considers only wearable sensors and visible-spectrum imaging. To advance the state of human activity recognition via remote sensing, new data sets are required that consider more remote sensors.

In this thesis, a portable multi-modal data recording system was developed. The system encompasses an RGB camera, a stereo-vision based depth camera, a low-resolution infrared camera, a microphone, and a millimeter-wave radar. This thesis documents the used sensors, the architecture of the software, and the data output formats of the software. The developed system remained slightly unstable, but still suitable for a small-scale data collection campaign. The instability was likely due to the used radar device or some programming error. If the instability can be fixed, the system will be capable of carrying out even larger-scale data collection campaigns.

Keywords: Human activity recognition, Data set, Sensor fusion, Radar

TIIVISTELMÄ

Juhani Kupiainen: Useamodaalisen datan nauhoitusjärjestelmä
Diplomityö
Tampereen yliopisto
Tietotekniikan DI-ohjelma
Kesäkuu 2023

Tietokoneiden laskentatehon ja sensorien kyvykkyyden kasvu avaa uusia mahdollisuuksia aktiviteetin tunnistukselle, jolle onkin jo esitetty useita mahdollisia sovelluksia. Merkittävimpiä sovelluksia ovat muun muassa turvallisuus- ja terveyssovellukset, kuten myymälävarkauksien ja ilkivallan tunnistaminen, kaatumisen tunnistaminen, sekä hengitystiheyden havainnointi. Joitain kaupallisia sovelluksia on jo saatavilla, kuten älykelloista ja -puhelimista löytyvät terveyssovellukset, jotka osaavat tunnistaa joitain liikuntamuotoja.

Tyypillisesti aktiviteetintunnistukseen käytetään puettavia sensoreita. Gyroskoopeilla ja kiihtyvyysantureilla saadaankin seurattua avainkohtien liikkeitä hyvin tarkasti, mikä mahdollistaa suorityskykyisen aktiviteetintunnistuksen, kunhan sensoreita on riittävästi. Puettavien sensoreiden varjopuolena kuitenkin on, että ne saattavat olla epämukavia ja epämuodikkaita. Lisäksi ne vaativat suostumuksellista pukemista, mikä saattaa rajoittaa käyttökohteita. Puettavien sensoreiden olisi myöskin syytä olla akkukäyttöisiä, mikä tekee niistä epäluotettavia verrattuna verkkovirtaan kytkettäviin sensoreihin.

Aktiviteetin tunnistuksessa näkyvän spektrin valon kuvaaminen (RGB kamera) on perinteisesti ollut suosituin vaihtoehto etämittaukselle. Valitettavasti RGB kamera on herkkä näköesteille ja valoolosuhteille, mikä rajoittaa sen käytettävyyttä. Lisäksi RGB kamera on luonnostaan yksityisyyttä loukkaava. On selvää, että muille etämittausmenetelmille on kysyntää. Valtaosa aktiviteetin tunnistukseen saatavilla olevista dataseteistä kattaa vain puettavia sensoreita sekä RGB videon. Jotta etämittauksella suoritettavaa aktiviteetintunnistusta voidaan kehittää, on luotava lisää datasettejä, jotka kattaavat muitakin etämittausmenetelmiä.

Tässä diplomityössä luotiin liikuteltava useamodaalinen nauhoitusjärjestelmä. Järjestelmä kattaa RGB kameran, stereonäköä hyödyntävän syvyyskameran, matalaresoluutioisen infrapunakameran, mikrofonin, sekä millimetriaaltotutkan. Tämä diplomityö dokumentoi käytetyt sensorit, toteutetun ohjelmiston arkkitehtuurin, sekä toteutetun ohjelmiston tuottamat dataformaatit. Toteutettu järjestelmä jäi hieman epävakaaksi, mutta se soveltuu kuitenkin pienimuotoisten datasettien keräämiseen. Epävakaus johtui luultavasti joko käytetystä tutkalaitteesta tai ohjelmointivirheestä. Jos epävakaus saadaan korjattua, järjestelmä soveltuu myös suuremman mittakaavan datasettien keräämiseen.

Avainsanat: Aktiviteetintunnistus, Datasetti, Sensorifuusio, Tutka

PREFACE

Carrying out this thesis has been a significant learning experience, and I wish to thank the individuals who have contributed to this accomplishment.

I am utmost grateful for my supervisor, professor Bo Tan, for proposing this project to me and offering his guidance. Without his enthusiasm and feedback, the end result would not have been half as good. Under his guidance, my knowledge of radar processing also increased significantly. I would also like to thank my second supervisor, Dr. Joonas Säe, for his help in refining the quality of the literary product.

Special thanks must also be given to my current employer Insta Advance and to my foreman, Teemu Kankaanranta, for allowing me the necessary resources for finishing the literature part of this thesis.

My gratitude also extends to my friends, to my family, and to my partner Elina, who have supported me through this process and pushed me forward when I was lacking the resolve.

To everyone who has played a role, whether big or small, I extend my sincerest thanks. Your contributions have enriched the quality of this thesis and have helped shape its final outcome. It is with profound gratitude that I present this work, hoping it will contribute to the field and inspire further exploration and research.

Tampere, 5 June 2023

Juhani Kupiainen

CONTENTS

ABBREVIATIONS

2D-FFT	2-Dimensional Fast Fourier Transform
2D-MUSIC	2-Dimensional Multiple Signal Classification
AIC	Akaike Information Criteria
AoA	Angle of Arrival
API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
CA	Cell Averaging
CFAR	Constant False Alarm Rate
CLI	Command Line Interface
CSV	Comma Separated Value
FBSS	Forward-Backward Spatial Smoothing
FFT	Fast Fourier Transform
FIFO	First In First Out
FMCW	Frequency Modulated Constant Waveform
FoV	Field of Vision
FPGA	Field-Programmable Gate Array
FPS	Frames Per Second
GIL	Global Interpreter Lock
HAR	Human Activity Recognition
I/Q	In-phase/Quadrature
IPv4	Internet Protocol version 4
IR	Infrared
MDL	Minimum Description Length
mmWave	millimeter-wave
MUSIC	Multiple Signal Classification
OS	Ordered Statistic
SNR	Signal to Noise Ratio

SSD	Solid State Drive

SYMBOLS

$*$	Complex conjugate
$\dagger$	Hermitian transpose
α_l	Amplitude of the signal reflected from the l:th target
$\boldsymbol{\delta}$	A vector of samples in the flattened scanning window in Forward Backward Spatial Smoothing
$\Delta\varphi$	Phase difference
Δf	Difference in frequency
Δf_{b}	Difference in beat frequency
ΔR	Difference in range
Δv	Velocity difference
$\mathbf{\Lambda}$	Diagonal eigen values matrix
λ	Wavelength of a signal
μ	Mean power of the power spectrum
$\hat{\mu}$	Estimated mean power of the power spectrum
ω	Phase velocity of a signal
$\omega(k, n)$	Additive white gaussean noise in the n:th sample of the k:th receiver
φ	Phase of a signal
φ_{RX}	Phase of the received signal
φ_{TX}	Phase of the transmitted signal
Ψ	Maximum field of vision in altitude direction
ψ	Altitude angle
Θ	Maximum field of vision in azimuth direction
θ	Azimuth angle
θ_l	The azimuth angle of the l:th target
$\mathbf{A}$	Angle steering matrix
$\mathbf{a}$	Angle steering vector
B	Bandwidth of the transmitted chirp

$\mathbf{b}$	A vector of bytes red from a file
$\mathbf{c}$	A sampled radar chirp
c	The speed of light
$\mathbf{C}_{\tilde{\mathbf{D}}}$	Covariance matrix of the mean of the smoothed matrices $\mathbf{D}_m$ for a radar cube
$\mathbf{C}_{\tilde{\mathbf{D}}_m}$	Covariance matrix for the smoothed data matrix $\mathbf{D}_m$
C_m	Width of the cell averaging window in CA CFAR algorithm
c_{m}	Column of an element in the cell averaging window in CA CFAR algorithm
$\mathbf{c}_m$	The m:th chirp in a radar frame
$c_{\mathrm{m,max}}$	Maximum column of an element in the cell averaging window in CA CFAR algorithm
$c_{\mathrm{m,min}}$	Minimum column of an element in the cell averaging window in CA CFAR algorithm
C_n	Height of the cell averaging window in CA CFAR algorithm
c_{n}	Row of an element in the cell averaging window in CA CFAR algorithm
$c_{\mathrm{n,max}}$	Maximum row of an element in the cell averaging window in CA CFAR algorithm
$c_{\mathrm{n,min}}$	Minimum row of an element in the cell averaging window in CA CFAR algorithm
$\mathbf{D}$	Mean of the radar data cube along the first axis (receivers)
$\mathbf{d}$	A vector of data: raw bytes interpreted as some data type
d	The distance between two adjacent receivers
$\mathbf{d}_{\mathrm{i}}$	Imaginary samples of the data vector $\mathbf{d}$
$\mathbf{D}_m$	A matrix of samples corresponding the the m:th dwell in a radar cube
$\mathbf{d}_{\mathrm{r}}$	Real samples of the data vector $\mathbf{d}$
$\tilde{\mathbf{D}}_m$	Spatially smoothed matrix $\mathbf{D}_m$
$\mathbf{f}$	The samples of a single chirp
f	Frequency of the transmitted signal
f_{b}	Beat frequency
$f_{\mathrm{b},l}$	Beat frequency of the l:th target

f_{d}	Doppler-shift
$f_{\mathrm{d},l}$	Doppler-shift of the l:th target
F_{s}	Sampling Rate
$\mathbf{F}_z$	The z:th frame as a matrix
$\mathbf{f}_z$	The z:th frame as a vector
G	Sum of the guard window in CA CFAR algorithm
G_m	Width of the guard window in CA CFAR algorithm
g_m	Column of of an element in the guard window in CA CFAR algorithm
$g_{\mathrm{m,max}}$	Maximum column of of an element in the guard window in CA CFAR algorithm
$g_{\mathrm{m,min}}$	Minimum column of of an element in the guard window in CA CFAR algorithm
G_n	Height of the guard window in CA CFAR algorithm
g_n	Row of of an element in the guard window in CA CFAR algorithm
$g_{\mathrm{n,max}}$	Maximum row of of an element in the guard window in CA CFAR algorithm
$g_{\mathrm{n,min}}$	Minimum row of of an element in the guard window in CA CFAR algorithm
H	The vertical resolution of a camera
h	The row of a pixel in a IR, RGB, or Depth video frame
H_0	The null hypothesis
H_1	Alternative hypothesis, $\neg H_0$
j	The imaginary unit: $j = \sqrt{-1}$
K	Number of receiving channels
k	Index of a receiver in the radar cube
K'	Number of receiving channels in a conditionally augmented radar cube
L	Number of reflecting radar targets
l	Index of a radar target
M	Number of chirps per frame
m	Index of a chirp in a frame
N	Number of samples per chirp
n	Index of a sample in a chirp

N_{FPS}	Number of frames per seconds, i.e. framerate
N_{total}	Total number of samples in a frame: $N_{\mathrm{total}} = NMK$.
$\mathbf{P}$	Power spectrum matrix
$\mathbf{p}$	A vector of pixels
p_1	Number of possible rows for the scanning window for Forward Backward Spatial smoothing
p_2	Number of possible columns for the scanning window for Forward Backward Spatial smoothing
P_{fa}	Probability of a false alarm
$\mathbf{P}_{\mathrm{filtered}}$	Power spectrum that has been attenuated according to coefficients in $\mathbf{u}_{\mathrm{R}}$ and $\mathbf{u}_{\mathrm{v}}$
$P_{n,m}$	Power of the cell in n:th row and m:th column of P
$\hat{\mathbf{P}}$	The range spectrum matrix of a radar cube
$\hat{\mathbf{p}}$	Range spectrum vector of a chirp in the radar cube
$\tilde{p}_1$	Row coordinate of the scanning window in Forward Backward Spatial Smoothing
$\tilde{p}_2$	Column coordinate of the scanning window in Forward Backward Spatial Smoothing
$\mathbf{P}'$	The unshifted range-velocity spectrum after applying the 2D-FFT to a radar data cube
$\mathbf{p}'$	The velocity spectrum vector for a single range bin accross multiple chirps
Pr	Probability function
$\mathbf{Q}$	Eigen vectors matrix
q_1	Number of rows in the scanning window for Forward Backward Spatial smoothing
q_2	Number of columns in the scanning window for Forward Backward Spatial smoothing
$\mathbf{Q}_{\mathrm{n}}$	Noise subspace of the eigen vectors matrix
$\mathbf{R}$	Range steering matrix
R	Range
$\mathbf{r}$	Range steering vector
R_l	Range to the l:th target
$R_{\max}$	Maximum range

$R_{\min}$	Minimum range
$\mathbf{S}$	The array containing radar cubes (4:th order tensor)
$\mathbf{s}$	A vector of complex radar samples
s	Slope of the transmitted chirp
$\mathbf{S}_z$	The z:th radar cube, i.e. the z:th element in the first dimension of $\mathbf{S}$
$\mathbf{s}_z$	A slice of the vector $\mathbf{s}$ that corresponds to the z:th radar frame
$\mathbf{S}'$	An array containing conditionally modified radar cubes
$\mathbf{S}'_z$	The z:th radar cube in $\mathbf{S}'$
T	Transpose
t	A point in time
T_{c}	Chirp cycle time, i.e. time between two chirps
T_{d}	Detection threshold
T_{dwell}	Radar dwell time: $T_{\mathrm{dwell}} = N \div N_{\mathrm{FPS}}$
T_{e}	Ending time
$t_{\max}$	Maximum time
$t_{\min}$	Minimum time
T_{RTT}	Round-trip time
T_{s}	Starting time
$\mathbf{U}_{\mathrm{R}}$	Diagonal matrix with $\mathbf{u}_{\mathrm{R}}$ along its diagonal
$\mathbf{u}_{\mathrm{R}}$	Vector of coefficients for attenuating the range power spectrum
$\mathbf{U}_{\mathrm{v}}$	Diagonal matrix with $\mathbf{u}_{\mathrm{v}}$ along its diagonal
$\mathbf{u}_{\mathrm{v}}$	Vector of coefficients for attenuating the velocity power spectrum
$\mathbf{V}$	Range-azimuth steering matrix
v	Velocity
v_{err}	The velocity required to cause an error
$v_{\max}$	Maximum velocity
$v_{\min}$	Minimum velocity
$\mathbf{W}$	Additive white Gaussian noise matrix
W	The horizontal resolution of a camera
w	The column of a pixel in a IR, RGB, or Depth video frame
Y_0	Noise power
Y_1	Signal power

Z	Sum of the cell averaging window in CA CFAR algorithm
z	The index number of a frame

1. INTRODUCTION

As the power of computer grows and capability of sensors increases, more and more incredible things are becoming possible. One such thing is HAR (Human Activity Recognition), for which the applications are many.

Not only is it possible to detect when people are running or walking [1], but even respiratory rates [2], falling down [3], shoplifting [4], operating room procedures [5], hand gestures [6], and certain illnesses [7] have been proven to be recognizable by sensors. The possibilities seem almost limitless.

For long, the focus in HAR research has been in visible spectrum video. While it is undeniably a powerful sensing mechanism for the purpose, it has multiple major downsides. Visible spectrum imaging is very prone to occlusion and lighting conditions. In addition, due to face recognition, there are genuine concerns about the privacy issues related to visible spectrum sensing. [8] Many actions can also look very similar depending on the viewing angle. With additional sensors, extra information can be used to better distinguish otherwise similar action from each other.

Wearable sensors are a well-established mechanism human activity recognition [9]. This is already being used in multiple commercial applications, such as exercise and sleep recognition in smart watches. The downside of wearable sensors is that they can be cumbersome and unfashionable require willful equipping and rarely can be connected to mains. Clearly, if high-performance remote sensing can be achieved with a reasonable price, it is the superior alternative.

In addition to visible spectrum imaging, remote sensors include depth imaging (stereo camera), infrared imaging, acoustic sensing (microphones) and electromagnetic sensing (e.g. radar). Different kinds of proximity sensors, such as magnetic switches, pressure sensors, temperature sensors, and electrostatic proximity sensors can also be used, although their installation may be more labour-intensive [8].

It has been shown that depth imaging can achieve at least similar performance in activity recognition as visible spectrum imaging, while simultaneously preserving privacy [10]. In complex scenes, using depth imaging can increase the performance of activity recognition substantially [11]. Depth imaging still suffers from many of the same problems as visible spectrum imaging, most importantly obstruction.

Radar devices are capable of sensing through visual obstructions and have been demonstrated to achieve very good performance at HAR, especially when operating on the mmWave (millimeter-wave) spectrum. Another upside for radar imaging is that unlike cameras, they are not susceptible to lightning conditions. [12] As a downside, they are active devices in the sense that they must also have an active transmitter. Although passive radars exist, the sensing performance is compromised. WiFi signals and channel state information have also been demonstrated to be an effective alternative to radar sensing in environments where WiFi is available [8].

Another alternative for visible spectrum imaging is the infrared camera. Because the measured quantity is emitted directly from the human body, it is capable of operating even in zero-light conditions where a normal visible-spectrum camera would fail.

It has been demonstrated that even with as low as 8×8 pixel resolution infrared cameras, recognizing some activities has been proven possible. In some cases the recognized activities have been very simple, such as sitting, standing, or lying on ground [13]. With some additional context information, more complex household activities can also be recognized [14].

Sensor fusion can be leveraged to provide extra context information for improving human activity recognition [15, 16]. For example, an infrared sensor may detect that a person is lying in a bed. Sound information from an acoustic sensor and respiratory rate information from a radar sensor could be used to tell whether the person is sleeping or not.

Machine learning is one of the key techinologies used for HAR. For training machine learning models, it is necessary to have large quantities of data available. Unfortunately, the number of data sets that considers sensor fusion in HAR context is quite low [8]. The number of available data sets is brought even lower when only the data sets that consider remote sensors are included.

Remote sensing may very well be the future of HAR. For this reason, it was considered valuable to create a portable multi-modal sensing system that can be used for recording data sets. As the product of this thesis, such a system was created. The sensors installed in the systems were a combined visible and depth spectrum camera, an 8×8 pixel infrared camera, a 4×4 channel microphone and a 60 GHz radar.

The created system is a key enabler of research considering sensor fusion and remote sensing in HAR. Due to the portable and low-labour install of the system, it can be used to record data sets in various environments and with multiple data modalities with ease. Surely, the data sets that will be recorded with the system will push forward the boundaries of HAR.

Chapter 2 will describe the sensors used in the system in more detail and establish requirements for the developed system. In Chapter 3, the implementation of the system

will be detailed on an architectural level. Details of the source code will not be discussed. The source code of the project is publicly available for viewing on GitHub [17]. The data formats produced by the recording system will be documented in Chapter 4 along some data processing examples. The quality and performance of the system will be briefly discussed in Chapter 5, and finally, Chapter 6 will conclude the thesis.

2. PREMISE

The model developed by White [18] and later refined for HAR by Fu et al. [8] can be used to categorize sensors and data modalities recorded by them. To produce a versatile data set, multiple data modalities should be included. This requires the inclusion of multiple devices in the array. The categorization model is presented in Figure 2.1.

The sensors used for the system, with the recorded modalities and highlight colour in parentheses, were the following:

- Texas Instruments IWR6843ISK + DCA1000EVM (radar, cyan),
- Panasonic Grid-EYE (infrared, yellow),
- Intel RealSense D435i (RGB-D video, red), and
- MiniDSP UMA-16 (acoustic, green).

The modalities recorded by the sensors are highlighted in different colours in Figure 2.1. From the figure, it can be observed that three out of five sensor categories are present, and from the present sensor categories all but three modalities can be recorded. Most of the excluded sensors are rather laborious to install in contrast to the included sensors. Capacitive sensing only works on ranges up to half a meter, ambient temperature sensing provides very minimal information about human activities, and magnetic sensing requires a large number of sensors compared to the useful information gained from them. The excluded acoustic sensors could provide useful information with very little installation cost. WiFi based electromagnetic sensing could also potentially bring more value to the system, but radar can detect movements with a much finer resolution and the radar devices are perhaps easier to interface with. [8] All in all, most of the interesting data modalities are recorded by the sensors in the system, with some room for improvement.

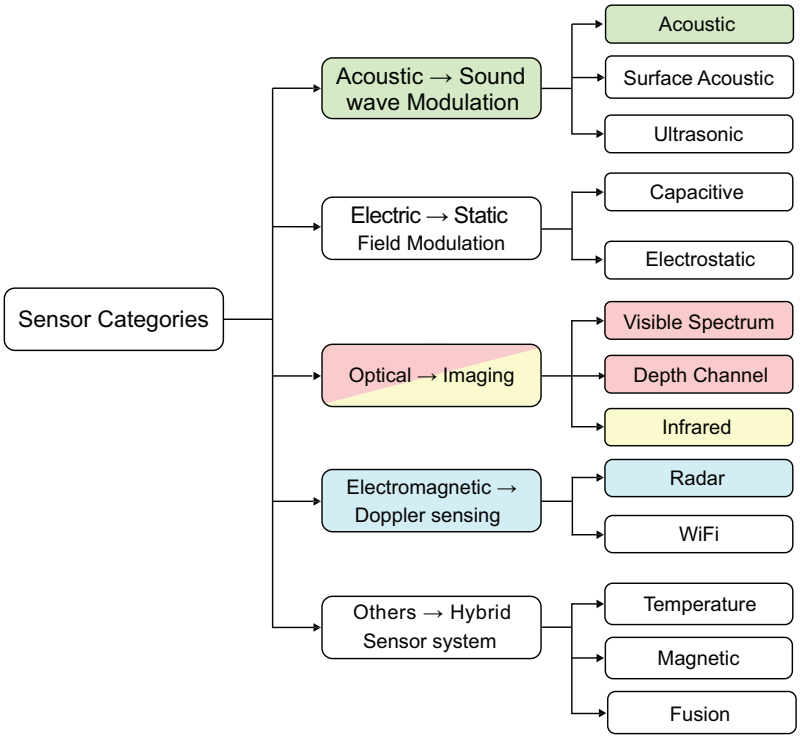


Figure 2.1. *Sensor categorization model by Fu et al. [8]. Data modalities present in the data collection system of this thesis are highlighted in different colours.*

The sensors are mounted on a single steel plate, giving them an uniform perspective to the target. The mounting bracket can be attached to a standard camera tri-pod. The system is very portable as the sensors can be connected via USB to a computer, and only two of the devices require an external power source: USB cannot deliver enough power for the radar device, and the microphone requires a 12-volt input voltage.

The mounting bracket and the positions of the sensors on it are illustrated in Figure 2.2. The mounting bracket is drawn in white. The blue outline represents the components of the radar device, yellow outline represents the infrared camera, red outline represents the RGB-D camera, and the green outline represents the microphone. The deeper-coloured circles represent the screw holes the sensors were mounted on. The white circle represents the tri-pod attachment point.

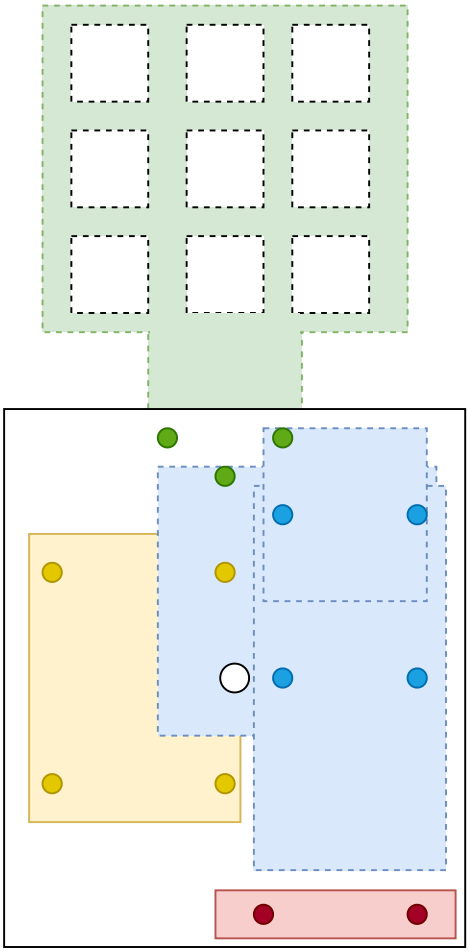


Figure 2.2. *The sensor array illustrated, not in scale.*

With the sensors chosen, before a data collection campaign can ensue, software needs to be written to interface with the sensors and to record the data produced by them. The contribution of this thesis in the data collection project is to develop this software.

