Software Implementation of mmWave Demo Application

PROJECT - PART 2 (20AG3FP30) report submitted to

Indian Institute of Technology Kharagpur

in partial fulfilment for the award of the degree of

Bachelor of Technology

in

Agricultural and Food Engineering

by

Gaurav Patidar

(20AG3FP30)

Under the supervision of

Professor Sandip Chakraborty

Computer Science and Engineering

Indian Institute of Technology Kharagpur

Spring Semester, 2023-24

April, 2024

DECLARATION

I certify that

(a) The work contained in this report has been done by me under the guidance of my supervisor.

(b) The work has not been submitted to any other Institute for any degree or diploma.

(c) I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.

(d) Whenever I have used materials (data, theoretical analysis, figures, and text) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references. Further, I have taken permission from the copyright owners of the sources, whenever necessary.

Date: April, 2024	(Gaurav Patidar)
Place: Kharagpur	(20AG3FP30)

COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

KHARAGPUR - 721302, INDIA

CERTIFICATE

This is to certify that the project report entitled "**Software Implementation of mmWave Demo Application**" submitted by **Gaurav Patidar** (Roll No. 20AG3FP30) to Indian Institute of Technology Kharagpur towards partial fulfilment of requirements for the award of degree of Bachelor of Technology in Agricultural and Food Engineering is a record of bona fide work carried out by him under my supervision and guidance during Spring Semester, 2023-24.

Professor Sandip Chakraborty
Computer Science and Engineering
Indian Institute of Technology Kharagpur
Kharagpur - 721302, India

Date: April, 2024
Place: Kharagpur

Acknowledgements

I would like to thank my supervisor **Prof. Sandip Chakraborty** to give me an opportunity to work under his guidance, who gave me constructive advice through out the course of my B.Tech project.

I would like to thank **Mr. Argha Sen**, PhD Scholar who helped me in this project whenever I was stuck, it would not have been possible to do this project without him.

I would like to thank every faculty member of Department of Computer Science and Engineering, IIT Kharagpur for guiding us in every way possible during the last four years.

Finally, I would like to thank my family and friends for their constant love and support.

Contents

Chapter 1

Introduction

1.1 Introduction

Millimeter-wave (mmWave) technology has shown promise in sensing human activity due to its ability to detect and track movements with high accuracy and precision. MmWave sensors emit electromagnetic waves with a wavelength in the millimeter range, which allows them to penetrate through some materials and detect changes in the environment caused by the movement of objects, including humans.

MmWave technology can be used for a range of sensing applications related to human activity, including:

1. **Gesture recognition:** MmWave sensors can detect and track hand movements, allowing them to be used for gesture recognition applications. This could include controlling devices such as smart home appliances, virtual and augmented reality interfaces, and even vehicles [5, 10, 6, 16, 9, 8].

2. **Vital signs monitoring:** MmWave sensors can detect small movements caused by breathing and heartbeats, allowing for non-invasive monitoring of vital signs [12, 14, 3, 1, 13, 7]. This could be used in medical settings for remote patient monitoring, or for fitness tracking and health monitoring.

3. **Occupancy sensing:** MmWave sensors can detect the presence of humans in a room or space, allowing for automated lighting and HVAC systems, and security applications [15, 2].
4. **Fall detection:** MmWave sensors can detect changes in a person's posture and movements, allowing for the detection of falls and other accidents [4].

Overall, mmWave technology has the potential to revolutionize the way we interact with our environment, making it possible to develop more intuitive and responsive interfaces that can detect and respond to our movements and behaviors in real time.

mmWave-based human activity sensing has several advantages over other modalities for detecting and tracking human activity. These include:

1. **High accuracy and precision:** mmWave sensors can detect and track movements with high accuracy and precision, allowing for detailed analysis of human activity.
2. **Non-invasive:** mmWave sensors emit electromagnetic waves with a wavelength in the millimeter range, which allows them to penetrate through some materials and detect changes in the environment caused by the movement of objects, including humans. This means that mmWave sensing can be non-invasive and does not require physical contact with the object being sensed.
3. **Long-range:** mmWave sensors can detect objects at a distance, allowing for monitoring of human activity in large spaces.
4. **Robustness:** mmWave sensing is less affected by environmental factors such as lighting conditions and occlusions than camera-based sensing.
5. **Privacy:** mmWave sensing can be used to detect human activity through walls and other obstacles, without capturing visual information, thus preserving privacy.
6. **Low power consumption:** mmWave sensors can consume less power compared to other sensing modalities, making them suitable for applications where power consumption is a critical factor.