The following sections (2.1–2.3) will discuss the operating characteristics, connection interfaces, and recorded data modalities of the included sensors. Finally, Section 2.4 will discuss the requirements and evaluation criteria for the data collection software developed for this thesis.

2.1 Radar

The radar used in the assembly is a Texas Instruments IWR6843ISK mmWave radar evaluation board. It is capable of outputting various continuous radar signals. It operates on 60–64 GHz frequency and has a maximum of 120-degree horizontal FoV (Field of Vision) and 30-degree vertical FoV. It has a monostatic radar with distinct transmitting and receiving antennas mounted side-by-side. The receiving antenna is a uniform linear array that consists of four antennas with 1/2-wavelength separation. The transmitting antenna

consists of three antennas, also separated by 1/2-wavelength. The transmitting antenna is non-linear with the middle antenna slightly raised. The antennas are illustrated in figure 2.3.

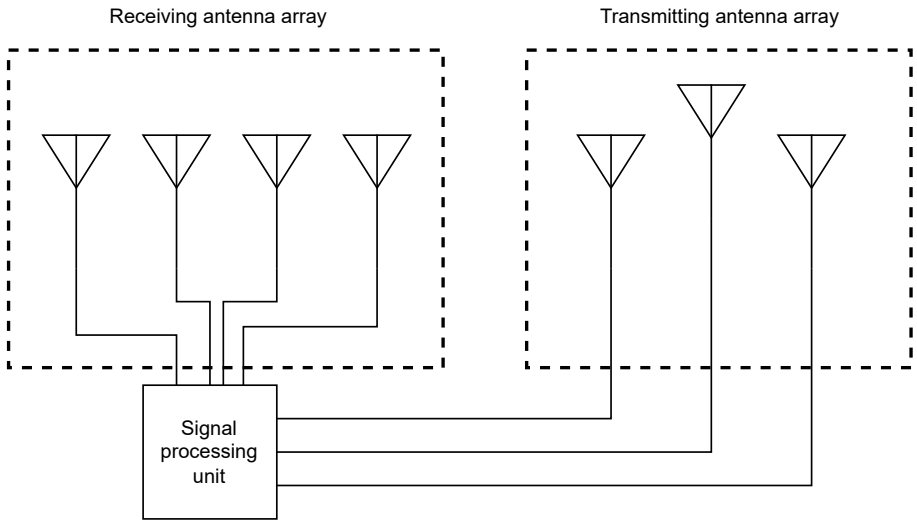


Figure 2.3. *Antenna arrays illustrated. More detailed documentation is available in the device user guide [19].*

When multiple active receivers are used, the location of the target can be detected in all three dimensions by applying angle and distance estimation algorithms, such as the 2D-MUSIC (2-Dimensional Multiple Signal Classification) and 2D-FFT (2-Dimensional Fast Fourier Transform) algorithms (Sections 4.3.2–4.3.4). The receivers can be toggled on and off from a special configuration file. The same configuration file can be used to define nearly arbitrary signal formats [20].

The IRW6843ISK is capable of processing the radar samples into radar frames on-board, but but depending on the operating parameters, its performance is limited to around 1–4 frames per second. To get more performance out of the device, the DCA1000EVM is used to interface with the IWR6843ISK to record the raw samples without processing them on-board. The processing is instead done separately by external software to form the radar frames.

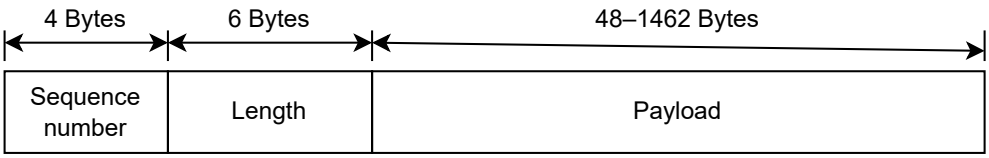


Figure 2.4. *The structure of the data frames output by the DCA1000EVM.*

The DCA1000EVM outputs the radar samples in a rather complicated format, that is documented in detail in the user guide for the device. The frames output by the device

depend on its configured operating mode. In this system, the device is configured in such a way that the frames consist of a 4-byte sequence number, 6-byte length field and a 48–1462-byte payload (Figure 2.4) [21].

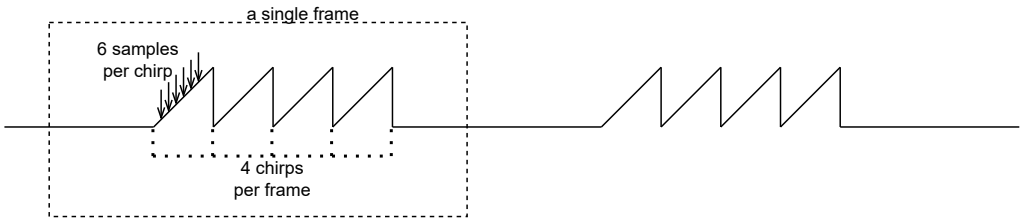


Figure 2.5. *Illustration of samples, chirps, and frames. With 2 receiving antennas, the equation 2.1 would give* $N_{\text{total}} = 4 \cdot 6 \cdot 4 \cdot 2 = 192$ *bytes per frame. In a practical situation, the numbers would, of course, be much higher.*

The data field of the frames consist of 2-byte complex samples, that can be arranged to 4-byte I/Q samples [22], i.e. four bytes of data are output for each sample. The number of samples in a single radar frame N_{total} can be calculated as given by equation 2.1, N is the number of samples per chirp, M is the number of chirps per frame, and K is the number of active receiving antennas. Figure 2.5 illustrates the formula. The number of bytes per radar frame is $4N_{\text{total}}$.

$$N_{\text{total}} = 4NMK, \tag{2.1}$$

As the frames output by the DCA1000EVM do not correspond exactly to the N_{total}-byte radar frames, the data segments should simply be concatenated in the order indicated by the sequence number until a full frame can be stored or processed. The developed data collection software saves the data segments on disk as-is, and the exact format of the data segments is presented in more detail in Section 4.3.

The radar sensor has proven to be effective in HAR. Unlike optical and acoustic sensors, radar is insensitive to environmental factors such as weather, lightning and acoustic noise. It is also capable of sensing through most (non-conductive) walls, making it less prone to occlusion. Using mmWave frequencies, radar is capable of detecting movements even in the sub-millimeter range.

After processing, various information can can be extracted from the data:

- target range, azimuth and altitude angle (Section 4.3.2),
- range-velocity spectrum (Section 4.3.3), and
- radar cross-section.

From HAR perspective, the first three items of the list are the most interesting, especially the doppler-spectrum (time-frequency spectrogram) [8].

The radar can effectively detect various activities based on the Doppler-spectrum [23–26]. Additionally it can very effectively measure the location of a target, which can be used as additional information for other sensors. As a downside, radar devices are very expensive and power-consuming. This can limit its usefulness in low-power or low-budget applications.

2.2 Optical

The optical sensors consist of conventional visible spectrum imaging (RGB video), depth imaging, and infrared imaging. The system in this thesis includes sensors that can record all these modalities. The Intel RealSense D435i is capable of both visual spectrum and depth imaging, while the Panasonic Grid-EYE records the IR (Infrared) spectrum.

Visible spectrum video is one of the best studied areas in computer vision and HAR. The primary approach to HAR from RGB video is using convolutional neural networks. Models such as AlexNet [27], and C3D have proven to be effective, the latter reaching as high as 90 % accuracy in detecting human actions [28].

Using a stereo camera, depth information can be extracted. This allows for much simpler segmentation and pose estimation compared to RGB video. [8] Combining the RGB and depth channels, detection accuracies as high as 98 % have been recorded on some data sets [29].

RGB video based segmentation and HAR in general can be problematic in some scenarios. If the image has poor contrast, proper segmentation may not be possible. In addition to unfortunate colors, this may be caused by bad lightning conditions, i.e. under- or overexposure or extreme fog. Visible spectrum imaging also raises some privacy concerns in certain environments.

Thermal cameras are capable of detecting the heat radiation emitted from warm objects. This makes them capable of detecting human motion from the background regardless of the lightning conditions [30]. Additionally, very low resolution (from 8x8 px to 16x16 px) IR sensors have been demonstrated to be capable of detecting human actions [31]. With a frame rate of 10 fps, detection accuracy of 85 % and above has been reached depending on the action [8, 32]. Although the detection accuracy is not as good as with the RGB-D camera, the possibility of operating in any lightning conditions and with high respect to privacy make the IR camera a viable choice.

2.2.1 RGB and Depth video

The RGB and Depth modalities are provided by an Intel RealSense D435i RGB-D camera. The D435i model uses a stereo camera aided by an infrared dot matrix to measure depth, making it similar in operating principle to the popular Microsoft Kinect.

The depth camera has a 87-degree horizontal FoV and a 58-degree vertical FoV. It can record up to 90 frames per second with up to 1280x720 pixel resolution. The ideal operating range is 0.3–3.0 meters and the depth measurement error is <2% at 2 meters. The RGB camera has a 69-degree horizontal FoV and a 42-degree vertical FoV. It can record at up to 1920x1080 resolution and up to 30 frames per second. [33]

Intel provides a library that can be used to interface with the camera programmatically. The library can be used natively from C++, and wrappers are provided for multiple languages and toolkits, perhaps most importantly Python, Matlab, and ROS (1 and 2). [34] The Python wrapper was used in the data collection software. It provides methods for getting the RGB-values, depth-values and time stamps of each frame [35].

2.2.2 Infrared

The IR camera used in the assembly is the Panasonic GRID-EYE (part no. AMG8834). It is an infrared camera with a resolution of 8-by-8 pixels. It has both vertical and horizontal FoV of 60 degrees, making each pixel 6-by-6 degrees. It can operate in temperatures of 0–80 °C, and measure temperature with a resolution of 0.5 °C, while the absolute temperature accuracy is 2.5–3.0 °C. According to the manufacturer, it can produce data at the rate of either 1 or 10 frames per second. [36] While developing the data collection software, it was measured that the actual frame rate was approximately 8.62 frames per second, though.

The evaluation kit outputs data in 135-byte segments, where the first 3 bytes are header bytes, followed by 130 bytes of data, and 2 bytes of tail (padding). The first two bytes of the data represent the temperature of the internal thermistor of the device (ambient temperature) and the following 128 bytes represent the IR camera matrix packed in row-major order. Temperatures are represented as multiples of four with little-endian two-byte integers. [37] The real temperature is therefore the unpacked number divided by four.

2.3 Audio

The microphone used in the assembly is a MiniDSP UMA-16, which is a 4-by-4 uniform rectangular array of microphones. It is capable of sampling at up to 48 kHz. Other possible sampling rates are 8, 11.025, 12, 16, 32, and 44.1 kHz. The microphone uses 24-bit quantization.

The microphone connects to the computer via USB, and can be interfaced with like any other microphone. Multiple convenience libraries exist for interfacing with microphones via code. Examples include SoundDevice (Python) [38], Simple DirectMedia Layer 2.0 (C, C++, C#, Python) [39], and PortAudio (C, C++) [40].

2.4 Data recording software requirements

To produce a high-quality data set, some requirements are also set for the software that ties the sensors together. In addition to giving guidelines for developing the data collection software, having defined requirements also allows evaluating the quality of the product.

As stated earlier in this chapter, the sensor array could be improved by including some different sensors. Therefore, the software must be extensible. The latency between data generation and application of time stamps must also be minimized, which calls for parallelism. In addition to software design choices, there are some requirements for the data it produces. The data shall:

- be time-synchronized,
- be labelled,
- be well-structured, and
- include sufficient metadata.

Time synchronization, in the case of the data set, means that the beginning of the data in each recorded modality corresponds to the same point in time. In other words, given a common starting time T_s, the first data point should correspond to T_s in each recorded modality. In addition, the recorded data for any modality shall not end before a common ending time T_e. Given a known frame rate N_FPS, satisfying this requirement allows easily mapping any moment in time (t) to a frame number (z) (equation 2.2).

$$z = \lfloor N_\mathrm{FPS} t \rfloor \tag{2.2}$$

For the data to be useful in supervised learning, the current activity at any given time must be included in the data set; the data must be labelled. Labelling can be done in various ways.

The most accurate labelling method is manual labelling: someone goes over the data and applies appropriate labels to each activity. Although this results in superior accuracy, it is very laborious and therefore not very desirable.

The activities could also be sourced from some sensors and algorithms that are already capable of recognising activities with a high accuracy. As previously stated, there are

numerous algorithms that can recognize activities from RGB-D video. The RGB-D video provided by the sensor array could therefore be used to map labels for the activities on the time domain. Using equation 2.2, the labels could then be generalized for the whole data set.

In addition to time synchronization and labels, the data must be structured in such a way that the data produced by each sensor in a single recording can be mapped to one another and the corresponding metadata and labels. It would additionally be beneficial to be able to pick and choose the sensors that are processed. The simplest way to achieve this is to produce one file per sensor and store them in a common directory. Another way would be to pack the data from different sensors into a single file, but this would be rather complicated especially because the recorded data is heterogenous in length.

3. SYSTEM IMPLEMENTATION

The system was implemented completely using Python (3.10), which is one of the most widely known programming languages [41]. Multiple libraries exist for Python for implementing a wide variety of tasks. Using high-level libraries is expected to save development time and the popularity of Python is expected to make the source-code easily approachable for future programmers.

To minimize the latency in reading the data from the sensors, the system was designed to use parallelism. The system implements a producer-consumer pattern, using the Python built-in Multiprocessing library.

The Multiprocessing library is the best of the python built-in parallelism libraries for the task at hand. The program is synchronous, hence Asyncio is perhaps a too complex solution [42]. While the Threading library would otherwise be a good solution, it is not truly concurrent as the Python GIL (Global Interpreter Lock) locks its thread model to a single process [43]. Multiprocessing, instead, implements process-level parallelism and is capable of achieving "true" concurrency [44].

The structure of the program is presented in Figrure 3.1 as a block diagram. The program consists of a Main block and five sub-blocks:

- Recorder,
- Radar,
- RGB-D,
- Infrared, and
- Microphone.

The producer-consumer pattern is implemented with all the different blocks being created as parallel processes. The Radar, Microphone, Infrared and RGB-D modules fetch data from the sensors, making them producers. The producers output data to queues which are then read by the Recorder block, hence the Recorder is a consumer. The main block is neither a producer or a consumer; rather it is responsible for allocating resources for the sub-blocks, spawning the sub-block processes (subprocesses), controlling them, providing a user interface and writing the metadata file.

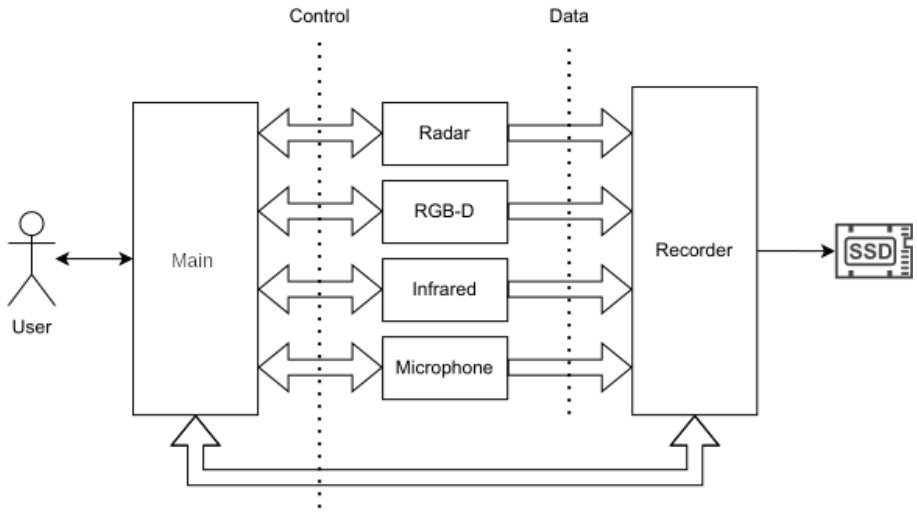


Figure 3.1. *Block diagram of the system.*

Inter-process communication is done via FIFO (First In First Out) queues. Each subprocess is connected to the main process via a single queue. These queues are colloquially called signaling queues. In addition, the producers are connected to the Recorder via queues, one queue per producer. These queues are called the data queues. The main process creates all these queues and passes them to appropriate subprocesses.

Queues are presented in Figure 3.1 as single- or double-headed arrows. A double-headed arrow means that the queue can be written to and read from in both ends, and single-headed arrow means that data can be read from the pointy end and written from the other. The signaling queues are read from and written to from both ends, whereas the data queues are only written to by the producers and read from by the consumer.

While using multiple queues introduces some complexity to the system, it prevents having to deal with race-conditions. Another option would have been to use only a single Data queue and to lock it for writing via mutexes in each of the producers. The locking operation can block if the mutex is locked from somewhere else, which could add latency to the data reading operations, making the multiple queues -approach a better solution.

The control queues are used by the Main process to advance the subprocesses to their next phase. The subprocesses write to the control queues to indicate their state. Three kinds of control messages are defined: "START", "STARTED", and "STOP". The control messages are simply Python String objects with defined content and meaning. The control messages are documented in Table 3.1.

Table 3.1. *Control messages in the program.*

Message	Sender	Meaning
START	Main	Recipient should start its primary function
STARTED	Producers	Sender has initialized and entered its primary function
STOP	Main	Recipient process should exit

The flow of the processes consists of three phases: initialization, primary function, and cleanup. The Main process and the data producers enter the primary function automatically after the initialization phase is over. The data producers write the "STARTED" message to the control queue after this transition. The Recorder process instead waits for the "START" message before transitioning from initialization to primary function. None of the subprocesses enters the cleanup phase autonomously. They all wait for the Main process to write the "STOP" message to the control queue before transitioning. The phases of the subprocesses are discussed in more detail in Sections 3.1–3.5.

The initialization phase of the Main process consists of parsing command line arguments, spawning the subprocesses, and waiting for the "STARTED" messages. After receiving the "STARTED" message from each data producer, the Main process sends the "START" message to the Recorder process and enters its primary function, which is a user interface loop.

The user interface is a simple interactive command line interface that can be used for semi-automatic data labelling and stopping the program. A file containing activity labels is provided via the command line arguments to the program. The program reads activities separated by a newline character into a list, and each time the user hits the Enter key, the time elapsed since sending the "START" message is recorded. The activity and timestamp are then written into a file and the list iterator is advanced. When either the list ends, the current activity label is "STOP", or the user writes the "q" character and then presses Enter, the primary function of the Main process ends, and the user interface loop is exitted.

After ending the primary function, it records the stopping time and sends the "STOP" signal to the data producers. After sending the "STOP" signal, the Main process waits for the data producer processes to exit. Then it sends the "STOP" signal and the current time to the Recorder process and waits until the process exits. After all the subprocesses have exitted, the Main process writes the metadata file and exits, ending the program. The flow of the main process is represented as a flowchart in Figure 3.2.

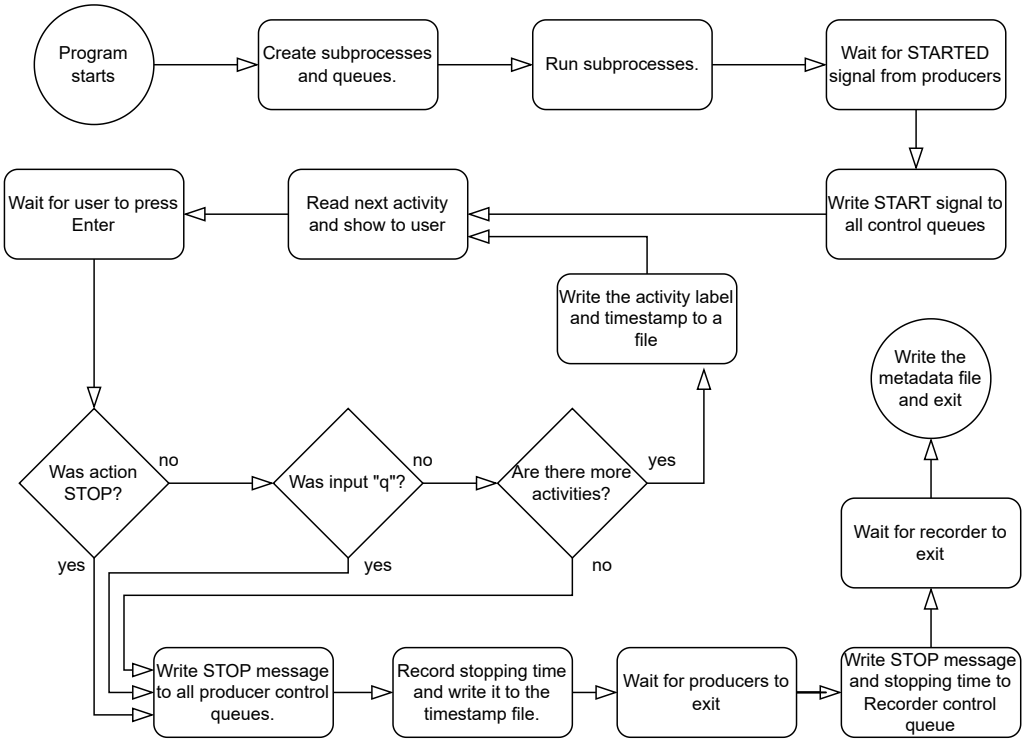


Figure 3.2. *Flow of the main process.*

The following Sections (3.1–3.5) will document the implementation of the subprocesses. For the data producers, communication with the sensors will be documented, and for the Recorder, the time-domain synchronization mechanism will be documented. The three phases of operation will be described similar to the former description of the Main process. Section 3.6 will briefly cover usage of the software.

3.1 Recorder block

The Recorder module is the only consumer in the program. It takes the data produced by the producers and writes it into a file. It is also responsible for time-domain synchronization of the different data streams.

The initialization phase of the Recorder block consists of opening file handles for the output files and initializing some counters. In the beginning of the second phase, i.e. after receiving the "START" message, the Radar block finds the common starting point for each data producer and synchronizes the data streams to the common starting point. After the data streams are synchronized, the Recorder enters a loop in which it simply reads the data from each data queue and writes it onto a disk.

Time synchronization is achieved in the Recorder block by observing the time stamps that the producers add to the packets they write in data queues. Each packet is a dict object that contains a segment of data and the time stamp when the data segment was recorded.

By the time the "START" message is received from the control queue, all the data producers have started producing data and have written their first message to the data queues. After the "START" message, the Main process also writes the current time to the control queue of the Radar block. The common starting point is found by comparing the time stamps in the data queues to the starting time provided by the Radar block. The data in each data queue is simply discarded until the timestamp is within half-a-frame from the starting time. After the appropriate amount of data has been discarded from each queue, the data streams are synchronized and the Recorder starts writing the data into files. The data starting point synchronization procedure is presented in pseudocode in Listing 14. The half-a-frame accuracy is omitted from the listing for brevity.

```
while control_queue.pop() != "START":
    pass
start_time = control_queue.pop()

for queue in data_queues:
    message = queue.pop()
    timestamp = message["timestamp"]
    data = message["data"]

    while (timestamp < start_time):
        message = queue.pop()
        timestamp = message["timestamp"]
        data = message["data"]
```

Listing 3.1. *Start of data synchronization pseudocode.*

Finally, when the user stops the system, the Main process sends the "STOP" message to the Recorder control queue, and in succession, the stopping time . By this time, the data producers have already exitted and no more data will be written to the output queues. The Recorder then enters the last phase of operation, which is the stream end synchronization.

The stopping time is recorded before the "STOP" signal is sent to the data producers. Knowing this, finding the common ending point is trivial, and the algorithm presented in Listing 14 can be modified for the purpose. Instead of the starting time, the time stamps are compared to the ending time, and instead of discarding the data, it is written into a file. The modified algorithm is presented in Listing 19.

```
while (control_queue.empty() or control_queue.pop() != "STOP"):
    for queue in data_queues:
        message = queue.pop()
        data = message["data"]
        # write data to file
```

```
end_time = control_queue.pop()

for queue in data_queues:
    message = queue.pop()
    timestamp = message["timestamp"]
    data = message["data"]

    while (timestamp < end_time):
        message = queue.pop()
        timestamp = message["timestamp"]
        data = message["data"]
        # write data to file
```

Listing 3.2. *End of data synchronization pseudocode.*

After the end synchronization procedure is finished, the Recorder will flush the data queues discarding all remaining data, and close the file handles. After freeing the resources and flushing the queues, the process will exit.

3.2 Radar block

The radar module actually consists of two devices: the Texas Instruments IWR6834ISK and the DCA1000EVM Radar Data capture card. In this section, the former is referred to as the "radar device" and the latter as the "capture card". The radar device connects to the computer via USB and implements a simple serial interface. The capture card uses Internet Protocol version 4 to connect to the computer and transfers data over the User Datagram Protocol.

In the Radar block, the Pyserial third-party library is used to connect to the serial interface. The Pyserial library implements an easy-to-use high-level serial port API (Application Programming Interface). [45] The serial port was configured to use a baud rate of 115200 bps and no parity.

For reading data from the capture card, the built-in Socket library is used. The Socket library implements the Berkeley Socket interface. By default, the capture card is listening at address `192.168.33.180:4096` and sends data to address `192.168.33.30:4098`. The addresses could be configured differently, but it was deemed easier to just configure an extra IPv4 (Internet Protocol version 4) address to the host computer. [21]

In the initialization phase, the Radar block configures the radar device and the capture card. Configuring the radar device is done by writing a special configuration file to the serial interface. The exact format of the configuration file is documented in the Texas Instruments mmWave SDK user guide [20]. The configuration file used during the testing

and development of this system is included as Appendix A.

The capture card is configured by transmitting special configuration packets to the address it is listening to. The packets consist of a header (`0x055A`), a two-byte unsigned integer length field, command identifier, body, and a footer (`0xEEAA`). The commands are listed in the DCA1000EVM user guide [21] and the details of the command bodies are documented in DCA1000EVM CLI Software Developer guide, which is included in the Texas Instruments mmWave studio collection of tools [46]. For the sake of availability, the commands and their bodies are also documented in Appendix B. While most settings are left to default values, the LVDS mode is changed to 2lane and Data format mode is changed to 16-bit. This is done via the CONFIG_FPGA_GEN_CMD_CODE command (Appendix B, Section B.3).

After configuration is done, the devices must also be started. The radar device can be started by writing "`sensorStart\n`" to the serial interface. The capture card is started by sending the RECORD_START_CMD_CODE (Appendix B, Section B.5). The capture card should then respond with status 0, indicating that the recording has been successfully started, thus concluding the initialization phase.

After starting the radar device and the capture card, the Radar module writes to the control queue the message "STARTED" and begins its primary function phase. In the primary function, the Radar module listens to the socket `192.168.33.30:4098` and reads data from it whenever available.

The Radar process can read the number of samples per frame from the radar configuration file. With this information, it is possible to organize the received data into frames and write them into the data queue frame-by-frame. The inter-frame time is also known from the configuration file. The radar device or the capture card do not add any timing information to the outputted data. The Radar process therefore records the current time when the first byte of the first frame is received, i.e. the starting time. The starting time is then used to calculate time stamps for the frames. For each received frame, a counter is incremented. The time offset of the frame is calculated from the counter and the inter-frame time and added to the starting time to form the time stamp for the frame.

The process simply reads data from the socket to a buffer until the buffer contains a single frame. When enough data is in the buffer, a dict-object containing the data of the frame and a time stamp for the frame is written into the data queue. The frame is then deleted from the beginning of the buffer and the loop is continued until the "STOP" message is received. The flow of the Radar module is represented as a flow chart in Figure 3.3.

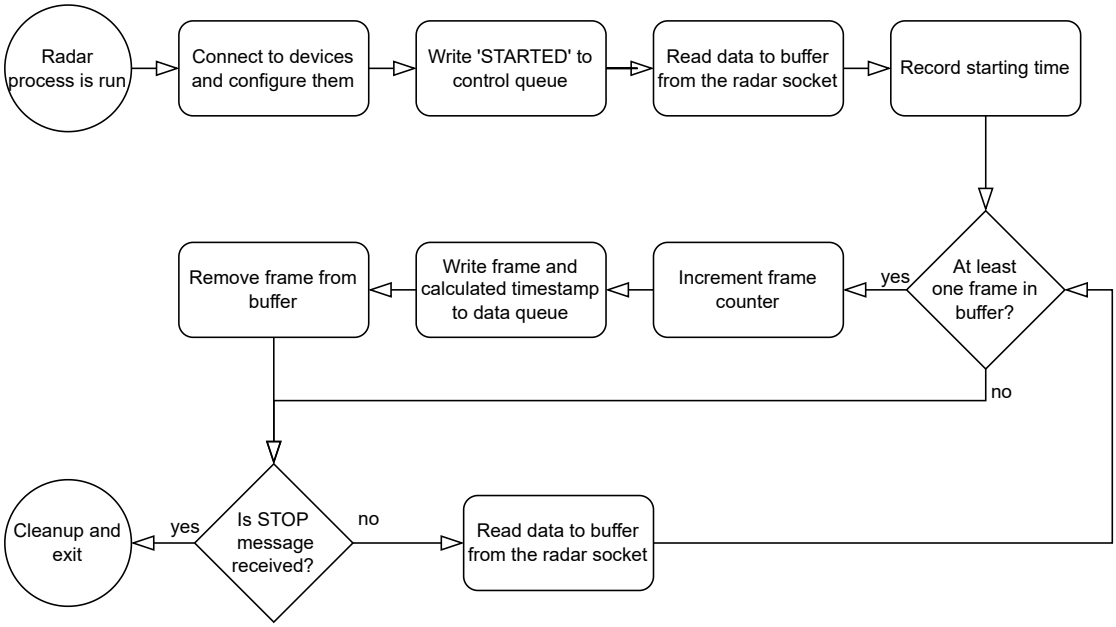


Figure 3.3. *Flow of the Radar module.*

After the "STOP" message is received, all the sockets and serial connections are closed. Additionally, the remaining data from the buffer is discarded and the data queue is closed from being written to. Thus, the clean up is concluded and the Radar process exits.

3.3 RGB-D block

Intel provides the Librealsense library for working with the camera with multiple programming languages. For this implementation, the Pyrealsense2 library was used, which implements Python bindings for the Librealsense library. [34] The Librealsense provides a high-level API for configuring the device and reading data from it. Configuring is done via a `Config` object and reading data is done via a `Pipeline` object. [35]

In the initialization phase of the RGB-D process, the `Config` and `Pipeline` objects are created. The used data streams must be explicitly enabled via the `Config` object. The depth and color streams are enabled. The functions that enable the streams also configure the frame rate, resolution, and number format for the data. In this implementation, the resolutions and frame rates are read from a configuration file provided by the user. The colored stream is configured to output data in the RGB8 format, where each pixel is represented by three bytes. The bytes are one-byte unsigned integers. The first byte is the red value, second byte is the green value and third byte is the blue value. The depth stream is configured to output data as 2-byte floating point numbers, where each pixel, or a 2-byte floating point number, represents the distance from the camera in the direction of the pixel.

After configuring the data streams, the `Pipeline` object is used to start the sensor. After

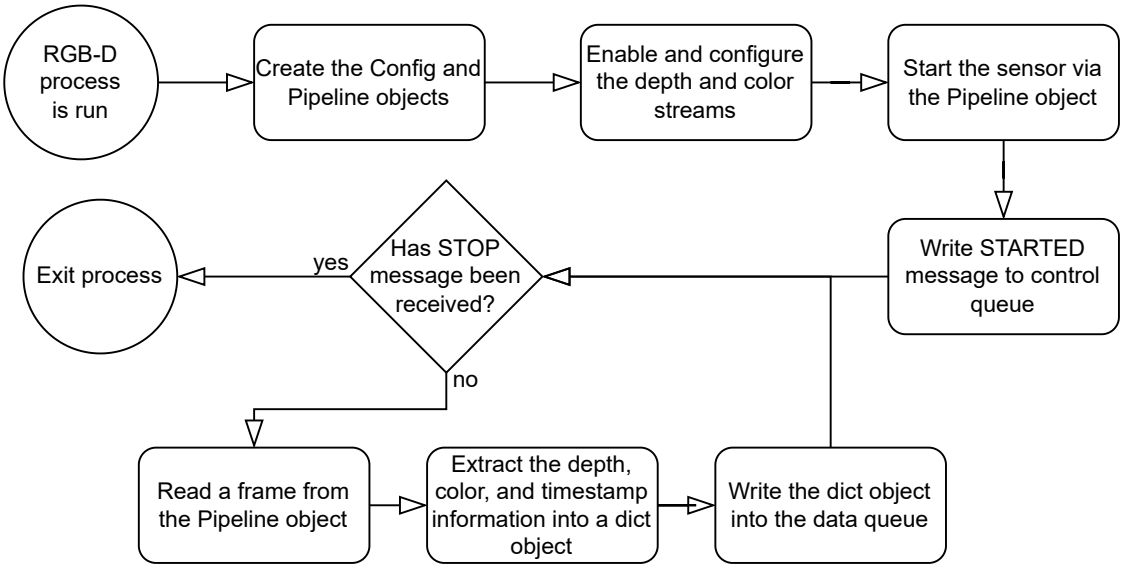


Figure 3.4. *Flow of the RGB-D process.*

the sensor has been started, the process writes the "STARTED" message to the control queue and enters its primary function.

In the primary function, the process reads frames from the `Pipeline` object using a member function of the Pipeline object that blocks until a frame is available. The function call returns a `composite_frame` object that contains the stream data, a time stamp, and some other less interesting metadata. The RGB-D process reads the color stream data and the depth stream data into arrays and stores them in a `dict` object along the time stamp for the frame, which is also read from the `composite_frame`. The `dict` object is then written into the data queue and the process is repeated until the "STOP" message is received from the control queue.

When the "STOP" message is received, the RGB-D process transitions into its cleanup phase. In the cleanup phase, the process simply exits, which calls the destructors of the Pyrealsense2 objects, which in turn stop the sensors and free their allocated resources. Figure 3.4 illustrates the flow of the RGB-D process.

3.4 Infrared block

Similar to the TI IWR6843ISK, the Panasonic GRID-EYE implements a simple serial interface over USB. Pyserial was also used for connecting to the GRID-EYE. Two kinds of packets are implemented for the GRID-EYE protocol: Sensor Data frames and Command frames. The Command frames are used by the connecting computer to configure the device and after being started, the device will write Sensor Data frames to the serial port. The Command frames can be used to set the sensor into either 10 FPS (Frames Per Second) for 1 FPS mode. [37] The sensor is used in the 10 FPS mode in the system,

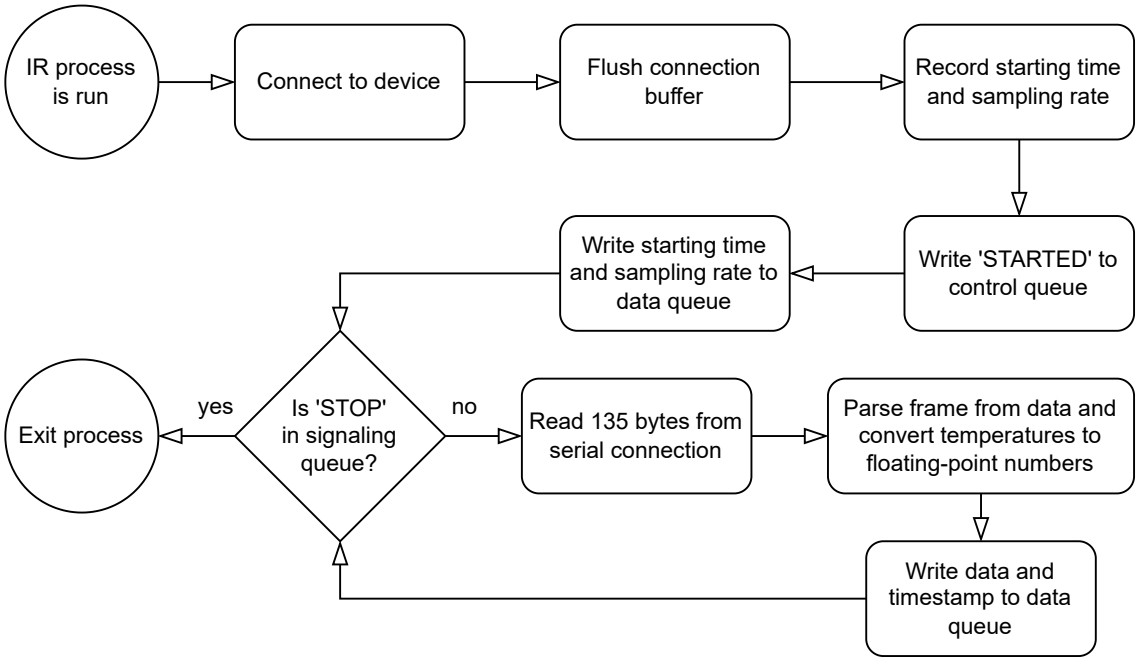


Figure 3.5. *Flow of the IR process.*

because the 1 FPS mode is only the sum of every 10 frames and similar effect can easily be achieved by post-processing the data after recording [36].

In the initialization phase, the IR process opens the serial port (9600 bps, no parity, one stop bit, 8-bit byte size). The device requires no further configuration as it operates in the 10 FPS mode by default. After opening the serial port, the data in the serial port is flushed, and the "STARTED" message is written to the control queue, concluding the initialization phase.

After the initialization phase, the process immediately begins the primary function. In the primary function, the process reads data from the serial port in 135-byte chunks. The 5th–132th (0-indexing) bytes contain the temperatures for each of the pixels as two-byte little-endian integers. Every two bytes is a multiple of four of the temperature reading.

The Python built-in Struct library is used to unpack the raw data to numeric values. Conveniently, two-byte floating point numbers can accurately represent integer multiples of 0.25. Using this knowledge, the unpacked numbers are divided by four and repacked as two-byte floating point numbers. The repacked floating point numbers are then stored in a `dict` object with a time stamp that is sourced from the computer clock using the `perf_counter` function from the Python built-in Time library. The process is then repeated until the "STOP" message is received from the control queue.

After receiving the "STOP" signal, the clean-up phase begins. In the clean up, the process simply exits, which frees the serial port file handle. No other resources were allocated in the process. The flow of the system is illustrated in Figure 3.5.

3.5 Microphone block

Unlike the other devices, the data produced by the microphone cannot be organized into frames. It produces a continuous stream of data on all 16 channels at a configurable sampling rate and bit depth. The SoundDevice library is used for interfacing with the device [38]. It provides Python bindings for the PortAudio library, which, among other features, has the ability to pick a sound device, configure it to a wished sampling rate and bit depth, and record audio from it [40].

Picking a device is done via the `query_devices` function, which returns information about available devices. It accepts two arguments. The first argument is a numeric device ID or the name of the device as a string. If the first argument is given, only the information of one device is returned. The second parameter can be used to list only input or output devices. [38].

The SoundDevice library implements an API, where `Stream` objects are created and attached to audio devices. The `Stream` objects can then write to and read from devices. A callback function may be passed to the constructor of the `Stream` object, which is periodically called automatically by the PortAudio library. The callback function handles writing data to the device and provides access to the read data. [47]

The callback function may also be omitted, in which case reading and writing should be done via the blocking `read` and `write` functions. Using the callback function is the preferred way of using the interface, as the callback function automatically has a high processing priority. To achieve robust audio with minimal latency, the callback function should not call functions with long or unpredictable execution times. [47]

After the `Stream` object has been instantiated, the stream must also be started. This can be done via `start` and `stop` functions. The stream objects are also context managers and when used in the `with`-statement, the `start` and `stop` functions are automatically called in the beginning and end of the statement, respectively. [47] Context managers and the `with` statement are a part of the Python programming language.

The initialization phase of the Microphone module finds the correct sound device and constructs and `InputStream` context manager, which is a specialized `Stream` object that can only be used for input devices. Searching the device is done via the `query_devices` function with the first argument set to "micArray16". The device is then set as the default device in the SoundDevice library. Upon instantiation, the `InputStream` object will utilize the default device without further configuration.

The configuration is done in the constructor of the `InputStream` object. The sampling rate is set to 44.1 kHz, bit depth to 32 bits and recording is done on all 16 channels of the device. After instantiating the `InputStream` object, the "STARTED" signal is written to the

control queue and the primary function is entered, ending the initialization phase.

The primary function is a simple busy-loop that observes the control queue. The process runs until the "STOP" message is read from the control queue. While waiting for the "STOP" message, the callback function is being periodically called by the PortAudio library. The callback function writes data from the device to the data queue along the timestamp for when the callback was called. The timestamp is derived using the `perf_counter` function from the built-in Time library.

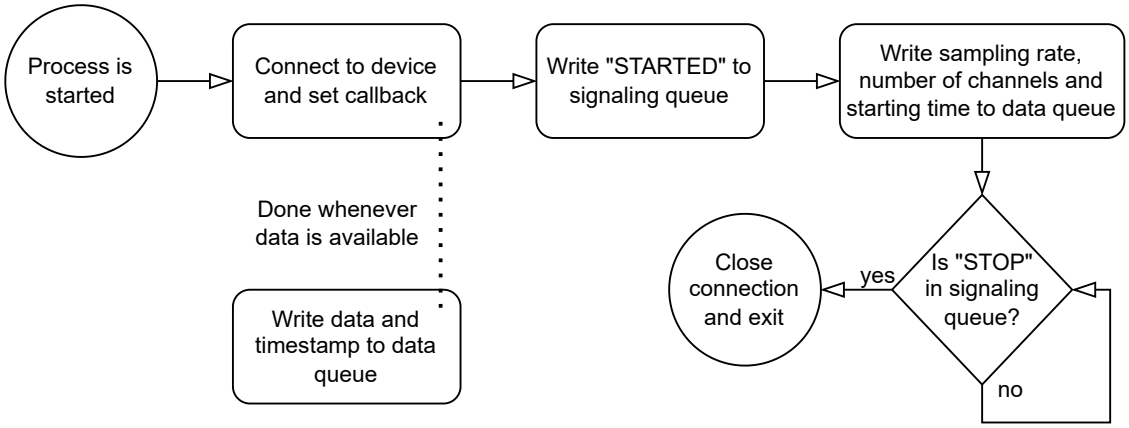


Figure 3.6. *Flow of the Microphone block.*

When the "STOP" message is received from the control queue, the aforementioned busy-loop exits, ending the `with`-statement and stopping the `InputStream` instance. The Microphone process then exits. The logic of the Microphone process is presented as a flowchart in Figure 3.6.

3.6 Using the software

The software is started via a CLI (Command Line Interface) that is implemented using the Python built-in `argparse` library. Three positional arguments must be provided for the program: `activities`, `config`, and `outdir`. All of the arguments are file paths.

The first argument, `activities`, can be used to provide a list of activities for the software. This can be used for semi-automatic activity labelling during the recording process. This is useful when recording a predetermined list of activities. The activities must be separated by newline (`\n`) characters. The software outputs a CSV (Comma Separated Value) file which contains the same list of activities, but each activity is assigned a timestamp. The file can also be empty, in which case no timestamps will be recorded other than the starting and stopping time.

The second argument, `config`, is a configuration file for the program. The values present in the file are used to configure the sensors. The file must follow the YAML data serialization

language. An example file is provided in Listing 6.

```
radar:
    config: filename
camera:
    resolution: [width, height]
    fps: integer
```

Listing 3.3. *Example configuration file.*

Under the key `radar`, the `filename` that is the value of `config` must refer to a configuration file for the TI IWR6843ISK radar device. Under the key `camera`, the value of `resolution` must be a two-element list whose values are some supported resolution of the Intel RealSense 435i camera. Similarly, the `fps` must be some supported frame rate of the camera. The values under the key `camera` are used for both the RGB and Depth stream. The IR and microphone devices should preferrably be configured via the same file but due to time constraints, they were left hardcoded.

The last argument, `outdir`, must be a directory path. The files created by the software are written into this directory. The output files and their exact formats are documented in Chapter 4.

4. OUTPUT FILE FORMATS AND POST-PROCESSING

The software outputs seven files: five sensor data files, a metadata file, and a file containing time stamps and labels. The file names and a general description of the content is documented in Table 4.1.

Table 4.1. *Files output by the software.*

File name	Description
audio.wav	Microphone samples
depth.raw	Depth camera record
ir.raw	IR camera record
metadata.yaml	Metadata
radar.raw	Radar samples
rgb.raw	RGB camera record
timestamps.csv	Activity labels and time stamps

The sensor data files contain unprocessed (raw) data outputted by the sensors; only the data in `ir.raw` has been repacked into a different number format. To extract meaningful data from the files, the file formats must be understood. Additionally, it may be beneficial to perform some post-processing for the data, especially in the case of the `radar.raw` file.

The following sections will document the exact formats of the file and present some relevant data processing algorithms for each data file format.

4.1 Activity labels and time stamps

The activity labels and their corresponding timestamps are stored in the `timestamps.csv` file. As implied by the file name, the file follows the CSV format, delimited by the comma character (",", unicode U+002C). The file contains two columns. The first column contains time stamps and the second column contains labels. Listing 7 provides an example of the file.

```
0.000506,sitting
1.689381,stand_up
4.106970,walking
7.184435,pick
9.060120,walking
12.494,STOP
```

Listing 4.1. *Example* `timestamps.csv` *file.*

The activities are sorted by time in an ascending order. The time stamps tell the time elapsed since the beginning of the recording. The frame rate (N_{FPS}) for each sensor is recorded in the `metadata.yaml` file. Based on the timestamps and the frame rate of each sensor, the mapping between frames and labels can be made. The time of recording (from $t_{\min}$ to $t_{\max}$) for a given frame can be calculated based on the frame number z via equation 4.1.

$$\begin{cases} t_{\min} = \frac{z}{N_{\mathrm{FPS}}} \\ t_{\max} = \frac{z+1}{N_{\mathrm{FPS}}} \end{cases} \tag{4.1}$$

Given $t_{\min} \leq t < t_{\max}$, where t is the timestamp of an activity, the activity should be mapped to the frame.

4.2 IR camera record

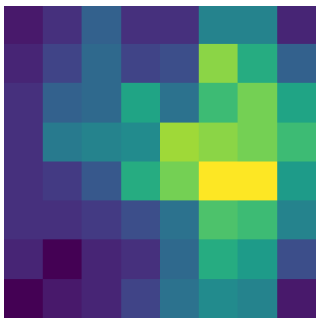
The frames produced by the infrared block consist of 64 numbers (8x8 pixels). The frames are stored in the `ir.raw` file. Each pixel is represented by a half-precision (16-bit) floating point number. The values represent the recorded temperatures in each pixel.

The pixels of each frame are stored in row-major order, thus denoting the vector containing the recorded values as $\mathbf{d}$, the frames can be represented by equation 4.2, where $\mathbf{F}_z$ is the z:th frame, h is the row of a pixel and w is the column of a pixel.

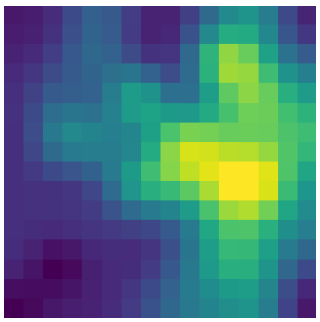
$$\forall h \in [0,7] \wedge w \in [0,7] \wedge z \in \left[0, \frac{|\mathbf{d}|}{64}\right] : \mathbf{F}_z(h,w) = \mathbf{d}(64z + 8h + w) \tag{4.2}$$

The origin of the image is in the upper left corner. Figure 4.1a shows an example frame parsed from the file and 4.1d shows the corresponding RGB image. The data in frame $\mathbf{F}_z$ may be used as-is, but the resolution of the image may be increased by interpolating. Figures 4.1b and 4.1c illustrate the effects of increasing the resolution of the original image

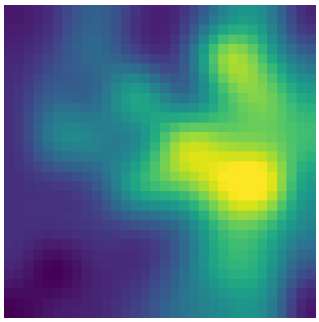
to 16x16 and 32x32 using cubic interpolation. A code example in Python for parsing the frames from the file is given in Appendix C.