Overall, mmWave-based human activity sensing offers a powerful and versatile technique for detecting and analyzing human activity, with several advantages over other modalities. It has the potential to enable a wide range of applications in fields such as healthcare, security, and smart homes. While mmWave-based human activity sensing has several advantages over other modalities, it also has some limitations and disadvantages. These include:

1. **Cost:** mmWave sensors are generally more expensive than other sensing modalities, which may limit their adoption in some applications.
2. **Limited penetration:** While mmWave sensors can penetrate some materials, such as drywall and glass, they may not be able to penetrate dense or metallic materials, which can limit their effectiveness in some environments.
3. **Limited field of view:** mmWave sensors typically have a limited field of view, which can limit their ability to capture activity in large spaces.
4. **Susceptible to interference:** mmWave sensors can be susceptible to interference from other electromagnetic sources, which can affect their accuracy and reliability.
5. **Complexity:** The technology behind mmWave sensing is complex and requires specialized knowledge, which can make it challenging to develop and deploy mmWave sensing systems.
6. **Safety concerns:** There are potential safety concerns associated with the use of mmWave technology, particularly at high power levels. However, these concerns can be addressed through proper system design and deployment.

And the most important disadvantage of any RF-based sensing approach is the annotation of the data. In the visual modality, humans can identify the activity and, accordingly, can annotate the activity, but in the case of mmWave modality, a range-doppler spectrogram can not be easily classified by the human brain. To make a subject familiar with the mmWave data, we need a platform where we can show the data in real-time and can plot the variation in the data over time. Thus the final objective of this work is to design a visualizer software which can showcase the mmWave data and can show it's variation over time.

Chapter 2

Related Works

mmWave Demo Visualizer software is typically used in the context of millimeter-wave (mmWave) technology, which is a type of wireless communication technology that operates in the frequency range of 30-300 GHz [11]. This technology is used in a variety of applications, including 5G networks, radar systems, and autonomous vehicles.

mmWave Demo Visualizer software is designed to help engineers and researchers visualize and analyze mmWave data. This data can include things like radar signals, antenna patterns, and channel measurements. The software may provide tools for visualizing this data in different formats, such as 2D or 3D plots, heatmaps, and spectrograms. It may also provide tools for analyzing the data, such as signal processing algorithms and statistical analysis tools.

Some examples of mmWave Demo Visualizer software include the National Instruments mmWave Visualizer[1], Keysight's PathWave software suite[2], and MATLAB's mmWave Explorer[3]. These tools are typically used in research and development settings, where engineers and scientists are working to develop new mmWave technologies and applications.

[1]`https://www.ni.com/pdf/manuals/378173b.html` (Accessed: April 21, 2024)
[2]`https://www.keysight.com/in/en/products/software.html` (Accessed: April 21, 2024)
[3]`https://www.mathworks.com/matlabcentral/fileexchange/73665-avnet-rfsoc-explorer` (Accessed: April 21, 2024)

Another popular software for visualizing mmWave data is mmWave Demo Visualizer[4] from Texas Instruments [5]. It works in conjunction with mmWave Radars from Texas Instruments. One needs to connect the radar to a computer via USB, and then the captured data can be plotted using the selected configuration in the software. Although it's a javascript-based software that can only work with internet connectivity. In this work, we closely followed this architecture in designing our software and also made it offline. The designed software can be executed on any machine, even a low-processing edge device like Raspberry Pi. In the next section we discuss the inputs and outputs of our software.

[4]`https://dev.ti.com/gallery/view/mmwave/mmWave_Demo_Visualizer/ver/2.1.0/` (Accessed: April 21, 2024)

[5]`https://www.ti.com/` (Accessed: April 21, 2024)

Chapter 3

System Architecture

The proposed software primarily consists of two Tabs. One is the configuration Tab, where one needs to select the configurations and pass the same to the mmWave Radar to boot the radar in the selected configuration, while the other tab is the Plot Tab, where the data coming from the mmWave Radar via USB, is plotted. Below we thoroughly discuss the details present in these tabs.

3.1 Configuration Tab

This tab allows users to select the parameters for configuring the mmWave device. In this document, the subsection numbers dictate a certain sequence of programming so that precedent parameters are set first followed by dependent parameters. Users can follow any sequence in configuring the settings but note that certain knobs, sliders, and drop-downs influence the values of other dependent parameters, and you may have to readjust dependent parameters to obtain the desired selection.