(a) *Original 8x8 image.*



(b) *Original image upscaled to 16x16 by interpolation.*



(c) *Original image upscaled to 32x32 by interpolation.*

(d) *Corresponding RGB image.*

Figure 4.1. *An IR frame interpolated to different higher resolutions and the corresponding RGB frame.*

4.3 Radar samples

The Texas Instruments mmWave Radar Device ADC Raw Data Capture application report defines multiple data formats for the DCA1000EVM data output format, which is also the format the data is stored in the `radar.raw` file [22]. Combined with the TI6843ISK radar module, the data formats are limited to only one option [20]. The data is sampled in the I/Q (In-phase/Quadrature) format. Given there are N samples per chirp, M chirps per frame, and K active receivers, the data can be arranged to radar cubes and frames via the procedure explained in Section 4.3.1.

After the raw data has been rearranged into radar cubes and frames of complex data, numerous radar processing algorithms can be applied to the data to extract valuable

information about the targets. Algorithms for calculating the range-angle spectrum, range-velocity spectrum, and detecting targets are presented in Sections 4.3.2–4.3.4.

Throughout this section, the symbols listed in Table 4.2 are used. The values represented by the symbols are stored in the `metadata.yaml` file under the `radar` section. The keys that correspond to the values of the symbols and the meanings of the symbols, are also listed in the table.

Table 4.2. *Values that are recorded in in the `radar` section of the `metadata.yaml` file, their meanings, and corresponding symbols.*

Symbol	Key	Meaning
N	samples_per_chirp	Number of samples per chirp
M	chirps_per_frame	Number of chirps per frame
K	num_channels	Number of active receivers
T_c	chirp_cycle_time	Time between each frame [s]
N_FPS	framerate	Number of frames (radar cubes) recorded per second [Hz]
F_s	samplerate	Sampling frequency in the receivers [Hz]
S	slope	Slope of the transmitted chirp [Hz/s]

4.3.1 Radar file format

Based on the metadata, the raw samples in the `radar.raw` file can be rearranged into radar cubes. Each radar cube consists of $N \times M \times K$ complex samples as illustrated in Figure 4.2 Each complex sample in the raw data consists of two values: the in-phase and the quadrature component. Both components are represented as 16-byte integers in the raw data.

Denoting the raw 16-byte integer samples as $\mathbf{d}$, the real (in-phase) and imaginary (quadrature) components of the complex samples can be extracted with the following formulas:

$$\forall x \in \left[0, \frac{|\mathbf{d}|}{2}\right] : \begin{cases} \mathbf{d}_\mathrm{r}(x) = \mathbf{d}(4x) \\ \mathbf{d}_\mathrm{i}(x) = \mathbf{d}(4x+2) \end{cases} , \mathrm{mod}(n, 2) = 0 \\ \begin{cases} \mathbf{d}_\mathrm{r}(x) = \mathbf{d}(4x+1) \\ \mathbf{d}_\mathrm{i}(x) = \mathbf{d}(4x+3) \end{cases} , \mathrm{mod}(x, 2) \neq 0 \tag{4.3}$$

where $\mathbf{d}_\mathrm{r}(x)$ is the x:th element of the vector containing the real samples, and alike, $\mathbf{d}_\mathrm{i}(x)$ is the x:th element of the vector containing the imaginary samples. Thus, the vector

containing the complex samples s is given by equation 4.4, where j is the imaginary unit.

$$\mathbf{s} = \mathbf{d}_{\mathrm{r}} + j\mathbf{d}_{\mathrm{i}} \tag{4.4}$$

Each $N \times M \times K$ elements of s constitute a single radar cube. The samples for the z:th cube (or frame), denoted as $\mathbf{s}_z$, can be extracted from s as given by equation 4.5

$$\mathbf{s}_z = \begin{bmatrix} \mathbf{s}(zNMK) & \mathbf{s}(zNMK+1) & \dots & \mathbf{s}(zNMK+(NMK-1)) \end{bmatrix}^T \tag{4.5}$$

Each radar cube consists of $M \times K$ chirps, whereas each chirp consists of N samples. Each N samples in $\mathbf{s}_z$ constitutes for a chirp, thus the m:th chirp in $\mathbf{s}_z$, i.e. $\mathbf{c}_m$, is given by equation 4.6.

$$\mathbf{c}_m = \begin{bmatrix} \mathbf{s}_z(mN) & \mathbf{s}_z(mN+1) & \dots & \mathbf{s}_z(mN+(N-1)) \end{bmatrix}^T \tag{4.6}$$

The chirps are organized in such way in the data that each K chirps are sampled at the same time, but in different receivers. Thus, the frame $\mathbf{s}_z$ can be reshaped into a tensor $\mathbf{S}_z$ via the transformation given by equation 4.7.

$$\mathbf{S}_z = \begin{bmatrix} \mathbf{c}_0 & \mathbf{c}_K & \mathbf{c}_{2K} & \dots & \mathbf{c}_{(M-1)K} \\ \mathbf{c}_1 & \mathbf{c}_{K+1} & \mathbf{c}_{2K+1} & \dots & \mathbf{c}_{(M-1)K+1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{c}_{K-1} & \mathbf{c}_{2K-1} & \mathbf{c}_{3K-1} & \dots & \mathbf{c}_{MK-1} \end{bmatrix} \tag{4.7}$$

For each sample in $\mathbf{S}_z$ that has the same first dimension index has an equal sampling time. The sampling time increases by T_{c} for each index in the second dimension of $\mathbf{S}_z$ and by $1 \div F_{\mathrm{s}}$ for each index in the third dimension. The third dimension is the index of the vectors $\mathbf{c}_m$. The dimensions of the tensor $\mathbf{S}_z$ are therefore equal to the radar data cube presented in Figure 4.2.

Using the described transformations, the vector s can be transformed into a 4-dimensional tensor $\mathbf{S}$ given by 4.8. The Appendix D presents an example of parsing the tensor $\mathbf{S}$ from the `radar.raw` file using Python and Numpy.

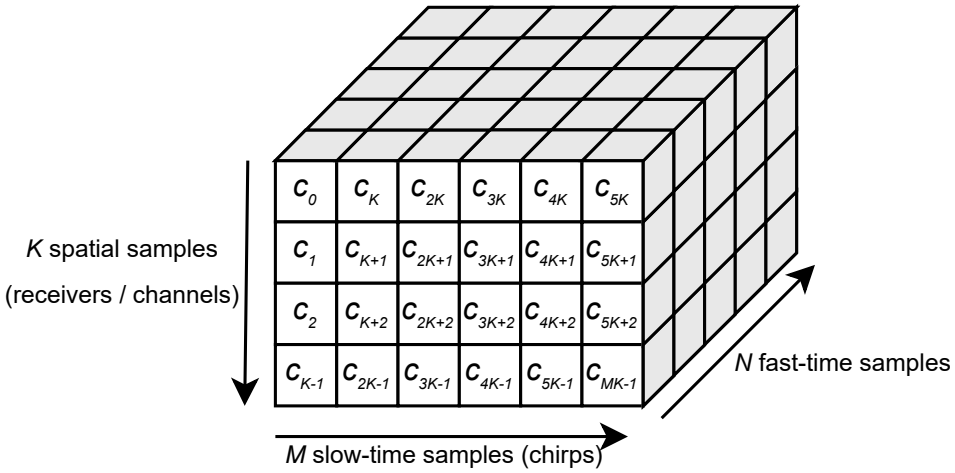


Figure 4.2. *Radar cube with dimensions corresponding to* $\mathbf{S}_z$*.*

$$\mathbf{S} = \begin{bmatrix} \mathbf{S}_0 & \mathbf{S}_1 & \mathbf{S}_2 & \dots & \mathbf{S}_{|\mathbf{s}| \div NMK} \end{bmatrix}^T \tag{4.8}$$

To turn the radar data cubes into useful information, multiple data processing algorithms can be applied to the data. Since most HAR methods for radar signals use the velocity spectrum, the most interesting information is the target positions and the corresponding velocity spectra [8]. The position information can be used to track the targets and get continuous time-Doppler data for a given target. Additionally, the position information can be used to aid in the microphone beam forming.

Section 4.3.2 briefly covers the methods for extracting the range-azimuth information from the radar samples. Section 4.3.4 briefly covers the methods for detecting and tracking targets from the data, and finally, Section 4.3.3 briefly covers the FFT (Fast Fourier Transform) based algorithm for extracting the Doppler-spectrum.

4.3.2 Range-azimuth spectrum

For extracting the range-azimuth data from the radar signals, 2D-MUSIC is one of the most attractive methods. Compared to the traditional FFT based methods, the 2D-MUSIC can achieve significantly better resolution in both the angular domain and the range domain [48]. A major downside of the 2D-MUSIC algorithm is extremely high computational load and memory usage. The use of FPGAs (Field-Programmable Gate Arrays) or ASICs (Application Specific Integrated Circuits) may be used in applications to make the computations faster.

The radar device used in the sensor assembly can be configured to switch between two transmitting antennas, such that odd chirps are transmitted on a different antenna than

even chirps. If this feature is used, the effective amount of spatial channels in the radar data is $2K$, and each two radar cubes can be combined along the first axis to form another data cube. [19] Based on this, a new tensor $\mathbf{S}'$ can be defined along a new variable K' that is the number of active receivers.

$$\begin{cases} \forall z \in \left[0, \frac{|\mathbf{s}|}{2NMK}\right] : \mathbf{S}'(z) = (\mathbf{S}(2z)|\mathbf{S}(2z+1)) \\ K' = 2K \end{cases}, \quad \text{if switching transmitters} \tag{4.9}$$

$$\begin{cases} \mathbf{S}' = \mathbf{S} \\ K' = K \end{cases}, \quad \text{othwerwise} \tag{4.10}$$

The tensor augmentation operation $(\mathbf{A}|\mathbf{B})$ is defined by the transformation $(\mathbf{A}^{K\times M\times N}, \mathbf{B}^{K\times M\times} N) \rightarrow (\mathbf{A}|\mathbf{B})^{2K\times M\times N}$, given by equation 4.11 where the magnitude of each element $\mid (\mathbf{A}|\mathbf{B})(k,m) \mid = N$.

$$(\mathbf{A}|\mathbf{B}) = \begin{bmatrix} \mathbf{A}(0,0) & \mathbf{A}(0,1) & \dots & \mathbf{A}(0,M-1) \\ \mathbf{A}(1,0) & \mathbf{A}(1,1) & \dots & \mathbf{A}(1,M-1) \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}(K-1,0) & \mathbf{A}(K-1,1) & \dots & \mathbf{A}(K-1,M-1) \\ \mathbf{B}(0,0) & \mathbf{B}(0,1) & \dots & \mathbf{B}(0,M-1) \\ \mathbf{B}(1,0) & \mathbf{B}(1,1) & \dots & \mathbf{B}(1,M-1) \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{B}(K-1,0) & \mathbf{B}(K-1,1) & \dots & \mathbf{B}(K-1,M-1) \end{bmatrix}, \tag{4.11}$$

Similar to $\mathbf{S}_z$, the z:th radar cube in $\mathbf{S}'$ can be denoted by $\mathbf{S}'_z$. Using these definitions, the 2D-MUSIC spectrum can be calculated. The first and third dimension of the tensor $\mathbf{S}'_z$ constitute for the samples recorded during a single dwell, i.e. a chirp. It can be represented by the tensor $\mathbf{D}_m$, given by equation 4.12. The part of the data cube represented by $\mathbf{D}_m$ is highlighted in Figure 4.3.

$$\mathbf{D}_m = \begin{bmatrix} \mathbf{S}'_z(0,m) & \mathbf{S}'_z(1,m) & \dots & \mathbf{S}'_z(K',m) \end{bmatrix}^T \tag{4.12}$$

Table 4.3. *Definitions of the symbols used in equation 4.13.*

Symbol	Definition
k	Index of an antenna in the receiving uniform linear array
n	Index of a fast-time sample in $\mathbf{c}_m$
L	Number of reflecting radar targets
α_l	Amplitude of the reflected signal from the l:th target
$e^{j\varphi}$	Phase of the reflected signal from the l:th target
R_l	Range from the receiver to the l:th target
B	The bandwidth of the chirp
F_s	Sampling frequency
c	The speed of light
T_dwell	Dwell time: $N \div N_\mathrm{FPS}$
θ_l	Angle of arrival of the signal reflected from the l:th target
$\omega(k,n)$	Additive white Gaussean noise in the n:th sample of the k:th receiver
d	The distance between adjacent receivers

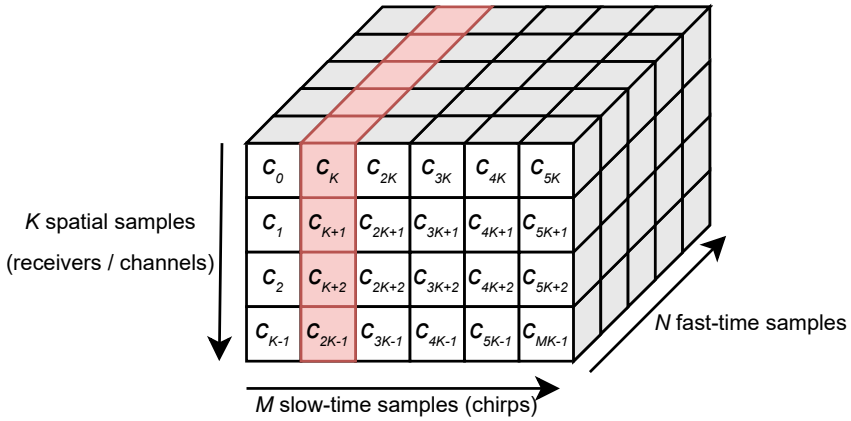


Figure 4.3. *The part of the radar data cube represented by the tensor $\mathbf{D}_m$ is highlighted in red ($m = 1$).*

Under the narrowband assumption, the samples of the tensor $\mathbf{D}_m$ can be modeled as given by equation 4.13 [49]. The symbols used in the equation are defined in Table 4.3.

$$\mathbf{D}_m(k,n) = \sum_{l=0}^{L-1} \alpha_l e^{j\varphi}\, e^{j2\pi \frac{2R_l B}{cT_\mathrm{dwell}N_\mathrm{FPS}} n}\, e^{j\frac{2\pi}{\lambda} dk \sin\theta_l} + \omega(k,n). \tag{4.13}$$

The tensor $\mathbf{D}_m$ may also be represented in matrix format as shown by the equations 4.14–4.19.

$$\mathbf{D}_m = \mathbf{AXR} + \mathbf{W} \tag{4.14}$$

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}(\theta_0) & \mathbf{a}(\theta_1) \dots & \mathbf{a}(\theta_{L-1}) \end{bmatrix}_{K \times L} \tag{4.15}$$

$$\mathbf{X} = \begin{bmatrix} \alpha_0\, e^{j\varphi_0} & & & \\ & \alpha_1\, e^{j\varphi_1} & & \\ & & \ddots & \\ & & & \alpha_{L-1}\, e^{j\varphi_{L-1}} \end{bmatrix}_{L \times L} \tag{4.16}$$

$$\mathbf{R} = \begin{bmatrix} \mathbf{r}(R_0) & \mathbf{r}(R_1) \dots & \mathbf{r}(R_{L-1}) \end{bmatrix}_{N \times L} \tag{4.17}$$

$$\mathbf{a}(\theta_l) = \begin{bmatrix} 1 & e^{j\frac{2\pi}{\lambda} d \sin\theta_l} & \dots & e^{j\frac{2\pi}{\lambda} d(K-1)\sin\theta_l} \end{bmatrix}_{1 \times K} \tag{4.18}$$

$$\mathbf{r}(R_l) = \begin{bmatrix} 1 & e^{j2\pi \frac{2R_l B}{cT_{\text{dwell}} F_{\text{s}}}} & \dots & e^{j2\pi \frac{2R_l B}{cT_{\text{dwell}} F_{\text{s}}}(N-1)} \end{bmatrix}_{1 \times N} \tag{4.19}$$

Equation 4.14 is equivalent to equation 4.13. In equation 4.14, $\mathbf{A}$ is the angle steering matrix consisting of the steering vectors $\mathbf{a}(\theta_0)$ ... $\mathbf{a}(\theta_{L-1})$. The matrix $\mathbf{R}$ is the range steering matrix which consists of the range steering vectors $\mathbf{r}(R_0)$ ... $\mathbf{r}(R_{L-1})$. The diagonal matrix $\mathbf{X}$ contains the complex amplitudes and phases of the reflected signals. Finally, the $\mathbf{W}$ is the additive white Gaussian noise matrix. Using this information, the 2D-MUSIC algorithm can be applied to calculate the range-azimuth spectrum.

The 2D-MUSIC algorithm is based on evaluation of the covariance matrix of the received signal and the separation of noise and target signal subspaces. Multiple sweeps (chirps) are typically used for evaluating the covariance matrix. The noise and target signal subspaces are estimated from the covariance matrix by applying eigenvalue decomposition to the covariance matrix. The resulting eigenvalues are then used to estimate the number of targets to separate the subspaces. The 2D-MUSIC spectrum is then estimated from the correlation of the noise subspace and the range-azimuth steering matrix given by equation 4.31. [48]

When the reflected signals are correlated, which is the case when a target spans multiple range or angle bins, the dimension of the signal subspace is not equal to the number of targets. This can be solved by applying smoothing techniques, such as FBSS (Forward-Backward Spatial Smoothing), to the data. [48, 50]

For a $K' \times N$ matrix, the FBSS algorithm is applied by defining a window with dimensions $q_1 \times q_2$, and then scanning the data matrix in all possible positions: $p_1 = K' - q_1$ positions in the angular dimension and $p_2 = N - q_2$ positions in the range dimension. For each scanning position, the sub-matrix is flattened in column-major order to form the vector $\boldsymbol{\delta}(\tilde{p}_1, \tilde{p}_2)$, where $\tilde{p}_1 \in [0, p_2)$ and $\tilde{p}_2 \in [0, p_2)$. The vectors $\boldsymbol{\delta}(\tilde{p}_1, \tilde{p}_2)$ are then stacked

column-wise to form the spatial-smoothed data matrix $\tilde{\mathbf{D}}_m$. The scanning procedure is illustrated in Figure 4.4. [48, 49]

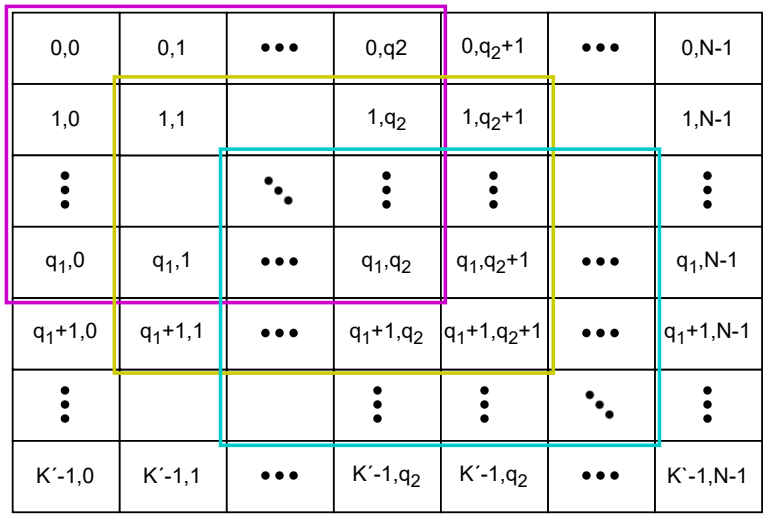


Figure 4.4. *Scanning window procedure for the data matrix. The matrix is scanned in all possible positions where the* $q_1 \times q_2$ *window can fit.*

$$\tilde{\mathbf{D}}_m = \begin{bmatrix} \boldsymbol{\delta}(0,0) & \boldsymbol{\delta}(1,0) & \dots & \boldsymbol{\delta}(p_2-1,0) & \boldsymbol{\delta}(0,1) & \boldsymbol{\delta}(1,1)\dots & \boldsymbol{\delta}(p_2-1,p_1-1) \end{bmatrix} \tag{4.20}$$

The resulting dimensions of $\tilde{\mathbf{D}}_m$ are $q_1q_2 \times p_1p_2$. The vectors $\boldsymbol{\delta}(\tilde{p}_1,\tilde{p}_2)$ are given by equation 4.21.

$$\boldsymbol{\delta}(\tilde{p}_1,\tilde{p}_2) = \begin{bmatrix} \mathbf{D}_m(\tilde{p}_1,\tilde{p}_2) \\ \mathbf{D}_m(\tilde{p}_1+1,\tilde{p}_2) \\ \vdots \\ \mathbf{D}_m(\tilde{p}_1+q_1-1,\tilde{p}_2) \\ \mathbf{D}_m(\tilde{p}_1,\tilde{p}_2+1) \\ \mathbf{D}_m(\tilde{p}_1+1,\tilde{p}_2+1) \\ \vdots \\ \mathbf{D}_m(\tilde{p}_1+q_1-1,\tilde{p}_2+q_2-1) \end{bmatrix}_{q_1q_2\times 1} \tag{4.21}$$

Having formed the smoothed data matrix $\tilde{\mathbf{D}}_m$, the data smoothed covariance matrix $\mathbf{C}_{\tilde{\mathbf{D}}_m}$ can then be evaluated as given by equation 4.22. The matrix $\mathbf{J}$ in equation 4.22 is the

transition matrix defined by equation 4.23. [48, 50]

$$\mathbf{C}_{\tilde{\mathbf{D}}_m} = \frac{1}{2p_1p_2}\left[\tilde{\mathbf{D}}_m{\tilde{\mathbf{D}}_m}^{\dagger} + \mathbf{J}(\tilde{\mathbf{D}}_m{\tilde{\mathbf{D}}_m}^{\dagger})^*\mathbf{J}\right] \tag{4.22}$$

$$J = \begin{bmatrix} 0 & 0 & \dots & 1 \\ \vdots & 0 & 1 & 0 \\ 0 & \iddots & 0 & \vdots \\ 1 & 0 & \dots & 0 \end{bmatrix}_{q_1q_2 \times q_1q_2} \tag{4.23}$$

To estimate the signal and noise subspaces, the mean of the covariance matrices of multiple sweeps is used. The final covariance matrix used for the eigenvalue decomposition is given by equation 4.24. [49]

$$\mathbf{C}_{\tilde{\mathbf{D}}} = \frac{1}{M}\sum_{m=0}^{M-1}\mathbf{C}_{\tilde{\mathbf{D}}_m} \tag{4.24}$$

The covariance matrix $\mathbf{C}_{\tilde{\mathbf{D}}}$ can be factorized as

$$\mathbf{C}_{\tilde{\mathbf{D}}} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{\dagger}, \tag{4.25}$$

where $\mathbf{Q}$ contains the eigenvectors of $\mathbf{C}_{\tilde{\mathbf{D}}}$ and $\mathbf{\Lambda}$ is a diagonal matrix containing the corresponding eigenvalues. Given the number of targets L (equations 4.27–4.29), the noise subspace $\mathbf{Q}_{\mathrm{n}}$ with dimensions $q_1q_2 \times p_1p_2 - L$ can be partitioned from $\mathbf{Q}$ as shown by equation 4.26 [48].

$$\mathbf{Q}_{\mathrm{n}} = \begin{bmatrix}\mathbf{Q}(L) & \dots & \mathbf{Q}(p_1p_2 - 1)\end{bmatrix} \tag{4.26}$$

The number of targets L can be estimated using information theoretic criteria, such as AIC (Akaike Information Criteria) or MDL (Minimum Description Length). As proposed by Wax and Kailath, equation 4.28 shows the formula for AIC for this problem and equation 4.29 shows the formula for MDL. For both equations, the number of targets is the argument of

the minimum for the criteria (Equation 4.27). [51]

$$L = \underset{0 \leq l < q_1 q_2 q_1 q_2}{\arg\min} \ \mathrm{AIC}(l) \quad \vee \quad L = \underset{0 \leq l < q_1 q_2}{\arg\min} \ \mathrm{MDL}(l) \tag{4.27}$$

$$\mathrm{AIC}(l) = -2 \log \left(\frac{\prod_{i=l}^{q_1 q_2 - 1} \Lambda(i,i)^{1 \div (q_1 q_2 - l)}}{\frac{1}{q_1 q_2 - k} \sum_{i=l}^{q_1 q_2 - 1} \mathbf{\Lambda}(i,i)} \right)^{(q_1 q_2 - l)K} + 2l(2q_1 q_2 - l) \tag{4.28}$$

$$\mathrm{MDL}(l) = -\log \left(\frac{\prod_{i=l}^{q_1 q_2 - 1} \Lambda(i,i)^{1 \div (q_1 q_2 - l)}}{\frac{1}{q_1 q_2 - l} \sum_{i=l}^{q_1 q_2 - 1} \Lambda(i,i)} \right)^{(q_1 q_2 - l)K} + \frac{1}{2} l(2q_1 q_2 - l) \log K \tag{4.29}$$

Having estimated the noise subspace, the 2D-MUSIC spectrum $\mathbf{P}(R, \theta)$ is given by equation 4.30. The range-azimuth steering matrix $\mathbf{V}(\theta, R)$ is given by equation 4.31, where $\mathbf{a}(\theta)$ and $\mathbf{r}(R)$ are given by equations 4.18 and 4.19 respectively. [49]

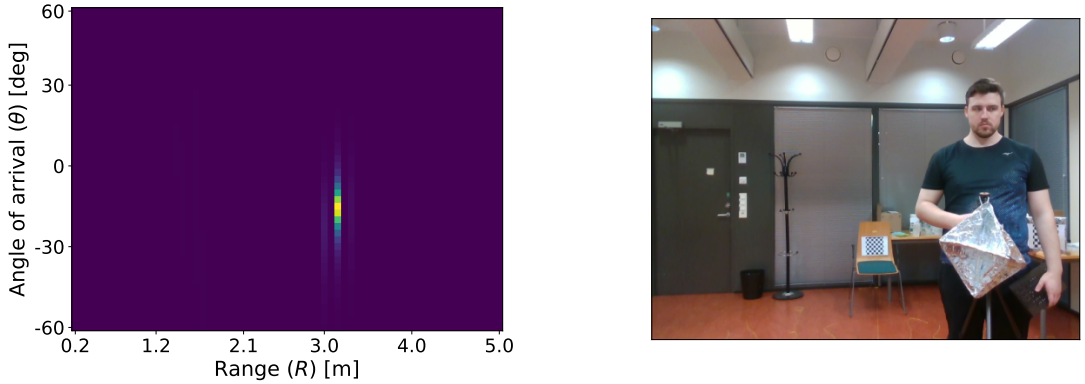
$$\mathbf{P}(R, \theta) = \frac{1}{\mathbf{V}(\theta, R)^{\dagger} \mathbf{Q}_{\mathrm{n}} \mathbf{Q}_{\mathrm{n}}{}^{\dagger} \mathbf{V}(\theta, R)} \tag{4.30}$$

$$\mathbf{V}(\theta, R) = \mathbf{a}(\theta) \otimes \mathbf{r}(R) \tag{4.31}$$

An example implementation of the FBSS algorithm is provided in Appendix E and an example implementation of the 2D-MUSIC algorithm is provided in Appendix F. Figure 4.5 shows an example graph of a 2D-MUSIC spectrum.

4.3.3 Range-velocity spectrum

When the range of a target changes by less than half-wavelength, or the propagation distance changes by less than one wavelength, the phase-shift in the received reflection is linearly proportional to the change in distance. Because the time between two samples is known, the change in phase between two samples is therefore proportional to the velocity



(a) *2D-MUSIC spectrum.* ***(b)*** *Corresponding RGB video frame.*

Figure 4.5. *2D-MUSIC spectrum and a corresponding RGB video frame.*

of the target. When the velocity of the target $v \ll \frac{cM}{fT_\mathrm{c}}$, the relation between the phase shift ($\Delta\varphi$) and velocity (v) is given by equation 4.32 (derived in Appendix H), where T_c is the chirp cycle time, f is the frequency of the transmitted signal and c is the speed of light.

$$\Delta\varphi = \frac{4\pi f v T_\mathrm{c}}{c} \tag{4.32}$$

The equation 4.32 could be used to create a velocity steering vector and the 2D-MUSIC algorithm could applied to acquire the range-velocity spectrum. The MUSIC (Multiple Signal Classification) algorithm is computationally very demanding, though. By taking multiple samples of the signal, a sufficient resolution can be had for the range–velocity spectrum with 2D-FFT method, which is computationally much faster.

Given the signal is sampled N times per each of the M chirps on K channels, the dimensions of the resulting data cube are $K \times M \times N$ (Section 4.3.1). For the 2D-FFT method, the range and velocity resolutions (ΔR and Δv respectively), the maximum range (R_max), and the minimum and maximum velocities ($v_\mathrm{min}$and v_max respectively) are given by equations 4.33–4.36 (derived in Appendix H).

$$\Delta R = \frac{F_\mathrm{s} c}{2Ns} \tag{4.33}$$

$$\Delta v = \frac{c}{2MfT_\mathrm{c}} \tag{4.34}$$

$$R_{\max} = \frac{F_s c}{2s} \tag{4.35}$$

$$v_{\min} = -v_{\max} = -\frac{c}{4fT_c} \tag{4.36}$$

The range-velocity spectrum is calculated by first averaging the data cube corresponding to the z:th frame ($\mathbf{S}_z$) along the first axis (receivers) to increase the SNR (Signal to Noise Ratio), which will result in the data matrix $\mathbf{D}$, as given by equation 4.37. The dimensions of $\mathbf{D}$ are $M \times N$.

$$\mathbf{D}(m,n) = \sum_{k=0}^{K-1} \mathbf{S}_z(k,m,n) \tag{4.37}$$

After calculating $\mathbf{D}$, the Fourier transform is performed on the second axis of the matrix, which is the fast-time samples. Denoting $\mathbf{D} = \begin{bmatrix}\mathbf{f}(0) & \mathbf{f}(1) & \dots & \mathbf{f}(M-1)\end{bmatrix}$, where $\mathbf{f}(m)$ is the vector containing the fast-time samples for the m:th chirp, the operation is given by equation 4.38.

$$\hat{\mathbf{p}}(m) = \mathcal{F}\{\mathbf{f}(m)\} \tag{4.38}$$

The operation $\mathcal{F}\{\cdot\}$ in equation 4.38 is the discrete Fourier transform. The resulting vector $\hat{\mathbf{p}}$ has contains the Fourier transform of $\mathbf{f}(m)$ as its m:th element. Each bin in the vectors $\hat{\mathbf{p}}$ corresponds to a range as given by equation 4.39. The function $R(n)$ gives the corresponding range for the n:th bin in vector $\hat{\mathbf{p}}(m)$ (Appendix H).

$$R(n) = n\Delta R = \frac{nF_s c}{2Ns} \tag{4.39}$$

Assuming the range of the target changes by less than half a wavelength between two chirps, the phase difference between the n:th element (frequency component) of vectors $\hat{\mathbf{p}}(m)$ and $\hat{\mathbf{p}}(m+1)$ is dictated by the distance the target has moved (Appendix H). Thus, denoting $\hat{\mathbf{P}} = \begin{bmatrix}\hat{\mathbf{p}}(0) & \hat{\mathbf{p}}(1) \dots & \hat{\mathbf{p}}(M-1)\end{bmatrix}^T$, the Fourier transform can be applied along the first dimension of $\hat{\mathbf{P}}$ to acquire the velocities of the targets. The range-velocity power-spectrum can thus be denoted as $\mathbf{P}'$ as given by equation 4.40, where $\mathbf{p}'(n)$ is the velocity

spectrum for the n:th range bin, as given by equation 4.41.

$$\mathbf{P}' = \begin{bmatrix} \mathbf{p'}(0) & \mathbf{p'}(1) & \dots & \mathbf{p'}(N-1) \end{bmatrix} \tag{4.40}$$

$$\mathbf{p'}(n) = \mathcal{F}\left\{ \begin{bmatrix} \hat{\mathbf{P}}(0,n) & \hat{\mathbf{P}}(1,n) & \dots \hat{\mathbf{P}}(M-1,n) \end{bmatrix}^T \right\} \tag{4.41}$$

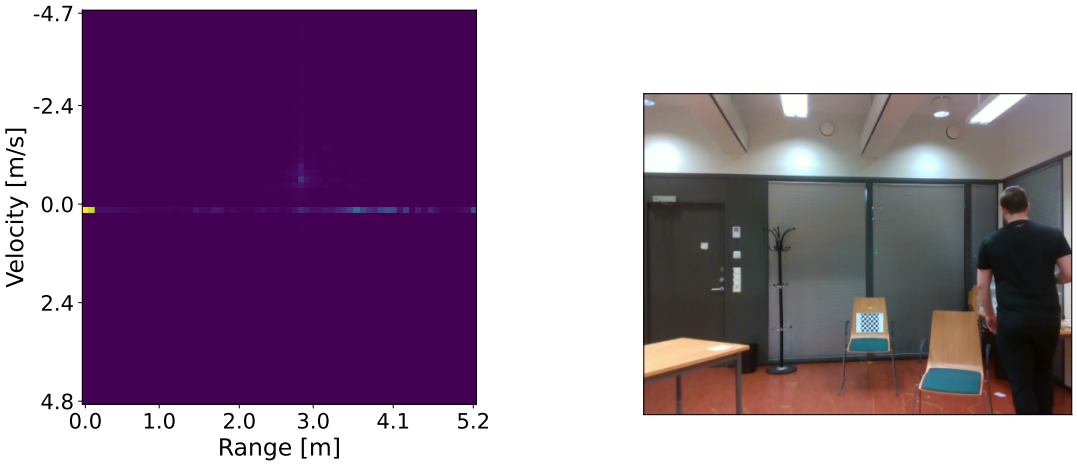
Because the velocity can be either positive or negative, the matrix $\mathbf{P}'$ still needs to be shifted so that the zero-velocity bin is in the center. The shifted range-velocity spectrum can be defined as $\mathbf{P}$ as given by equation 4.42.

$$\begin{cases} \forall m < \frac{M}{2} : \mathbf{P}(m,n) = \mathbf{P}'(m + \frac{M}{2}, n) \\ \forall m \geq \frac{M}{2} : \mathbf{P}(m,n) = \mathbf{P}'(m - \frac{M}{2}, n) \end{cases} \tag{4.42}$$

The velocity corresponding to the m:th bin on the first axis of $\mathbf{P}$ ($v(m)$) can be calculated from the minimum velocity and velocity resolution (equations 4.36 and 4.35). The velocity $v(m)$ is given by equation 4.43.

$$v(m) = -\frac{c}{4fT_\mathrm{c}} + \frac{mc}{2MfT_\mathrm{c}} \tag{4.43}$$

Therefore, the matrix $\mathbf{P}$ contains the range-velocity spectrum of the radar cube. The power of the reflection in element $\mathbf{P}(n,m)$ is the absolute value of the element, whereas the range and velocity are given by equations 4.39 and 4.43 respectively. Figure 4.6a shows an example range-velocity spectrum and Figure 4.6b shows the corresponding RGB image. The target is walking away from the sensor, thus the velocity is negative.



(a) *Range-velocity spectrum (2D-FFT).* ***(b)*** *Corresponding RGB video frame.*

Figure 4.6. *Range-velocity spectrum and the corresponding RGB video frame.*

It is seen from Figure 4.6a that there is a lot of power on the near-zero range and in the zero-velocity bins. The near-zero range power is caused mainly by self-interference in the radar and reflections of static nearby targets. Static clutter appears in the zero-velocity bins. The first few range bins should always be muted and if only moving targets are of interest, also the zero-velocity bins should be muted.

Muting the bins can be done by defining vectors $\mathbf{u}_\mathrm{R}$ and $\mathbf{u}_\mathrm{v}$ of magnitudes N and M respectively, where $\mathbf{u}_\mathrm{R}(n)$ is the amplification of the n:th range bin and $\mathbf{u}_\mathrm{v}(m)$ is the amplification of the m:th velocity bin. Corresponding diagonal matrices $\mathbf{U}_\mathrm{R}$ and $\mathbf{U}_\mathrm{v}$ can then be defined with the elements of $\mathbf{u}_\mathrm{R}$ and $\mathbf{u}_\mathrm{v}$ along their diagonals. Given a minimum range of interest R_min, the vectors $\mathbf{u}_\mathrm{R}$ and $\mathbf{u}_\mathrm{v}$ can be defined via equations 4.44 and 4.45. The only velocity bins that are muted are the ones corresponding to zero-velocity.

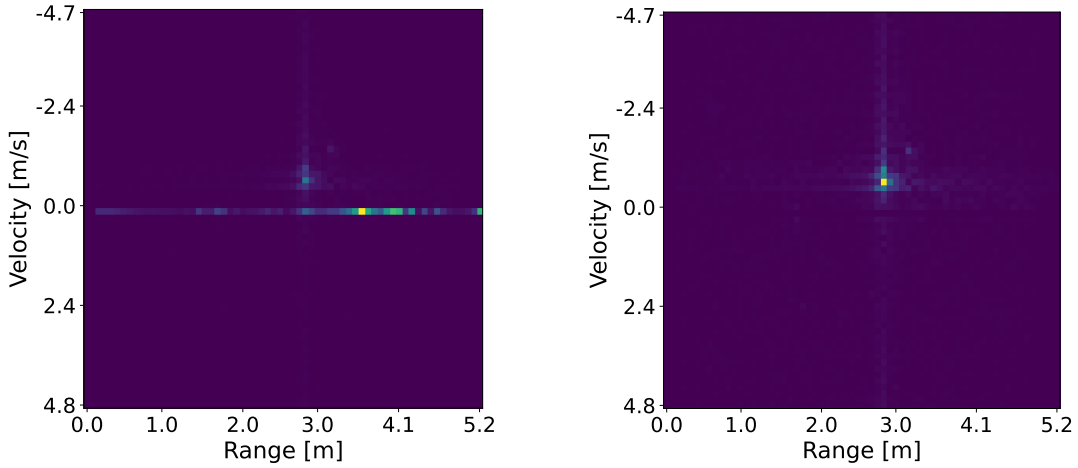
$$\begin{cases} \forall n \leq \left\lfloor \frac{R_\mathrm{min}}{\Delta R} \right\rfloor : \mathbf{u}_\mathrm{R}(n) = 0 \\ \forall n > \left\lfloor \frac{R_\mathrm{min}}{\Delta R} \right\rfloor : \mathbf{u}_\mathrm{R}(n) = 1 \end{cases} \tag{4.44}$$

$$\begin{cases} \forall m = \frac{1}{2}M : \mathbf{u}_\mathrm{v}(m) = 0 \\ \forall m \neq \frac{1}{2}M : \mathbf{u}_\mathrm{v}(m) = 1 \end{cases} \tag{4.45}$$

Having defined the matrices $\mathbf{U}_\mathrm{R}$ and $\mathbf{U}_\mathrm{v}$ according to equations 4.44 and 4.45, the muting of the unwanted bins is achieved by multiplying the matrix $\mathbf{P}$ with matrices $\mathbf{U}_\mathrm{R}$ and $\mathbf{U}_\mathrm{v}$ as given by equation 4.46. The Figure 4.7 shows the velocity spectrum from Figure 4.6a,

but with only the zero-range noise filtered in Figure 4.7a and both zero-range noise and zero-velocity clutter filtered in Figure 4.7b.

$$\mathbf{P}_{\text{filtered}} = \mathbf{U}_{\text{v}}(\mathbf{U}_{\text{R}}\mathbf{P}^{T})^{T} \tag{4.46}$$



(a) *Range-velocity spectrum with only zero-range noise filtered.*

(b) *Range-velocity spectrum with zero-range noise and zero-velocity clutter filtered.*

Figure 4.7. *Filtered range-velocity spectrum.*

An example implementation of the 2D-FFT algorithm for acquiring and filtering the range-velocity spectrum is presented in Appendix G. The example is written in Python.

4.3.4 Target detection and tracking

From the calculated spectra, targets can be detected using target detection algorithms. The most common way of detecting targets from the spectra is picking a threshold and checking if the power a cell is over it. The threshold be picked in such a way that is higher than the noise power to minimize false positives. The threshold must also be lower than the power reflected from targets in order to detect true positives. In other words, the threshold must be picked in such a way that the error rate is minimized, as illustrated by Figure 4.8.

Formally, the target detection is done via hypothesis testing, where H_0 is that the cell contains only noise and H_1 is that the cell contains noise superimposed with a target. When the power of the tested cell is denoted as $P_{n,m}$ and detection threshold as T_{d}, the hypothesis test is defined by equation 4.47.

	H_0 is accepted Pure noise	H_1 is accepted Noise supoerimposed with a target
$P_{n,m} < T_\mathrm{d}$	True negative	False negative Type I error
$P_{n,m} \geq T_\mathrm{d}$	False positive Type II error	True positive

Figure 4.8. *Different kinds of detection errors. The signal power in the test cell is denoted as $P_{n,m}$, whereas T_d is the detection threshold.*

$$\begin{cases} H_0 : P_{n,m} < T_\mathrm{d} \\ H_1 : P_{n,m} \geq T_\mathrm{d} \end{cases} \tag{4.47}$$

The threshold depends on parameters of the noise-only distribution. If the noise distribution was completely known prior to testing, picking a correct detection threshold would be a trivial task. In real-world applications the noise spectrum is space-time variant and therefore cannot be known a priori. This leaves two choices for picking the detection threshold: using distribution-free detection procedures [52] or estimating the noise spectrum and using an adaptive algorithm [53].

Typically a category of adaptive algorithms called CFAR (Constant False Alarm Rate) detection algorithms is applied for detecting the targets. The principle of CFAR algorithms is that the noise distribution is estimated from the calculated spectra (e.g. range-velocity or range-azimuth), whereof the probability distribution for noise power is estimated. The detection threshold is picked from the estimated noise probability distribution based on a predefined probability of a false alarm, thus giving a false alarm rate that is constant.

When the probability density function for noise power in cell y is denoted as $f_{Y_0}(y)$, the probability that H_0 is rejected is given by the Neyman-Pearson criterion (equation 4.48) [54, 55]. The criterion is illustrated by Figure 4.9.

$$P_\mathrm{fa} = \Pr(Y_0 \geq T_\mathrm{d}) = \int_{T_\mathrm{d}}^{\infty} f_{Y_0}(y) dy \tag{4.48}$$

When a square law detector is used for detecting targets in the signal, i.e. the power of the signal is used as decision criterion, and the noise in each cell is narrowband Gaussian noise, the probability distribution for noise follows the exponential distribution. Thus, the

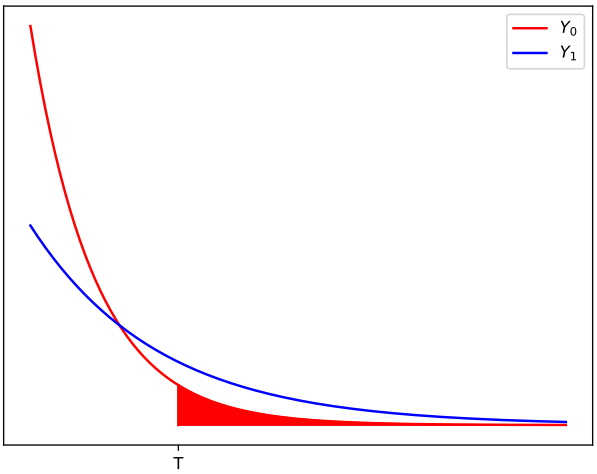


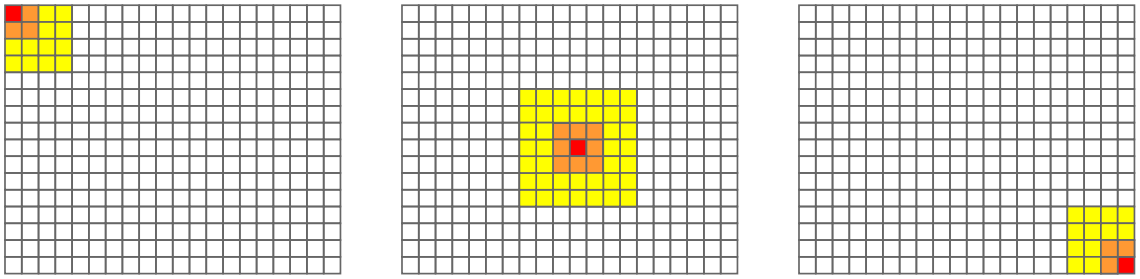
Figure 4.9. *The probability density functions for the noise power Y_0 and signal power Y_1. The highlighted area is the probability of a false alarm as given by equation 4.48 for given decision threshold T_d.*

decision threshold T_d is given by the quantile function of the exponential distribution (equation 4.49), where μ is the average noise power. [53]

$$T_\mathrm{d} = -\mu \cdot \ln(1 - P_\mathrm{fa}) \tag{4.49}$$

Different CFAR algorithms differ from each other in how the mean noise power μ is estimated. The CA (Cell Averaging) CFAR algorithm assumes that the noise power in each cell independent and identically distributed. Based on the central limit theorem, the average noise power may be estimated by defining a window with sufficiently large dimensions, and calculating the average power in the cells inside the window, disregarding the test cell and some guard cells around it. This method is based on the assumption that the average noise power is calculated from cells that have a very high likelihood of containing purely noise. The assumption is fair when the targets are sparse.

The estimated average noise power $\hat{\mu}$ around the cell (n, m) in the CA-CFAR algorithm is given by the equation 4.50. The term $Z(n, m)$ and $G(n, m)$ in the equation are the sum power of the $C_n \times C_m$ observation window around the test cell (equation 4.51) and the sum power of the $G_n \times G_m$ guard cells around the test cell (equation 4.54) respectively. The terms N and M are the dimensions of the $N \times M$ matrix that is the power spectrum $\mathbf{P}$, given by the equations 4.30 and 4.42. [54] The test cell and the surrounding windows are illustrated by Figure 4.10.



(a) $(n, m) = (0, 0)$ ***(b)*** $(n, m) = (N/2, M/2)$ ***(c)*** $(n, m) = (N - 1, M - 1)$

Figure 4.10. *The observation window, guard cells and test cell in the Cell Averaging Constant False Alarm Rate algorithm. The test cell is highlighted in red, guard cells in orange, and noise estimation window in yellow. In all cases, the noise estimation window has dimensions* 7×7 *and the guard window has dimensions* 3×3*.*

$$\hat{\mu}(n, m) = \frac{1}{C_n C_m - G_n G_m} \left(Z(n, m) - G(n, m) \right) \tag{4.50}$$

$$Z(n, m) = \sum_{c_\mathrm{n}=c_{\mathrm{n,min}}}^{c_{\mathrm{n,max}}} \sum_{c_\mathrm{m}=c_{\mathrm{m,min}}}^{c_{\mathrm{m,max}}} \mathbf{P}(c_\mathrm{n}, c_\mathrm{m}) \tag{4.51}$$

$$c_{\mathrm{n,min}} = \max\left(n - \lfloor C_n/2 \rfloor, 0\right) \quad c_{\mathrm{n,max}} = \min\left(N, n + \lfloor C_n/2 \rfloor\right) \tag{4.52}$$

$$c_{\mathrm{m,min}} = \max\left(m - \lfloor C_m/2 \rfloor, 0\right) \quad c_{\mathrm{m,max}} = \min\left(M, m + \lfloor C_m/2 \rfloor\right) \tag{4.53}$$

$$G(n) = \sum_{g_n=g_{\mathrm{n,min}}}^{g_{\mathrm{n,max}}} \sum_{g_m=g_{\mathrm{m,min}}}^{g_{\mathrm{m,max}}} \mathbf{P}(g_n, g_m) \tag{4.54}$$

$$g_{\mathrm{n,min}} = \max\left(n - \lfloor G_n/2 \rfloor, 0\right) \quad g_{\mathrm{n,max}} = \min\left(N, n + \lfloor G_n/2 \rfloor\right) \tag{4.55}$$

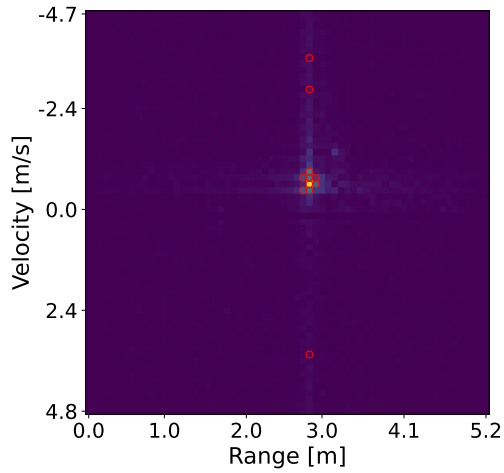
$$g_{\mathrm{m,min}} = \max\left(m - \lfloor G_m/2 \rfloor, 0\right) \quad g_{\mathrm{m,max}} = \min\left(M, m + \lfloor G_m/2 \rfloor\right) \tag{4.56}$$

It is noteworthy that the 2D-MUSIC algorithm (equation 4.30) does not produce a true power spectrum, but rather a pseudo-power spectrum where the cells contain a correlation number instead of actual signal power. Thus, the noise pseudo-power may not in reality be exponentially distributed and another distribution or a target detection algorithm is likely better suited for the task.