3.1.1 Platform

This is a drop-down menu that prompts users to select the correct mmWave device to which the app is connected through the serial port. If you are trying just to save config to a PC, then select the mmWave device for which the desired configuration must generate.

3.1.2 SDK Version

This is a drop-down menu that prompts users to select the matching SDK version as the one running on the mmWave device. If the version does not match, then the GUI produces an error when using the Send Config to mmWave Device button or Load config from PC and send button. However, there is no such check done when using the Save config to PC button, and users can choose the SDK version for which the compatible configuration should be generated.

3.1.3 Antenna Config (Azimuth Resolution - Degrees)

This is a drop-down menu that prompts users to select the azimuth resolution and elevation configuration for TI mmWave EVMs. The options for setting the azimuth resolution depend on the antenna layout on the EVMs.
Effects on other user knobs:

- Affects the possible minimum and maximum values for all sliders in the scene selection.
- When elevation is selected, the Scatter Plot selection results in a 3D plot in the plots tab.

CLI command details: defines RX and TX antenna mask in the channelCfg command and creates the chirpCfgs.

Note: Users must reboot the mmWave device when switching between options in this menu.

3.1.4 Desirable Configuration

This is a drop-down menu that prompts users to select the parameter that they are most concerned about, to tune the system resources towards that configuration. When the user selects a value in this menu, all the sliders, text boxes, checkboxes, and drop-downs (except for Platform) get reset to their default values (equivalent to pressing a Reset Selection button).

3.1.5 Frequency Band (GHz)

This is a drop-down menu that prompts users to select the frequency band of operation. The mmWave sensors support two choices: wideband (4 GHz spanning 77 to 81 GHz/60 to 64 GHz) and narrow band (1 GHz spanning 76 to 77 GHz).
CLI command details: defines the start frequency in GHz in the profileCfg command.

3.1.6 Scene Selection – Best Range Resolution

This section guides users through the sequence of selections they should follow when Best Range Resolution is selected under the Desirable Configuration menu.

3.1.6.1 Frame Rate (fps)

This is a slider that prompts users to select the rate at which the measurement data must be shipped out of the mmWave device.
Effects on other user knobs:

- Controls the minimum value on the Maximum Radial Velocity slider.
- Faster rates limit the number of plots and measurement data that can be captured from the mmWave device.

CLI command details: defines frame duration (frame periodicity in ms) in the frameCfg command.

3.1.6.2 Range Resolution (m)

This slider prompts users to select the desired range resolution in meters. Select the value based on the minimum amount of separation you expect between the detected objects or points in the point cloud. Because this is the Best Range Resolution configuration, the slider presents options only for the best possible range resolution for the user-selected frequency band.
Effects on other user knobs:

- Relatively lower or finer Range Resolution selected by this slider provides options for longer Max Unambiguous Range but lower values for Maximum Radial Velocity.
- Relatively higher or coarser Range Resolution selected by this slider provides options for shorter Max Unambiguous Range but higher values for Maximum Radial Velocity.

This slider allows for minor tweaking in the centimeter units, to help users attain desired fine-tuning of configuration for the Max Unambiguous Range and Maximum Radial Velocity.
CLI command details: defines frequency slope constant and Ramp end time in μs in the profileCfg.

3.1.6.3 Maximum Unambiguous Range (m)

This slider prompts users to select the desired Maximum Unambiguous Range in meters. Select the value based on the farthest distance you expect to see objects detected. To understand the Radar cross-section and the actual range at which objects equal to the radar cross-section can be theoretically detected by mmWave technology.
Effects on other user knobs:

- Setting this slider to lower values provides more options for the Radial Velocity Resolution, versus setting this slider to higher values providing fewer options for the Radial Velocity Resolution.

CLI command details: defines the number of ADC samples and ADC sampling frequency in *ksps* in the profileCfg command.

3.1.6.4 Maximum Radial Velocity (m/s)

This slider prompts users to select the desired Maximum Radial Velocity in meters/second. Select the value based on the maximum radial velocity you expect targets to be moving in within the radar field of view.
Effects on other user knobs:

- Radial Velocity Resolution is directly proportional to the Maximum Radial Velocity setting. Setting this slider to lower values provides finer Radial Velocity Resolution versus setting this slider to higher end of values provides coarser Radial Velocity Resolution

CLI command details: defines the idle time in μs in the profileCfg command.