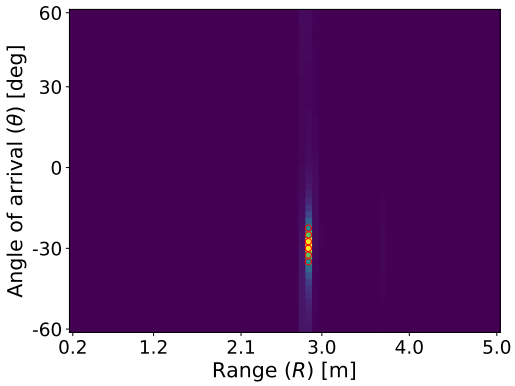
In a multitarget situation, the average noise power estimate of CA-CFAR is no longer a good estimate, as it is skewed from targets appearing inside the noise estimating window. The average noise level is thus estimated too high and weak targets are "masked" by nearby strong targets, which causes type II errors (H_0 is erroneously accepted). The OS (Ordered Statistic)-CFAR algorithm may be deployed in multi-target situations to achieve better detection performance. [56]

In the OS-CFAR algorithm, the powers of the cells inside the observation window are sorted in smallest-first order. The l:th element is then picked from the sorted array and

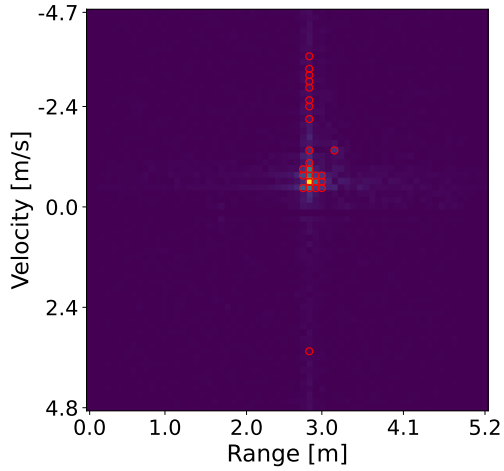
used as the average noise power estimate. Given the observation window has dimensions $C_n \times C_m$, based on the work of Rohling, a reasonable value for is $l = \frac{3}{4}C_nC_m$. In the OS-CFAR algorithm, the guard cells are omitted. [57] The figure 4.11 illustrates the results of applying the CA- and OS-CFAR algorithms on the range-velocity and range-angle spectra. The range-angle spectrum is calculated via the 2D-MUSIC algorithm whereas the range-velocity spectrum is calculated via 2D-FFT. It is apparent from figure 4.11d that the exponentially distributed noise power assumption may not be good for the 2D-MUSIC spectrum. The effects of masking in the CA-CFAR algorithm can be seen by comparing figures 4.11a and 4.11c.



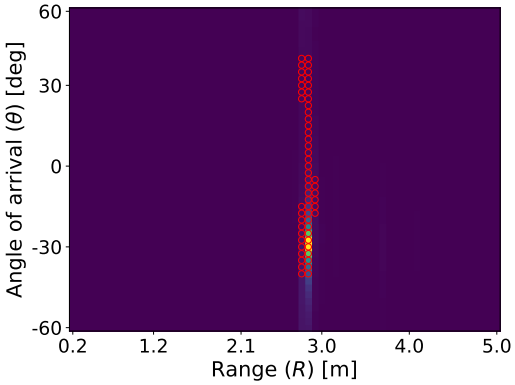
(a) *CA-CFAR applied on range-velocity spectrum calculated via FFT.*



(b) *CA-CFAR applied on range-angle spectrum calculated via 2D-MUSIC.*



(c) *OS-CFAR applied on range-velocity spectrum calculated via FFT.*



(d) *OS-CFAR applied on range-angle spectrum calculated via 2D-MUSIC.*

Figure 4.11. *The Cell Averaging and Ordered Statistic Constant False Alarm Rate algorithms applied on the range-velocity and range-angle spectra. The figures are scaled show the amplitude of the spectrum rather than power.*

Table 4.4. *Metadata fields for the depth camera.*

Key	Meaning
framerate	Number of recorded frames per second, decimal number
resolution	Resolution of the stored frames, string representation of a list: '[W,H]'

In addition to the basic CA- and OS-CFAR algorithms, multiple variations and combinations of the two have been developed over the years. The CA- and OS-CFAR still serve as the basis for most of the variations. The variations typically consider different computational complexity, noise distribution, target amplitude statistics, and multiple target situations. [56]

After target detection, target tracking algorithms may be used to track the same target between multiple frames. With target tracking, the track of a single target may be recorded and predicted and the appearance of new targets and the disappearance of old targets may be detected. Additionally, when the velocity spectrum of a single target can be recorded through multiple frames, this information can be used for HAR.

Target tracking can be split into two parts: track filtering and measurement-to-track association. The first part considers predicting the possible target tracks to determine the possible locations for a target in upcoming frames. The latter part considers assigning measurements (detected targets) to predicted paths. For track filtering, Kalman filtering is a popular choice. For measurement-to-track association, nearest-neighbour filtering or probabilistic models are commonly used. [58]

4.4 Depth camera record

The depth camera record is stored in the `depth.raw` file. The recording program produces two bytes of output per pixel. The output represents the measured distance at the pixel in millimeters as a 16-bit integer number (short int). The frame resolution and frame rate are stored in the `metadata.yaml` file. Table 4.4 documents the metadata fields recorded under the `camera.depth` section in `metadata.yaml` file.

Given the resolution is $H \times W$ (height $\times$ width) and the recorded data is represented by $\mathbf{d}$, where each element is two bytes long, the index of the i:th pixel of the z:th frame $\mathbf{f}_z(i)$ can be calculated using equation 4.57.

$$\forall i \in [0, WH-1] \wedge z \in \left[0, \frac{|\mathbf{d}|}{WH} - 1\right] : \mathbf{f}_z(i) = \mathbf{d}(WHz + i) \tag{4.57}$$

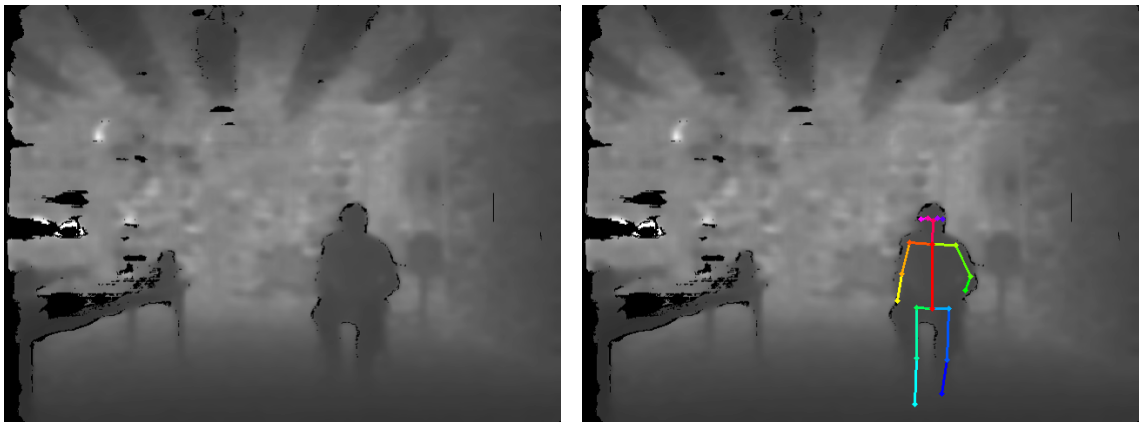
The frame $\mathbf{f}_z$ can then be rearranged as a matrix $\mathbf{F}_z$, which represents the pixels of the image in such a way that the origin is in the upper left corner. The pixel in coordinates

(h, w) in the frame $\mathbf{F}_z$ is given by the equation 4.58.

$$\forall h \in [0, H-1] \wedge w \in [0, W-1] : \mathbf{F}_z(h, w) = \mathbf{f}_z(hW + w) \tag{4.58}$$

Given the FoV of the sensor is $\Theta \times \Psi$ in the azimuth and altitude dimensions respectively, the FoV of each subpixel is $\frac{\Theta}{W} \times \frac{\Psi}{H}$. Given the angle is measured from the upper-left corner of a pixel and origin is in the center pixel, the range (in the z:th frame) R_z measured at angle (ψ, θ) is given by equation 4.59. The term $1 \div 1000$ converts the measured range from millimeters to meters.

$$\forall \psi \in \left[-\frac{\Psi}{2}, \frac{\Psi}{2}\right] \wedge \theta \in \left[-\frac{\Theta}{2}, \frac{\Theta}{2}\right] : R_z(\psi, \theta) = \frac{1}{1000}\mathbf{F}_z\left(\left\lfloor \frac{H}{2} + \frac{H\psi}{\Psi} \right\rfloor, \left\lfloor \frac{W}{2} + \frac{W\theta}{\Theta} \right\rfloor\right) \tag{4.59}$$



(a) *An example depth image.*

(b) *Joint position information extracted from the depth image (PNG) using OpenPose [59].*

Figure 4.12. *Depth image extracted from the* `depth.raw` *file.*

Appendix I shows an example of parsing the depth camera frames from the recorded data. An example of a resulting image is presented in Figure 4.12a. Machine learning models may be used to extract e.g. posture information from the depth images. Example models include TexMesh [60], A2J [61], and DoubleFusion [62]. Figure 4.12b shows an example of an image where the joint positions have been estimated using OpenPose [59] from the depth image that has been exported to PNG format and rendered to the image.

Table 4.5. *Metadata fields for the RBG camera.*

Key	Meaning
framerate	Number of recorded frames per second, decimal number
resolution	Resolution of the stored frames, string representation of a list: '$[W, H]$'

4.5 RGB camera record

The RGB camera record is stored in the `rgb.raw` file. The recording program produces three bytes of output per pixel. The bytes correspond to the red, green and blue values of the pixel, such that the first byte tells the red value, second byte tells the green value and the third byte tells the blue value. Each byte represents an unsigned 8-bit integer. The frame rate and resolution of the recording are stored in the `metadata.yaml` file under the `camera.rgb` section. Table 4.5 documents the metadata fields recorded for the RGB camera.

Denoting the vector of bytes stored in the file as $\mathbf{b}$, the data may be organized in pixels such that the i:th pixel $\mathbf{p}(i)$ is given by equation 4.60. When the pixels have been parsed, given the resolution of the recording is $H \times W$ (height $\times$ width), the z:th frame, denoted as $\mathbf{F}_z$ can be parsed based on equation 4.61.

$$\forall i \in \left[0, \frac{|\mathbf{b}|}{3}\right] : \mathbf{p}(i) = \begin{bmatrix} \mathbf{b}(3i) & \mathbf{b}(3i+1) & \mathbf{b}(3i+2) \end{bmatrix}^T \tag{4.60}$$

$$\forall z \in \left[0, \frac{|\mathbf{p}|}{HW}\right] : \mathbf{F}_z(h, w) = \mathbf{p}(zHW + hW + w) \tag{4.61}$$

Appendix I shows an example of parsing the RGB camera frames from the recorded data. An example of a resulting image is presented in Figure 4.13a. Machine learning models may be used to e.g. extract posture information or detect items from the RGB images. Some example models are OpenPose [59], VNect [63], and MixSTE [64]. Figure 4.13b shows an example of an image where the joint positions have been estimated using OpenPose [59].

(a) *An example RGB image.*

(b) *Joint position information extracted from the RGB using OpenPose[59].*

Figure 4.13. *RGB image extracted from the* `rgb.raw` *file.*

4.6 Microphone samples

The microphone record is stored in the `audio.wav` file. The file contains 16 audio channels and the sampling rate is stored in the `metadata.yaml` file. The sampling rate is also stored as metadata in the WAV file. Table 4.6 documents the metadata recorded for the audio file in the `metadata.yaml` file in the `audio` section.

Table 4.6. *Metadata recorded for the* `audio.wav` *file.*

Key	Meaning
samplerate	The sampling frequency for the microphone recording, integer number.

Figure 4.14 shows the positions of the microphones in the used MiniDSP UMA-16 microphone. The microphone numbers correspond to the audio channel numbers such that MIC1 is channel 0, MIC2 is channel 1, etc. Each sample in the data is represented by a 32-bit floating-point number. The samples are real-valued. Appendix J shows an example of reading the audio file with Python using the Soundfile [65] library.

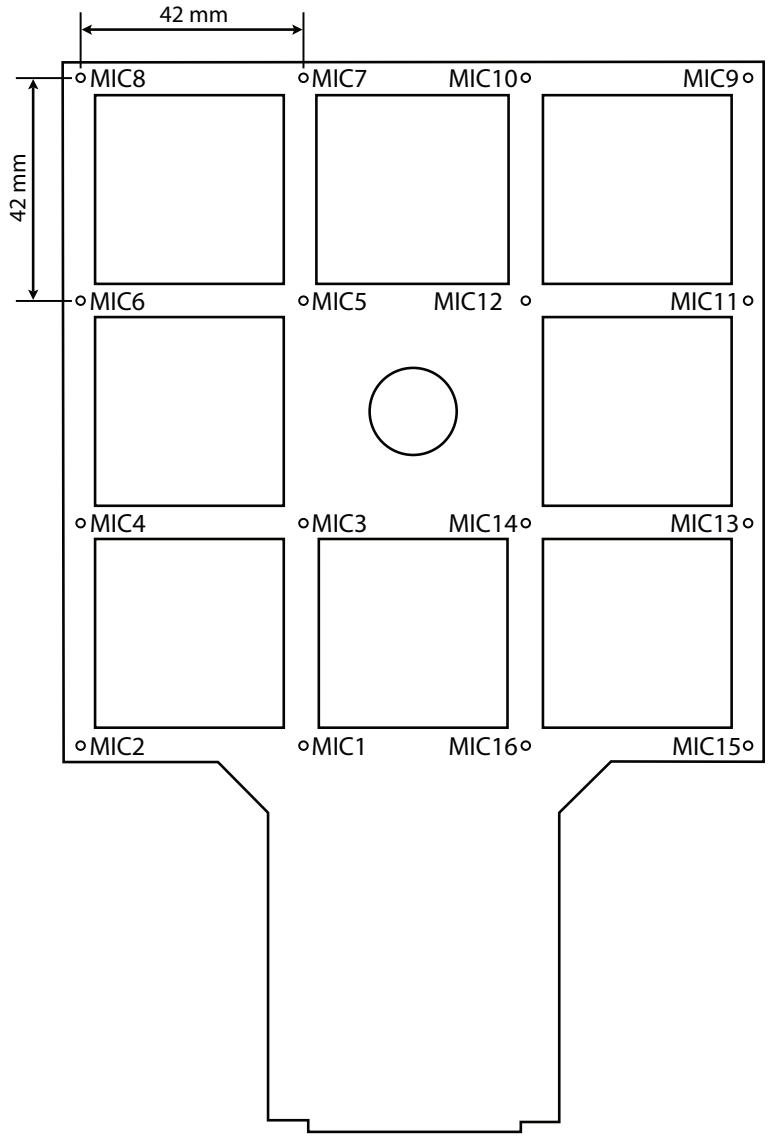


Figure 4.14. *Channels of the MiniDSP UMA-16 microphone [66].*

5. SYSTEM EVALUATION

In Section 2.4, requirements for the data recording system were defined. The quality of the system is evaluated based on the defined criteria. Additionally, the extensibility and stability of the software are considered.

5.1 Time synchronization of data

The criterion for data synchronization was defined in 2.4. The frame number z that corresponds to a moment in time t is given by the equation $z = \lfloor N_{\mathrm{FPS}} t \rfloor$, where N_{FPS} is the frame rate (equation 2.2). This holds true for any of the data sources. Thus, if a frame corresponding to the same point in time is extracted from each data source, high correlation should be seen between the frames.

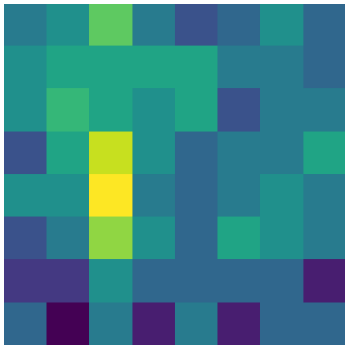
Judging the data synchronization is done, in this case, simply by calculating the range-angle spectrum from the radar frame and drawing it next to the RGB and IR frames. Because the RGB and Depth video are coming from the same sensor, they are guaranteed to be time-synchronized.

The audio sensor is omitted from this evaluation. For audio-to-radar synchronization the angle of arrival of a single tone could be estimated from the audio signal. Given the source for the tone is a radar target, the sound and radar angle-power spectrum could be observed. For audio-to-video synchronization, a video could be recorded of a person starting a tone generator. If the tone then appears in the audio signal at the same moment when the person starts the tone generator, it could be concluded that the sound and audio recordings are synchronized.

Figures 5.1–5.3 show the RGB, Radar (range-angle) and IR frames corresponding to different times. By looking at the frames, there seems to be high correlation between the angle of the target, thus it can be concluded that the data is well synchronized.

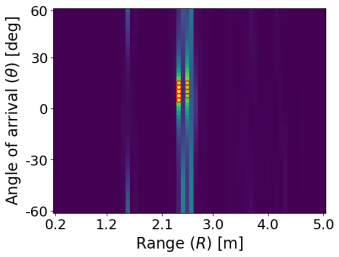
5.2 Data labelling

The data can be semi-automatically labelled during testing. Using this method, considerable effort must be made in planning the set of performed activities, which makes the performance less natural. Automatic activity recognition using i.e. RGB video based meth-



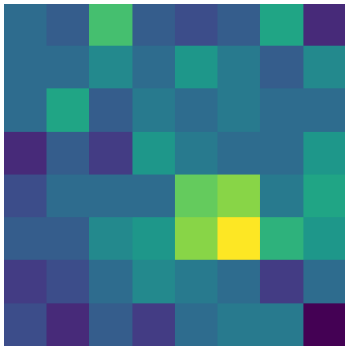
(a) *IR frame*

(b) *RGB frame*



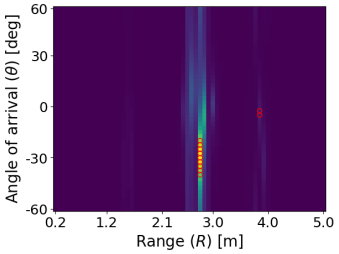
(c) *Radar angle-range spectrum*

Figure 5.1. *IR, RGB and radar frames,* $t = 7.0$.



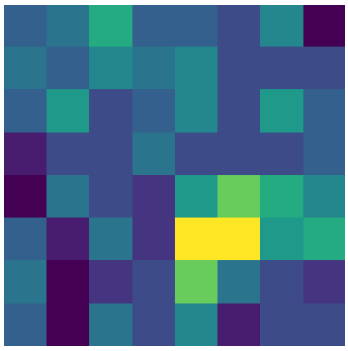
(a) *IR frame*

(b) *RGB frame*

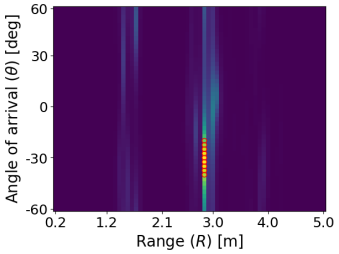


(c) *Radar angle-range spectrum*

Figure 5.2. *IR, RGB and radar frames,* $t = 24.0$.



(a) *RGB frame*



(b) *Radar angle-range spectrum*

Figure 5.3. *IR, RGB and radar frames,* $t = 53.0$.

ods could also be used to produce the activity labels in post-processing. The generated labels could then be used as ground-truth information for the other sensors in machine learning algorithms.

The format of the activity labels is very simple and because the data sources are time synchronized and the frame/sampling rates are known, it is easy to map the activities and data frames based on the time stamps and activity labels. The data labelling can therefore be considered sufficient.

5.3 Data structuring

Each recording is stored in a single directory. The software CLI provides a way to determine the name of the directory. The directory names can thus be used as unique identifiers to the recordings, and additional data such as person or room information can be mapped to the recordings using the directory names.

Under each directory, the files have consistent naming and the output from each sensor is stored in a separate file. This makes it easy to pick and choose used sensors and minimizes the amount of data that has to be loaded into memory for processing the output of a single sensor.

The data in the output files is unprocessed raw data from the sensors. As a downside, lots of post-processing may be required but as an upside, no information is lost. The structuring of the data, in the autor's opinion, is good.

5.4 Metadata

The recorded metadata considers only the sensor characteristics. The metadata file is easy to parse programmatically as it follows the YAML syntax. Sufficient parameter information is provided for processing the data. Thus, the metadata can be considered sufficient.

5.5 Extensibility of the system

The system is easy to extend to include new sensors. A recorder submodule must be written for each added sensor, and the sensor must be added as a source to the recroded submodule.

The system was written using a very crude proceduric model. Any well-established programming patterns were not followed. This may have had negative effects on extensibility of the code. Considering the module interfaces for the subprograms may have been worth considering. Instead of passing constant strings as signals to control the program, the signaling could have been abstracted to functions.

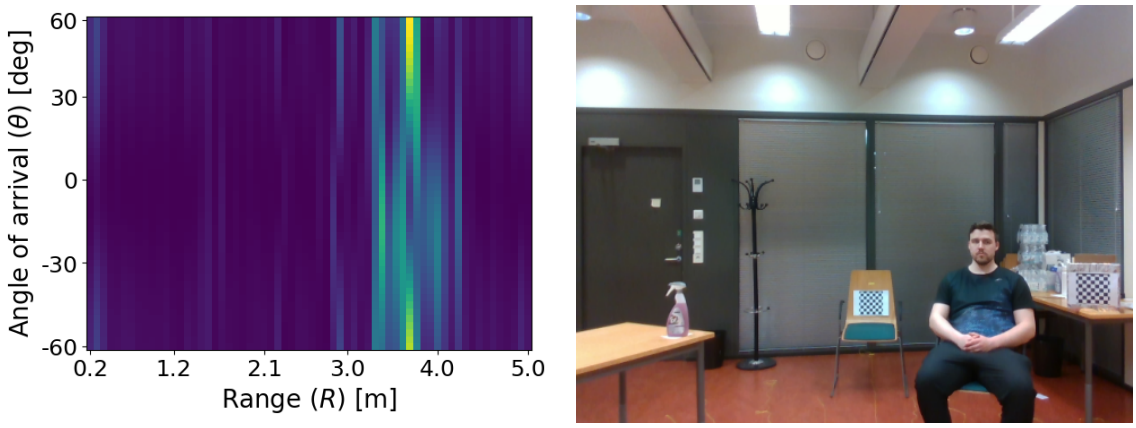
5.6 Stability of the system

In use, the system proved to be slightly instabile. Sometimes after starting the software, the recording never started as the system froze at some point of the initialization. Power-cycling the radar device fixed the issue, but this instability made the system much slower to use.

Effort should be made to fix the issue. Since the issue was fixed by power-cycling the radar device, the issue is likely related to not sending the correct commands to the radar device. Because the issue did not occur every time, it may also be related to parallelization.

5.7 Radar data quality

When processing the radar data, it was noticed that sometimes the range-angle spectrum becomes extremely noisy. Figure 5.4 shows an example. It may be worth investigating if using the two transmitter switching to increase the virtual number of receivers would alleviate this issue.



(a) *Radar angle-range spectrum.* ***(b)*** *Corresponding RGB image.*

Figure 5.4. *Example of a very noisy radar range-angle spectrum.*

6. CONCLUSION

As the product of this thesis, a multimodal sensor data recording system was produced. The implementation of the system was discussed, the produced data formats were documented and data processing examples were given to prove the documentation is truthful. Additionally, the quality of the system was assessed.

The system is capable of recording five different modalities of data: 4×4 acoustic, RGB video, depth video, 8×8 infrared video, and mmWave radar signal. The system is very portable as the sensors can be mounted on a single bracket that attaches to a camera tri-pod. Only a sufficient power source and a computer equipped with an SSD (Solid State Drive) drive are required. Wearable sensors were omitted from the system due to being cumbersome. Proximity sensors, such as magnetic switches, pressure plates, electrostatic sensors, etc. labour-intensive sensors were also omitted.

The system was designed to be parallel to maximize the data throughput. Each sensor has its own recorder subprocess that is controlled by a single main process. The subprograms are implemented as modules. The system can be extended to include additional sensors by writing a recording module for the sensor, and starting it from the main process.

The recorded data from the different sensors is synchronized in time. This means, that given a point in time, a corresponding frame can be easily extracted from any of the sensor outputs. In addition, the program provides a way to attach activity labels to the data during recording. The activity labels are stored in a separate file and the file format is very simple. It is also possible to apply the activity labels after recording. Manual labelling can be used for ultimate accuracy, or well-established activity recognition models can be used to detect actions form the RGB video. The labels can then be used as ground-truth information to train a machine learning model to recognize activities from the data produced by the other sensors.

The output from each sensor is stored in separate files and each recording is stored in a separate directory. Based on the output formats documented in Chapter 4, the data may be parsed into other formats, such as the popular `.mat` format used by MATLAB. Additionally, post-processing may be performed on the data to make it more suitable for machine learning. Most importantly, the range-angle and range-velocity spectra of the radar may be calculated using the algorithms presented in Sections 4.3.2 and 4.3.3.

While the system fulfilled the requirements set for in in Chapter 2, some instability was detected when using it, which makes the system unusable for carrying out a large-scale data recording campaign. For small-scale campaigns, the system is satisfactory.

Sometimes, after starting, the system freezes. It is unclear whether this is caused by problems in the parallel design, or whether it is caused by the radar sensor. Power-cycling the radar sensor and restarting the program fixed the issue with a high likelihood, though, which suggests that the problem may be caused by the radar. It should be looked into, if there is some further configuration that can be done to the radar device to increase its stability.

Additionally, the range-angle spectrum was extremely noisy in some frames. This issue was discussed in Section 5.7. The cause of the noisiness should be investigated. It should also be investigated if the issue can be alleviated by using the two-transmitter switching to increase the virtual number of receivers from 4 to 8. It is also possible that the used radar processing algorithms could be improved to produce a better range-angle estimate.

All in all, the implemented system serves a good basis. Some further development must still be done, but the system serves as a very good Proof of Concept. When the stability issues have been fixed, carrying out a larger-scale data collection campaign with the system should be feasible.

Even with small data sets, some preliminary research may be done with the system, especially considering transfer learning. Machine learning models trained with other data sets can be evaluated on data sets recorded on this system to see how the performance carries over. In addition, once the stability issues are fixed in the system, the data can even be used for training machine learning models. With the portable system, high quality data sets should be possible to collect, which will enable high-performance human activity recognition using remote sensing in various environments.

REFERENCES

[1] Lee, S.-M., Yoon, S. M. and Cho, H. Human activity recognition from accelerometer data using Convolutional Neural Network. *2017 IEEE International Conference on Big Data and Smart Computing (BigComp)*. 2017, pp. 131–134. DOI: 10.1109/BIGCOMP.2017.7881728.

[2] Rahman, A., Lubecke, V. M., Boric–Lubecke, O., Prins, J. H. and Sakamoto, T. Doppler Radar Techniques for Accurate Respiration Characterization and Subject Identification. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 8.2 (2018), pp. 350–359. DOI: 10.1109/JETCAS.2018.2818181.

[3] Li, Y., Ho, K. C. and Popescu, M. A Microphone Array System for Automatic Fall Detection. *IEEE Transactions on Biomedical Engineering* 59.5 (2012), pp. 1291–1301. DOI: 10.1109/TBME.2012.2186449.

[4] Ansari, M. A. and Singh, D. K. ESAR, An Expert Shoplifting Activity Recognition System. *Cybernetics and Information Technologies* 22.1 (2022), pp. 190–200. DOI: doi:10.2478/cait-2022-0012. URL: https://doi.org/10.2478/cait-2022-0012.

[5] Jacob, M. G. and Wachs, J. P. Context-based hand gesture recognition for the operating room. *Pattern Recognition Letters* 36 (2014), pp. 196–203.

[6] Lien, J., Gillian, N., Karagozler, M. E., Amihood, P., Schwesig, C., Olson, E., Raja, H. and Poupyrev, I. Soli: Ubiquitous Gesture Sensing with Millimeter Wave Radar. *ACM Trans. Graph.* 35.4 (July 2016). ISSN: 0730-0301. DOI: 10.1145/2897824.2925953. URL: https://doi.org/10.1145/2897824.2925953.

[7] Aversano, L., Bernardi, M. L., Cimitile, M. and Pecori, R. Early Detection of Parkinson Disease using Deep Neural Networks on Gait Dynamics. *2020 International Joint Conference on Neural Networks (IJCNN)*. 2020, pp. 1–8. DOI: 10.1109/IJCNN48605.2020.9207380.

[8] Fu, B., Damer, N., Kirchbuchner, F. and Kuijper, A. Sensing Technology for Human Activity Recognition: A Comprehensive Survey. *IEEE Access* 8 (2020), pp. 83791–83820. DOI: 10.1109/ACCESS.2020.2991891.

[9] Lara, O. D. and Labrador, M. A. A Survey on Human Activity Recognition using Wearable Sensors. *IEEE Communications Surveys & Tutorials* 15.3 (2013), pp. 1192–1209. DOI: 10.1109/SURV.2012.110112.00192.

[10] Mettel, M. R., Alekseew, M., Stocklöw, C. and Braun, A. Designing and evaluating safety services using depth cameras. *Journal of Ambient Intelligence and Humanized Computing* 10 (2019), pp. 747–759.

[11] Ni, B., Pei, Y., Moulin, P. and Yan, S. Multilevel Depth and Image Fusion for Human Activity Detection. *IEEE Transactions on Cybernetics* 43.5 (2013), pp. 1383–1394. DOI: 10.1109/TCYB.2013.2276433.

[12] Li, X., He, Y. and Jing, X. A survey of deep learning-based human activity recognition in radar. *Remote Sensing* 11.9 (2019), p. 1068.

[13] Shelke, S. and Aksanli, B. Static and dynamic activity detection with ambient sensors in smart spaces. *Sensors* 19.4 (2019), p. 804.

[14] Hevesi, P., Wille, S., Pirkl, G., Wehn, N. and Lukowicz, P. Monitoring household activities and user location with a cheap, unobtrusive thermal sensor array. *Proceedings of the 2014 ACM international joint conference on pervasive and ubiquitous computing*. 2014, pp. 141–145.

[15] Das, A., Sil, P., Singh, P. K., Bhateja, V. and Sarkar, R. MMHAR-EnsemNet: A Multi-Modal Human Activity Recognition Model. *IEEE Sensors Journal* 21.10 (2021), pp. 11569–11576. DOI: 10.1109/JSEN.2020.3034614.

[16] Bharti, P., De, D., Chellappan, S. and Das, S. K. HuMAn: Complex Activity Recognition with Multi-Modal Multi-Positional Body Sensing. *IEEE Transactions on Mobile Computing* 18.4 (2019), pp. 857–870. DOI: 10.1109/TMC.2018.2841905.

[17] Kupiainen, J. *MasterThesis*. 2023. URL: https://github.com/TeeKups/MasterThesis (visited on 05/23/2023).

[18] White, R. A Sensor Classification Scheme. *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control* 34.2 (1987), pp. 124–126. DOI: 10.1109/T-UFFC.1987.26922.

[19] *User's Guide: 60GHz mmWave Sensor EVMs*. SWRU546E. Texas Instruments. May 2022. URL: https://www.ti.com/lit/ug/swru546e/swru546e.pdf (visited on 02/05/2023).

[20] *MMWAVE SDK User Guide*. Texas Instruments. Nov. 2020. URL: https://dr-download.ti.com/software-development/software-development-kit-sdk/MD-PIrUeCYr3X/03.06.00.00-LTS/mmwave_sdk_user_guide.pdf (visited on 02/05/2023).

[21] *DCA1000EVM Data Capture Card User's Guide*. SPRUIJ4A. Texas Instruments. May 2019. URL: https://www.ti.com/lit/ug/spruij4a/spruij4a.pdf (visited on 02/05/2023).

[22] *Mmwave Radar Device ADC Raw Data Capture*. SWRA581B. Texas Instruments. Oct. 2018. URL: https://www.ti.com/lit/an/swra581b/swra581b.pdf (visited on 02/05/2023).

[23] Çağlıyan, B. and Gürbüz, S. Z. Micro-Doppler-Based Human Activity Classification Using the Mote-Scale BumbleBee Radar. *IEEE Geoscience and Remote Sensing Letters* 12.10 (2015), pp. 2135–2139. DOI: 10.1109/LGRS.2015.2452946.

[24] Seifert, A.-K., Amin, M. G. and Zoubir, A. M. Toward Unobtrusive In-Home Gait Analysis Based on Radar Micro-Doppler Signatures. *IEEE Transactions on Biomedical Engineering* 66.9 (2019), pp. 2629–2640. DOI: 10.1109/TBME.2019.2893528.

[25] Liu, L., Popescu, M., Rantz, M. and Skubic, M. Fall detection using doppler radar and classifier fusion. *Proceedings of 2012 IEEE-EMBS International Conference on Biomedical and Health Informatics*. 2012, pp. 180–183. DOI: 10.1109/BHI.2012.6211539.

[26] Kim, Y. and Moon, T. Human Detection and Activity Classification Based on Micro-Doppler Signatures Using Deep Convolutional Neural Networks. *IEEE Geoscience and Remote Sensing Letters* 13.1 (2016), pp. 8–12. DOI: 10.1109/LGRS.2015.2491329.

[27] Krizhevsky, A., Sutskever, I. and Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. *Commun. ACM* 60.6 (May 2017), pp. 84–90. ISSN: 0001-0782. DOI: 10.1145/3065386. URL: https://doi.org/10.1145/3065386.

[28] He, K., Gkioxari, G., Dollár, P. and Girshick, R. Mask R-CNN. *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 2980–2988. DOI: 10.1109/ICCV.2017.322.

[29] Cippitelli, E., Gasparrini, S., Gambi, E. and Spinsante, S. A human activity recognition system using skeleton data from RGBD sensors. *Computational intelligence and neuroscience* 2016 (2016).

[30] Han, J. and Bhanu, B. Human Activity Recognition in Thermal Infrared Imagery. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Workshops*. 2005, pp. 17–17. DOI: 10.1109/CVPR.2005.469.

[31] Hevesi, P., Wille, S., Pirkl, G., Wehn, N. and Lukowicz, P. Monitoring Household Activities and User Location with a Cheap, Unobtrusive Thermal Sensor Array. *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. UbiComp '14. Seattle, Washington: Association for Computing Machinery, 2014, pp. 141–145. ISBN: 9781450329682. DOI: 10.1145/2632048.2636084. URL: https://doi.org/10.1145/2632048.2636084.

[32] Tao, L., Volonakis, T., Tan, B., Jing, Y., Chetty, K. and Smith, M. L. Home Activity Monitoring using Low Resolution Infrared Sensor. *CoRR* abs/1811.05416 (2018). arXiv: 1811.05416. URL: http://arxiv.org/abs/1811.05416.

[33] *Intel® RealSense™ Product Family D400 Series Datasheet*. 337029-013. Intel. Nov. 2022. URL: https://www.intelrealsense.com/wp-content/uploads/2022/11/Intel-RealSense-D400-Series-Datasheet-November-2022.pdf (visited on 02/05/2023).

[34] Intel. *librealsense2*. GitHub repository. b874e42. Dec. 2022. URL: https://github.com/IntelRealSense/librealsense.

[35] *pyrealsense2: Librealsense™ Python Bindings*. Intel. 2017. URL: https://intelrealsense.github.io/librealsense/python_docs/_generated/pyrealsense2.html (visited on 02/05/2023).

[36] *User Manual: Grid-EYE Evaluation Kit*. Panasonic. URL: https://api.pim.na.industrial.panasonic.com/file_stream/main/fileversion/4712 (visited on 02/05/2023).

[37] *Communication Protocol*. Panasonic. URL: https://api.pim.na.industrial.panasonic.com/file_stream/main/fileversion/4710 (visited on 02/05/2023).

[38] *SoundDevice: Play and Record Sound with Python*. URL: https://python-sounddevice.readthedocs.io/en/0.4.5/ (visited on 02/07/2023).

[39] *Simple DirectMedia Layer*. URL: https://libsdl.org/ (visited on 02/13/2023).

[40] *PortAudio: Portable Cross-platform Audio I/O*. URL: http://www.portaudio.com/ (visited on 02/13/2023).

[41] *StackOverflow developer survey 2022*. URL: https://survey.stackoverflow.co/2022 (visited on 03/06/2023).

[42] *Asyncio: Asynchronous I/O*. URL: https://docs.python.org/3/library/asyncio.html (visited on 03/06/2023).

[43] *Threading: Thread-based parallelism*. URL: https://docs.python.org/3/library/threading.html (visited on 03/06/2023).

[44] *Multiprocessing: Process-based parallelism*. URL: https://docs.python.org/3/library/multiprocessing.html (visited on 03/06/2023).

[45] *PySerial*. URL: https://pythonhosted.org/pyserial/ (visited on 03/06/2023).

[46] Instruments, T. *MMWAVE-STUDIO*. URL: https://software-dl.ti.com/ra-processors/esd/MMWAVE-STUDIO/latest/index_FDS.html (visited on 03/07/2023).

[47] *SoundDevice: Streams using NumPy Arrays*. URL: https://python-sounddevice.readthedocs.io/en/0.4.6/api/streams.html#sounddevice.Stream (visited on 03/13/2023).

[48] Belfiori, F., Rossum, W. van and Hoogeboom, P. Application of 2D MUSIC algorithm to range-azimuth FMCW radar data. *2012 9th European Radar Conference*. 2012, pp. 242–245.

[49] Manokhin, G. O., Erdyneev, Z. T., Geltser, A. A. and Monastyrev, E. A. MUSIC-based algorithm for range-azimuth FMCW radar data processing without estimating number of targets. *2015 IEEE 15th Mediterranean Microwave Symposium (MMS)*. 2015, pp. 1–4. DOI: 10.1109/MMS.2015.7375471.

[50] Pillai, S. and Kwon, B. Forward/backward spatial smoothing techniques for coherent signal identification. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 37.1 (1989), pp. 8–15. DOI: 10.1109/29.17496.

[51] Wax, M. and Kailath, T. Detection of signals by information theoretic criteria. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 33.2 (1985), pp. 387–392. DOI: 10.1109/TASSP.1985.1164557.

[52] Dillard, G. M. and Antoniak, C. E. A Practical Distribution-Free Detection Procedure for Multiple-Range-Bin Radars. *IEEE Transactions on Aerospace and Electronic Systems* AES-6.5 (1970), pp. 629–635. DOI: 10.1109/TAES.1970.310063.

[53] Dillard, G. M. Mean-Level Detection of Nonfluctuating Signals. *IEEE Transactions on Aerospace and Electronic Systems* AES-10.6 (1974), pp. 795–799. DOI: 10.1109/TAES.1974.307886.

[54] Kronauge, M. and Rohling, H. Fast Two-Dimensional CFAR Procedure. *IEEE Transactions on Aerospace and Electronic Systems* 49.3 (2013), pp. 1817–1823. DOI: 10.1109/TAES.2013.6558022.

[55] Neyman, J., Pearson, E. S. and Pearson, K. IX. On the problem of the most efficient tests of statistical hypotheses. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character* 231.694-706 (1933), pp. 289–337. DOI: 10.1098/rsta.1933.0009. eprint: https://royalsocietypublishing.org/doi/pdf/10.1098/rsta.1933.0009. URL: https://royalsocietypublishing.org/doi/abs/10.1098/rsta.1933.0009.

[56] Rohling, H. SOME RADAR TOPICS: WAVEFORM DESIGN, RANGE CFAR AND TARGET RECOGNITION. *Advances in Sensing with Security Applications*. Ed. by J. Byrnes and G. Ostheimer. Dordrecht: Springer Netherlands, 2006, pp. 293–322. ISBN: 978-1-4020-4295-9.

[57] Rohling, H. Radar CFAR Thresholding in Clutter and Multiple Target Situations. *IEEE Transactions on Aerospace and Electronic Systems* AES-19.4 (1983), pp. 608–621. DOI: 10.1109/TAES.1983.309350.

[58] Mark A. Richards James A. Scheer, W. A. H. *Principles of Modern Radar: Basic Principles*. Vol. 1. SciTech Publishing, 2010. ISBN: 9781891121524.

[59] Cao, Z., Hidalgo Martinez, G., Simon, T., Wei, S. and Sheikh, Y. A. OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2019).

[60] Zhi, T., Lassner, C., Tung, T., Stoll, C., Narasimhan, S. G. and Vo, M. TexMesh: Reconstructing Detailed Human Texture and Geometry from RGB-D Video. *CoRR* abs/2008.00158 (2020). arXiv: 2008.00158. URL: https://arxiv.org/abs/2008.00158.

[61] Xiong, F., Zhang, B., Xiao, Y., Cao, Z., Yu, T., Zhou, J. T. and Yuan, J. A2J: Anchor-to-Joint Regression Network for 3D Articulated Pose Estimation from a Single Depth Image. *CoRR* abs/1908.09999 (2019). arXiv: 1908.09999. URL: http://arxiv.org/abs/1908.09999.

[62] Yu, T., Zheng, Z., Guo, K., Zhao, J., Dai, Q., Li, H., Pons-Moll, G. and Liu, Y. DoubleFusion: Real-time Capture of Human Performances with Inner Body Shapes from a Single Depth Sensor. *CoRR* abs/1804.06023 (2018). arXiv: 1804.06023. URL: http://arxiv.org/abs/1804.06023.

[63] Mehta, D., Sridhar, S., Sotnychenko, O., Rhodin, H., Shafiei, M., Seidel, H.-P., Xu, W., Casas, D. and Theobalt, C. VNect: Real-Time 3D Human Pose Estimation with a Single RGB Camera. *ACM Trans. Graph.* 36.4 (July 2017). ISSN: 0730-0301. DOI: 10.1145/3072959.3073596. URL: https://doi.org/10.1145/3072959.3073596.

[64] Zhang, J., Tu, Z., Yang, J., Chen, Y. and Yuan, J. MixSTE: Seq2seq Mixed Spatio-Temporal Encoder for 3D Human Pose Estimation in Video. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2022, pp. 13232–13242.

[65] *Soundfile*. URL: https://pypi.org/project/soundfile (visited on 05/21/2023).

[66] *UMA-16 v2 Microphone Array*. Version 1.1. MiniDSP. 2022. URL: https://www.minidsp.com/images/documents/Product%20Brief-UMA16%20v2.pdf (visited on 05/21/2023).

APPENDIX A: RADAR CONFIGURATION FILE USED IN DEVELOPMENT OF THE SYSTEM

```
sensorStop
flushCfg
dfeDataOutputMode 1
channelCfg 15 7 0
adcCfg 2 1
adcbufCfg -1 0 1 1 1
profileCfg 0 60 219 7 40 0 0 100 1 64 3500 0 0 30
chirpCfg 0 0 0 0 0 0 0 2
frameCfg 0 0 64 0 33.333 1 0
lowPower 0 0
guiMonitor -1 0 0 0 0 0 0
cfarCfg -1 0 2 8 4 3 0 15 1
cfarCfg -1 1 0 8 4 4 1 15 1
multiObjBeamForming -1 1 0.5
clutterRemoval -1 0
calibDcRangeSig -1 0 -5 8 256
extendedMaxVelocity -1 0
bpmCfg -1 0 0 1
lvdsStreamCfg -1 0 0 0
compRangeBiasAndRxChanPhase 0.0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1...
                                                    ... 0 1 0 1 0 1 0 1 0
measureRangeBiasAndRxChanPhase 0 1.5 0.2
CQRxSatMonitor 0 3 4 99 0
CQSigImgMonitor 0 63 4
analogMonitor 0 0
aoaFovCfg -1 -90 90 -90 90
cfarFovCfg -1 0 0 4.79
cfarFovCfg -1 1 -2.41 2.41
lvdsStreamCfg -1 0 1 0
```

APPENDIX B: DCA1000EVM CONFIGURATION COMMANDS

This page is empty on purpose

B.1 RESET_FPGA_CMD_CODE

Reset FPGA

B.1.1 Request

Name	Data Type	Number of bytes	Default Value	Min Value	Max Value	Description
Header	UINT16	2	0xA55A	-	-	0xA55A always. Start bits of packet.
Command code	UINT16	2	0x01	-	-	Command code
Size	UINT16	2	0	-	-	Data size
Footer	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet.

B.1.2 Response

Name	Data Type	Number of bytes	Default Value	Min Value	Max Value	Description
Header	UINT16	2	0xA55A	-	-	0xA55A always. Start bits of packet.
Command code	UINT16	2	0x01	-	-	Command code
Status	UINT16	2	0	0	1	0 – Success 1 – Failure
Footer	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet.

B.2 RESET_AR_DEV_CMD_CODE

Reset RADAR EVM

B.2.1 Request

Name	Data Type	Number of bytes	Default Value	Min Value	Max Value	Description
Header	UINT16	2	0xA55A	-	-	0xA55A always. Start bits of packet.
Command code	UINT16	2	0x02	-	-	Command code
Size	UINT16	2	0	-	-	Data size
Footer	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet.

B.2.2 Response

Name	Data Type	Number of bytes	Default Value	Min Value	Max Value	Description
Header	UINT16	2	0xA55A	-	-	0xA55A always. Start bits of packet.
Command code	UINT16	2	0x02	-	-	Command code
Status	UINT16	2	0	0	1	0 – Success 1 – Failure
Footer	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet.

B.3 CONFIG_FPGA_GEN_CMD_CODE

Configure FPGA

B.3.1 Request

Name	Data Type	Number of bytes	Default Value	Min Value	Max Value	Description
Header	UINT16	2	0xA55A	-	-	0xA55A always. Start bits of packet.
Command code	UINT16	2	0x03	-	-	Command code
Size	UINT16	2	6	-	-	Data size
Data Logging Mode	UINT8	1	1	1	2	1 – Raw mode 2 – Multi mode
LVDS mode	UINT8	1	1	1	2	1 – 4lane 2 – 2lane
Data transfer mode	UINT8	1	1	1	2	1 – LVDS capture 2 – DMM playback
Data capture mode	UINT8	1	2	1	2	1 – SD card storage 2 – Ethernet stream
Data format mode	UINT8	1	2	1	3	1 – 12-bit 2 – 14-bit 3 – 16-bit
Timer	UINT8	1	30 (0x1E)	0x0	0xFF	Timer info in seconds
Footer	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet.

B.3.2 Response

Name	Data Type	Number of bytes	Default Value	Min Value	Max Value	Description
Header	UINT16	2	0xA55A	-	-	0xA55A always. Start bits of packet.
Command code	UINT16	2	0x03	-	-	Command code
Status	UINT16	2	0	0	1	0 – Success 1 – Failure
Footer	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet.