3.1.6.5 Radial Velocity Resolution (m/s)

This is a drop-down menu that prompts users to select the desired Radial Velocity Resolution in meters/second.
Effects on other user knobs:

- This knob is the lowest dependent parameter in this configuration and gets constrained by all the preceding parameters.

CLI command details: defines the number of loops in the frameCfg command

3.1.7 Scene Selection – Best Velocity Resolution

3.1.7.1 Frame Rate (fps)

This is a slider that prompts users to select the rate at which measurement data must be shipped out of the mmWave device.
Effects on other user knobs:

- As the Best Velocity Resolution configuration, it controls the Radial Velocity Resolution, and sets it to the best possible value for the chosen frame rate.
- Controls the minimum value on the Maximum Radial Velocity slider.
- Faster rates limit the number of plots and measurement data that can be captured from the mmWave device.

CLI command details: defines frame duration (for example, frame periodicity in ms) in the frameCfg command.

3.1.7.2 Radial Velocity Resolution (m/s)

As a Best Velocity Resolution configuration, the Radial Velocity Resolution is set to the best possible value for the chosen frame rate.

3.1.7.3 Maximum Radial Velocity (m/s)

This slider prompts users to select the desired Maximum Radial Velocity in meter-s/second. Select the value based on the maximum radial velocity you expect the targets to be moving in within the radar field of view.
Effects on other user knobs: Based on the system resources diverted to fulfill the user-selected value for Maximum Radial Velocity, options for Range Resolution are affected in this configuration.

- Higher values for Maximum Radial Velocity selected by this slider provide only coarser options for Range Resolution and short-range options for Max Unambiguous Range.
- Lower values for Maximum Radial Velocity selected by this slider provide finer options for Range Resolution and long-range options for Max Unambiguous Range.

CLI command details: defines the number of loops in the frameCfg command.

3.1.7.4 Range Resolution (m)

This slider prompts users to select the desired range resolution in meters. Select the value based on the minimum separation you expect between the detected objects or points in the point cloud.
Effects on other user knobs:

- Relatively lower or finer Range Resolution selected by this slider provides options for shorter Max Unambiguous Range.
- Relatively higher or coarser Range Resolution selected by this slider provides options for longer Max Unambiguous Range.

CLI command details: Range Resolution and Max Unambiguous Range define the number of ADC samples, frequency slope constant, Ramp end time in μs, idle time in μs, and ADC sampling frequency in $ksps$ in the profileCfg command.

3.1.7.5 Maximum Unambiguous Range (m)

This slider prompts users to select the desired Maximum Unambiguous Range in meters. Select the value based on the farthest distance you expect to see objects detected. To understand the radar cross-section and the actual range at which objects equal to the radar cross-section can be theoretically detected by mmWave technology.
Effects on other user knobs:

- Moving this slider fine-tunes the user-selected Range Resolution value.

CLI command details: Range Resolution and Max Unambiguous Range define the number of ADC samples, frequency slope constant, Ramp end time in μs, idle time in μs, and ADC sampling frequency in $ksps$ in the profileCfg command.

3.1.8 Scene Selection – Best Range

3.1.8.1 Frame Rate (fps)

This is a slider that prompts users to select the rate at which measurement data must be shipped out of the mmWave device.
Effects on other user knobs:

- Controls the minimum value on the Maximum Radial Velocity slider.
- Faster rates limit the number of plots and measurement data that can be captured from the mmWave device.

CLI command details: defines the frame duration (for example, frame periodicity in ms) in the frameCfg command.

3.1.8.2 Maximum Unambiguous Range (m)

This slider prompts users to select the desired Maximum Unambiguous Range in steps of 5 meters. Select the value based on the farthest distance you expect to see objects detected.
Effects on other user knobs:

- Setting this slider to lower values provides fewer but finer options for Range Resolution, versus setting this slider to higher values providing more but coarser options for the Range Resolution.

CLI command details: defines the frequency slope constant in the profileCfg command.

3.1.8.3 Range Resolution (m)

This slider prompts users to select the desired range resolution in meters. Select the value based on the minimum amount of separation you expect between the detected objects or points in the point cloud.
Effects on other user knobs:

- Coarser values for Range Resolution selected by this slider provide more options and higher values for Maximum Radial Velocity.
- Finer values for Range Resolution selected by this slider provides fewer options and lower values for Maximum Radial Velocity.