B.4 CONFIG_EEPROM_CMD_CODE

Configure EEPROM

B.4.1 Request

Name	Data Type	Number of bytes	Default Value	Min Value	Max Value	Description
Header	UINT16	2	0xA55A	-	-	0xA55A always. Start bits of packet.
Command code	UINT16	2	0x04	-	-	Command code
Size	UINT16	2	6	-	-	Data size
System IP address	UINT8	4	192.168.33.30	0x0	0xFF	IP address
FPGA IP address	UINT8	4	192.168.33.180	0x0	0xFF	IP address
FPGA MAC address	UINT8	6	12-34-56-78-90-12	0x0	0xFF	MAC address
Configu-ration port	UINT16	2	4096	0x1	0xFFFF	Config Port number
Data port	UINT16	2	4098	0x1	0xFFFF	Data Port number
Footer	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet.

B.4.2 Response

Name	Data Type	Number of bytes	Default Value	Min Value	Max Value	Description
Header	UINT16	2	0xA55A	-	-	0xA55A always. Start bits of packet.
Command code	UINT16	2	0x04	-	-	Command code
Status	UINT16	2	0	0	1	0 – Success 1 – Failure
Footer	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet.

B.5 RECORD_START_CMD_CODE

Start record

B.5.1 Request

Name	Data Type	Number of bytes	Default Value	Min Value	Max Value	Description
Header	UINT16	2	0xA55A	-	-	0xA55A always. Start bits of packet.
Command code	UINT16	2	0x05	-	-	Command code
Size	UINT16	2	0	-	-	Data size
Footer	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet.

B.5.2 Response

Name	Data Type	Number of bytes	Default Value	Min Value	Max Value	Description
Header	UINT16	2	0xA55A	-	-	0xA55A always. Start bits of packet.
Command code	UINT16	2	0x05	-	-	Command code
Status	UINT16	2	0	0	1	0 – Success 1 – Failure
Footer	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet.

B.6 RECORD_STOP_CMD_CODE

Stop record

B.6.1 Request

Name	Data Type	Number of bytes	Default Value	Min Value	Max Value	Description
Header	UINT16	2	0xA55A	-	-	0xA55A always. Start bits of packet.
Command code	UINT16	2	0x06	-	-	Command code
Size	UINT16	2	0	-	-	Data size
Footer	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet.

B.6.2 Response

Name	Data Type	Number of bytes	Default Value	Min Value	Max Value	Description
Header	UINT16	2	0xA55A	-	-	0xA55A always. Start bits of packet.
Command code	UINT16	2	0x06	-	-	Command code
Status	UINT16	2	0	0	1	0 – Success 1 – Failure
Footer	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet.

B.7 PLAYBACK_START_CMD_CODE

Not documented.

B.8 PLAYBACK_STOP_CMD_CODE

Not documented.

B.9 SYSTEM_CONNECT_CMD_CODE

Query system aliveness status

B.9.1 Request

Name	Data Type	Number of bytes	Default Value	Min Value	Max Value	Description
Header	UINT16	2	0xA55A	-	-	0xA55A always. Start bits of packet.
Command code	UINT16	2	0x09	-	-	Command code
Size	UINT16	2	0	-	-	Data size
Footer	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet.

B.9.2 Response

Name	Data Type	Number of bytes	Default Value	Min Value	Max Value	Description
Header	UINT16	2	0xA55A	-	-	0xA55A always. Start bits of packet.
Command code	UINT16	2	0x09	-	-	Command code
Status	UINT16	2	0	0	1	0 – Success 1 – Failure
Footer	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet.

B.10 SYSTEM_ERROR_CMD_CODE

Query record process status

B.10.1 Request

Name	Data Type	Number of bytes	Default Value	Min Value	Max Value	Description
Header	UINT16	2	0xA55A	-	-	0xA55A always. Start bits of packet.
Command code	UINT16	2	0x0A	-	-	Command code
Size	UINT16	2	0	-	-	Data size
Footer	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet.

B.10.2 Response

Name	Data Type	Number of bytes	Default Value	Min Value	Max Value	Description
Header	UINT16	2	0xA55A	-	-	0xA55A always. Start bits of packet.
Command code	UINT16	2	0x0A	-	-	Command code
Status	UINT16	2	0	0	0xFF	System status code
Footer	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet.

B.11 CONFIG_PACKET_DATA_CMD_CODE

Configure record delay

B.11.1 Request

Name	Data Type	Number of bytes	Default Value	Min Value	Max Value	Description
Header	UINT16	2	0xA55A	-	-	0xA55A always. Start bits of packet.
Command code	UINT16	2	0x0B	-	-	Command code
Size	UINT16	2	6	-	-	Data size
Packet Size	UINT16	2	1472	48	1472	Packet size
Delay	UINT16	2	25	5	500	Inter-packet delay
Future Use	UINT16	2	0	-	-	Future Use
Footer	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet.

B.11.2 Response

Name	Data Type	Number of bytes	Default Value	Min Value	Max Value	Description
Header	UINT16	2	0xA55A	-	-	0xA55A always. Start bits of packet.
Command code	UINT16	2	0x0B	-	-	Command code
Status	UINT16	2	0	0	1	0 – Success 1 – Failure
Footer	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet.

B.12 CONFIG_DATA_MODE_AR_DEV_CMD_CODE

Not documented.

B.13 INT_FPGA_PLAYBACK_CMD_CODE

Not documented.

B.14 READ_FPGA_VERSION_CMD_CODE

Read FPGA version

B.14.1 Request

Name	Data Type	Number of bytes	Default Value	Min Value	Max Value	Description
Header	UINT16	2	0xA55A	-	-	0xA55A always. Start bits of packet.
Command code	UINT16	2	0x0E	-	-	Command code
Size	UINT16	2	9	-	-	Data size
Footer	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet.

B.14.2 Response

Name	Data Type	Number of bytes	Default Value	Min Value	Max Value	Description
Header	UINT16	2	0xA55A	-	-	0xA55A always. Start bits of packet.
Command code	UINT16	2	0x0E	-	-	Command code
Status	UINT16	2	-	0	0xFFFF	0–6th bits -> Major version 7th–13th bits -> Minor version 14th bit -> 0 – Record bit file 14th bit -> 1 – Playback bit file
Footer	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet.

APPENDIX C: PARSING THE IR RECORDER OUTPUT

Listing 8 shows a code example for parsing the IR frames from the `ir-raw` file.

```
import numpy

with open('ir.raw', 'rb') as _file:
    data = _file.read()

d = numpy.frombuffer(data, dtype='float16')
ir_frames = d.reshape((-1,8,8), order='C')
```

Listing C.1. *Code example for parsing infrared from file*

APPENDIX D: PARSING THE RADAR DATA CUBES FROM RAW RADAR DATA

The Listing 18 presents an example for parsing the radar data cubes from the raw radar data. The meanings of the variables used in the Listing, that correspond to the symbols in 4.3, are defined in D.1.

```
import numpy

def get_frames(N, M, K, path):
    with open(path, 'rb') as _file:
        data = _file.read()

    d = numpy.frombuffer(data, dtype='int16')

    n = numpy.arange(0, len(d), 4)
    d_r = numpy.array([d[n], d[n+1]]).flatten('F')
    d_i = numpy.array([d[n+2], d[n+3]]).flatten('F')
    s = d_r + 1j*d_i

    S = s.reshape((-1, M, K, N), order='C') \
            .transpose(0, 2, 1, 3)

    return S
```

Listing D.1. *Code example for extracting radar cubes from the raw radar data.*

Table D.1. *Variable meanings in Listing 18*

Variable	Definition
`N`	The number of samples per chirp
`M`	The number of chirps per sample
`K`	The number of active receivers
`d`	Raw 16-byte integer samples ($\mathbf{d}$)
`d_r`	Vector of real samples (in-phase component)
`d_i`	Vector of imaginary samples (quadrature component)
`s`	Vector of complex samples
`S`	The tensor $\mathbf{S}$, that contains the radar cubes (frames)

APPENDIX E: APPLYING FORWARD-BACKWARD SPATIAL SMOOTHING TO A DATA MATRIX

The listing 23 shows an example implementation of the FBSS algorithm written in python. The first argument for the function is a radar data cube, consisting of M chirps sampled N times on K receivers. The dimensions of the data cube are $K \times M \times N$. The second and third argument are the dimensions of the scanning window, and the last argument is the index of a chirp in the data cube; $m \in [0, M)$.

```
import numpy
def covariance_FBSS(data_cube, q1, q2, m):
    K = data_cube.shape[0]
    N = data_cube.shape[2]
    p1 = K–q1
    p2 = N–q2

    J = numpy.fliplr(numpy.identity(q1*q2))

    chirp = data_cube[:, m, :]
    d = lambda pp1, pp2 : \
        chirp[pp1:pp1+q1, pp2:pp2+q2].flatten('F')

    D = None
    for pp1 in range(p1):
        for pp2 in range(p2):
            D = numpy.column_stack( (D, d(pp1, pp2)) ) \
                if D is not None else d(pp1, pp2)

    DD = D @ D.conj().T
    D_cov = (1/(2*p1*p2)) * ( DD + J @ DD.T @ J)
    return D_cov
```

Listing E.1. *Code example for applying forward-backward spatial smoothing to a data matrix.*

APPENDIX F: 2D-MUSIC ALGORITHM FOR ESTIMATING THE RANGE-AZIMUTH POWER SPECTRUM

Listings 14–38 show an example of applying the 2D-MUSIC algorithm to a radar data cube with dimensions $K \times M \times N$, where K is the number of active receivers, M is the number of recorded dwells and N is the number of samples per dwell.

The first argument for the function defined in Listing 38 is the radar data cube. The second and third are respectively the range and AoA (Angle of Arrival) bins the spectrum shall be estimated for. The fourth argument is the slope of the chirp $s = B \div T_\mathrm{c}$, where s is the slope of the chirp, B is the bandwidth of the chirp during sampling and T_c is the duration of the chirp. The fifth argument is the sampling frequency and the sixth and final argument is the carrier frequency of the modulated chirp signal.

The number of targets L in the algorithm is estimated using AIC, as described by Wax and Kailath [51].

```
import numpy
from scipy import constants
from scipy import pi as PI
def music(frame, ranges, angles, slope, fs, fc):
    K, M, N = frame.shape
    wlen = constants.c / fc
    d = wlen / 2
    p1 = 2; p2 = 2
    q1 = K-p1; q2 = N-p2

    cov_mtx = numpy.mean( numpy.array(
        [ covariance_FBSS(frame, q1, q2, m) for m in numpy.arange(M) ]
    ), axis=0 )
```

Listing F.1. *Code example for estimating the range-azimuth spectrum for a radar data cube using the 2D-MUSIC algorithm. (Part 1/2)*

Listing 38 is continuation to listing 14.

```
eigvals, eigvecs = numpy.linalg.eigh(cov_mtx)
# sort largest first so noise subspace is in the end
sort_order = numpy.flip(numpy.argsort(eigvals))
eigvecs = eigvecs[:,sort_order]
W = eigvecs[:, L+1:]

p = len(eigvals)
def AIC(l):
    numerator = numpy.prod(eigvals[l:]) ** (1/(p-l))
    denominator = (1 / (p-l)) * numpy.mean(eigvals[l:])
    exponent = ((p-l)*(K))
    sum_factor = 2*l*(2*p-l)

    return -2*numpy.log10(
        (numerator/denominator)
    ) ** exponent + sum_factor

L = numpy.argmin(numpy.array([AIC(l) for l in range(p)]))
steering_vec = lambda theta: numpy.exp(
    1j*((2*PI)/wlen) * d*numpy.arange(q1) * numpy.sin(theta)
)

range_vec = lambda r: numpy.exp(
    1j*2*PI * ((2*r)/constants.c) * slope*numpy.arange(q2)*(1/fs)
)

alpha = lambda a, r: numpy.outer(
    steering_vec(a), range_vec(r)
).reshape((-1,1), order='F')

range_spectrum = lambda r, theta: 1 / (
    alpha(theta, r).conj().T @W@W.conj().T @ alpha(theta, r)
)

P = lambda theta : numpy.vectorize(range_spectrum)(ranges, theta)
spectrum = numpy.array([P(theta) for theta in angles])
return abs(spectrum)**2
```

Listing F.2. *Code example for estimating the range-azimuth spectrum for a radar data cube using the 2D-MUSIC algorithm. (Part 2/2)*

APPENDIX G: 2D-FFT ALGORITHM FOR ESTIMATING THE RANGE-VELOCITY SPECTRUM

Listings 24 shows an example of computing the range-velocity spectrum from a radar data cube using the two-dimensional Fast Fourier Transform method. The function takes a single argument, which is the radar data cube. The cube is three-dimensional with the first dimension corresponding to the receivers, second dimension to the chirps and third dimensions to the samples of each chirp.

```
import numpy
def range_velocity(frame):
    K, M, N = frame.shape
    C_1 = 1; C_2 = 2  ## Stetson-Harrison method
    lower = int((M/2-C_1/2)+0.5)
    upper = int((M/2+C_1/2)+0.5)

    range_filter_coefficients = numpy.diag(
        [ 0 if c in range(0, C_2) else 1 for c in range(N) ]
    )
    velocity_filter_coefficients = numpy.diag(
        [ 0 if c in range(lower, upper) else 1 for c in range(M) ]
    )

    bins = numpy.mean(frame, axis=0)
    fast_time_fft = numpy.fft.fft(bins, axis=1)
    slow_time_fft = numpy.fft.fft(fast_time_fft, axis=0)
    shifted = numpy.fft.fftshift(slow_time_fft, axes=(0))

    return abs(
        velocity_filter_coefficients
       @ (range_filter_coefficients @ shifted.T).T
    )**2
```

Listing G.1. *Code example for estimating the range-velocity spectrum for a radar data cube using the 2D-FFT algorithm. (Part 1/2)*

APPENDIX H: DERIVATION OF THE RANGE AND VELOCITY EQUATIONS FOR FMCW RADAR

Figure H.1 illustrates the transmitted chirp and and the reflected echo from one target. After converting the received echo to baseband signal, the frequency of the resulting signal will be the difference between the currently transmitted signal and received echo. This frequency is called the beat frequency $f_{\mathrm{b}}(t)$.

Given the signal is sampled N times during a single chirp with a sampling frequency of F_{s}, the discrete Fourier transform will result in N range bins, hence the resolution of the Fourier transform is as given by equation H.1.

$$\Delta f_{\mathrm{b}} = \frac{F_{\mathrm{s}}}{N} \tag{H.1}$$

H.1 Deriving the range equation

Assuming the radar is of monostatic kind and not moving, the beat frequency is produced solely by the delay caused by the round-trip-time from the radar to the target and back and the Doppler-shift caused by the target. Assuming the signal is propagating at the speed of light, the round-trip-time T_{RTT} is given by equation H.2, and the Doppler-shift f_{d} by

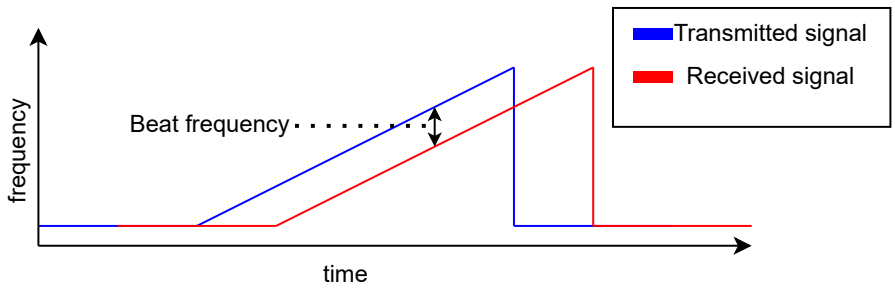


Figure H.1. *Transmitted signal, reflected echo, and beat frequency.*

equation H.3, where R_l is the range to the l:th target.

$$T_{\mathrm{RTT}} = \frac{2R_l}{c} \tag{H.2}$$

$$f_{\mathrm{d}} = \frac{v_l}{c} f \tag{H.3}$$

Because the slope s of the signal is constant, the beat frequency is linearly proportional to round-trip time. The beat frequency of the reflection from the l:th target is given by equation H.4.

$$f_{\mathrm{b},l} = s\frac{2R_l}{c} + f_{\mathrm{d},l} \tag{H.4}$$

When the doppler shift of the l:th target is much lower than the beat frequency resolution, i.e. $f_{\mathrm{d},l} \ll \Delta f_{\mathrm{b}}$, the equation H.4 can be approximated as given by equation H.5, thus the range of the target on range R_l is given by equation H.6.

$$f_{\mathrm{b},l} \approx \frac{2R_l s}{c} \tag{H.5}$$

$$R_l \approx \frac{f_{\mathrm{b},l} c}{2s} \tag{H.6}$$

By combining the equations H.1 and H.6, the range resolution of the fast-time Fourier transform is given by H.7.

$$\Delta R = \frac{\Delta f_{\mathrm{b}} c}{2s} \tag{H.7}$$

Due to complex sampling, the maximum frequency observable by the receiver is equal to the sampling frequency. Thus, the maximum beat frequency and thereby maximum range is dictated by the sampling frequency F_{s}. The maximum range is given by equation H.8.

$$R_{\max} = \frac{F_{\mathrm{s}} c}{2s} \tag{H.8}$$

H.2 Velocity equation

The change of the phase of a signal after it has been transmitted is given by the wavelength and distance travelled (equation H.9). Upon reflection, the signal experiences a phase

change of π radians and the frequency of the signal changes due to Doppler shift.

$$\Delta\varphi(R,f) = 2\pi\left(\frac{R}{\lambda} - \left\lfloor\frac{R}{\lambda}\right\rfloor\right) = 2\pi\left(\frac{fR}{c} - \left\lfloor\frac{fR}{c}\right\rfloor\right) \tag{H.9}$$

Equation H.9 is a surjection but not a bijection, thus the range of the target cannot be determined from the range unless it is less than the wavelength when the floor function term ($\lfloor\cdot\rfloor$) becomes zero and the function becomes a bijection. The special case is shown by equation H.10.

$$\forall R \in [0,\lambda] : \Delta\varphi(R,f) = 2\pi\frac{fR}{c} \tag{H.10}$$

When the signal reflects off a target, it experiences a phase change of π radians. The phase of the received reflection is thus given by equation H.11

$$\varphi_{\mathrm{RX}}(R,f) = \varphi_{\mathrm{TX}} + \Delta\varphi(R,f) + \pi + \Delta\varphi(R, f + \frac{v}{c}f) \tag{H.11}$$

Because phase is linearly proportional to range, as shown by equation H.10, the phase difference between the signals reflected from two targets moving at the same velocity can be calculated as shown by equation H.12.

$$\begin{aligned}\Delta\varphi(R_1,R_2,f) &= \Delta\varphi(R_1,f) + \pi + \Delta\varphi(R_1, f + \frac{v}{c}f)\\ &\quad - \left(\Delta\varphi(R_2,f) + \pi + \Delta\varphi(R_2, f + \frac{v}{c}f)\right)\\ &= \Delta\varphi(R_1,f) + \Delta\varphi(R_1, f + \frac{v}{c}f)\\ &\quad - \Delta\varphi(R_2,f) - \Delta\varphi(R_2, f + \frac{v}{c}f)\end{aligned} \tag{H.12}$$

Again assuming $R_1 - R_2 < \lambda$, the equation H.12 can be evaluated as given by equation H.13.

$$\begin{aligned}&\forall R_1 - R_2 \in [0,\lambda] : \Delta\varphi(R_1,R_2,f)\\ &\quad = 2\pi f R_1\left(\frac{1+\frac{v}{c}}{c}\right) - 2\pi f R_2\left(\frac{1+\frac{v}{c}}{c}\right)\\ &\quad = 2\pi f (R_1 - R_2)\left(\frac{1+\frac{v}{c}}{c}\right)\end{aligned} \tag{H.13}$$

The frequency of a sinusoidal wave can be expressed as $f = \omega \div (2\pi)$, where ω is the phase velocity of the wave. Given the phase difference of a single-tone signal sampled at a time interval of T_c is $\Delta\varphi_{T_\mathrm{c}}$, the corresponding frequency can be calculated using equation H.14.

$$\Delta\varphi_{T_\mathrm{c}} = 2\pi f T_\mathrm{c} \Leftrightarrow f = \frac{\Delta\varphi_{T_\mathrm{c}}}{2\pi T_\mathrm{c}} \tag{H.14}$$

From the properties of discrete Fourier transform, it is known that the frequency resolution for the transform is $\Delta f = 1 \div T_\mathrm{c}$. Because the target can have either a positive or negative velocity, both positive and negative frequencies may be induced by the change in range. Hence, the domain of interest for the slow-time Fourier transform is $f \in [-\frac{1}{2T_\mathrm{c}}, \frac{1}{2T_\mathrm{c}}]$. The corresponding set of phase shifts is $\varphi \in [-\pi, \pi]$ and the phase resolution of the transform is $\Delta\varphi = \frac{2\pi}{M}$.

Given the signal is sampled M times at the rate of T_c, the frequency resolution (Δf) for a discrete Fourier transform of the signal is $M \div T_\mathrm{c}$. Thus, the velocity required for a Doppler-shift to induce an error of one bin (v_err) is given by equation H.15. Given $M = 64$, $f = 60\ \mathrm{GHz}$ and $T_\mathrm{c} = 260\ \mu\mathrm{s}$ (realistic values for an FMCW (Frequency Modulated Constant Waveform) radar), $v_\mathrm{err} \approx 1230\ \frac{\mathrm{m}}{\mathrm{s}}$.

$$\frac{v_\mathrm{err}}{c} f = \frac{M}{T_\mathrm{c}} \Leftrightarrow v_\mathrm{err} = \frac{cM}{fT_\mathrm{c}} \tag{H.15}$$

Given $v \ll v_\mathrm{err}$, the equation H.13 can be approximated as given by equation H.16.

$$\begin{aligned} &\forall R_1 - R_2 \in [0, \lambda] : \Delta\varphi(R_1, R_2, f) \\ &\quad = \frac{2\pi f (R_1 - R_2)}{c} \end{aligned} \tag{H.16}$$

Substituting $\Delta\varphi$ in equation H.16 with the frequency resolution and denoting $R_1 - R_2 = 2\Delta R$ (change in target range causes twice the change in propagation distance), the equation can be solved for ΔR to acquire the range resolution for the slow-time Fourier transform. Dividing the value with the sampling interval T_c, the minimum and maximum velocity and the velocity resolution can be calculated as given by equations H.17 and H.18.

$$\Delta v = \frac{\Delta\varphi c}{4\pi f T_\mathrm{c}} = \frac{2\pi c}{4M\pi f T_\mathrm{c}} = \frac{c}{2Mf T_\mathrm{c}} \tag{H.17}$$

$$\begin{cases} v_\mathrm{max} = \frac{\varphi c}{4\pi f T_\mathrm{c}} = \frac{c}{4f T_\mathrm{c}} \\ v_\mathrm{min} = \frac{-\varphi c}{4\pi f T_\mathrm{c}} = \frac{-c}{4f T_\mathrm{c}} \end{cases} \tag{H.18}$$

APPENDIX I: PARSING FRAMES FROM RECORDED DEPTH AND RGB VIDEO

Listing 11 shows an example of parsing the frames from the `depth.raw` file. Listing 11 shows a similar example, but for the `rgb.raw` file.

```
import numpy

with open('depth.raw', 'rb') as f:
    data = f.read()

resolution = (480, 640)
d = numpy.frombuffer(data, dtype='int16')
frames = d.reshape((-1, *resolution), order='C')

F = lambda j : frames[j]  # get j:th frame
```

Listing I.1. *Code example for parsing depth frames from file*

```
import numpy

with open('rgb.raw', 'rb') as f:
    data = f.read()

resolution = (480, 640)
d = numpy.frombuffer(data, dtype='uint8')
frames = d.reshape((-1, *resolution, 3), order='C')

F = lambda j : frames[j]  # get j:th frame
```

Listing I.2. *Code example for parsing rgb frames from file*

APPENDIX J: PARSING THE AUDIO FILE WITH PYTHON SOUNDFILE

Listing 7 shows an example of parsing WAV files, such as the `audo.wav` file produced by the data recording system. The Soundfile [65] library is used for reading the file.

```
import soundfile

data, samplerate = soundfile.read('audio.wav', dtype='float32')

# All samples of n:th channel
S = lambda n : data[:, n]
```

Listing J.1. *Code example for parsing WAV files*