Based on the system resources diverted to fulfill the user-selected value for Range Resolution, options for Maximum Radial Velocity are affected in this configuration.
CLI command details: defines the number of ADC samples, Ramp end time in μs, and ADC sampling frequency in $ksps$ in the profileCfg command.

3.1.8.4 Maximum Radial Velocity (m/s)

This slider prompts users to select the desired Maximum Radial Velocity in meters/second. Select the value based on the maximum radial velocity you expect the targets to be moving in within the radar field of view.
Effects on other user knobs:
Radial Velocity Resolution is directly proportional to the Maximum Radial Velocity setting. Setting this slider to lower values provides finer Radial Velocity Resolution, versus setting this slider to higher values providing coarser Radial Velocity Resolution. **CLI command details:** defines the idle time in µs in the profileCfg command.

3.1.9 Plot Selection

This section prompts users to select the plots they want to see on the Plots tab. For the best performance, depending on the scene parameters that are selected by the user, selecting more than two plots may require a frame rate of around 10 fps. Selecting heatmaps requires the frame rate to be 1-3 fps, otherwise, the target (mmWave sensor) will not have enough frame duration to ship out data every frame over the UART.

3.1.9.1 Scatter Plot

This lets users enable the detected objects list to be sent out by the target (mmWave sensor) device and display it on the Scatter plot and Doppler-Range plot on the plots tab. **CLI command details:** defines the detected-objects parameter in the guiMonitor command.

3.1.9.2 Range Profile

This lets users enable the log-magnitude range profile data at zero Doppler to be sent out by the target (mmWave sensor) device and display it on the Range Profile plot on the plots tab.
CLI command details: defines the log-magnitude range parameter in the guiMonitor command.

3.1.9.3 Noise Profile

This lets users enable the log magnitude noise profile data to be sent out by the target (mmWave sensor) device and display it on the Noise profile graph (green color) in the same Range Profile plot window on the plots tab.
CLI command details: defines the noise profile parameter in the guiMonitor command.

3.1.9.4 Range Doppler Heat Map

This lets users enable the entire detection matrix to be sent out by the target (mmWave sensor) device and display it on the range-doppler heatmap plot on the plots tab. This requires the frame rate to be 1 fps otherwise the target (mmWave

sensor) will not have enough frame duration to ship out data every frame over the UART.
CLI command details: defines the range-doppler heat map parameter in the guiMonitor command.

3.1.9.5 Statistics

This lets users enable the statistics information to be sent out by the target (mmWave sensor) device and to show the CPU load plot on the plots tab. CLI command details: defines the statistics parameter in the guiMonitor command.

3.1.10 User Selected Configuration

When the user has selected the parameters in the previous section, they can now take the following actions.

3.1.10.1 Send Configuration to mmWave Device

This lets users send the generated configuration (CLI) commands to be sent to the target (mmWave sensor) device. This command operates successfully only if the following are met:

- Valid serial ports have been configured as shown in the setup section.
- Target (mmWave sensor) device is booted up and running the mmWave demo from the same SDK version as selected in the SDK version drop-down menu.

Users can see the commands echoed back on the Console Messages display with the feedback that is received from the target for every line of commands sent. These console messages are a great help when debugging issues or getting familiar with the CLI commands. Successful execution of this button results in the last command sensorStart to be sent to the device and a response Done to be received from the device.

After successful execution, users can switch to the Plots tab to view the requested plots. If the last two lines in the console messages contain any other lines than these, it means either the device generated an error, the device stopped responding, or the commands did not go through completely and the user must try again.

3.1.10.2 Saving Configuration to PC

This lets users save the generated configuration (CLI) commands in a .cfg file on the user's PC. Users can then read and modify this file as per their needs. This file can also later be used to send the configuration to the device using the advanced option Load Config from PC and Send button on the plots tab.

3.1.10.3 Reset Selection

This lets users reset the sliders, checkboxes, textboxes, and drop-downs to their original reset values, except for the platform drop-down.

3.2 Plot Tab

When users have successfully configured the device using either the Send Config to mmWave Device or Load Config from PC and Send button, they can begin to see data being plotted on the requested plots.

3.2.1 X-Y Scatter Plot

This plot shows the detected objects in the X-Y domain (see Figure 7). For the advanced frame, this plot shows the union of points detected in all subframes of a given frame.

Note: Select Scatter Plot in the configure tab and configure the device accordingly to view this plot.

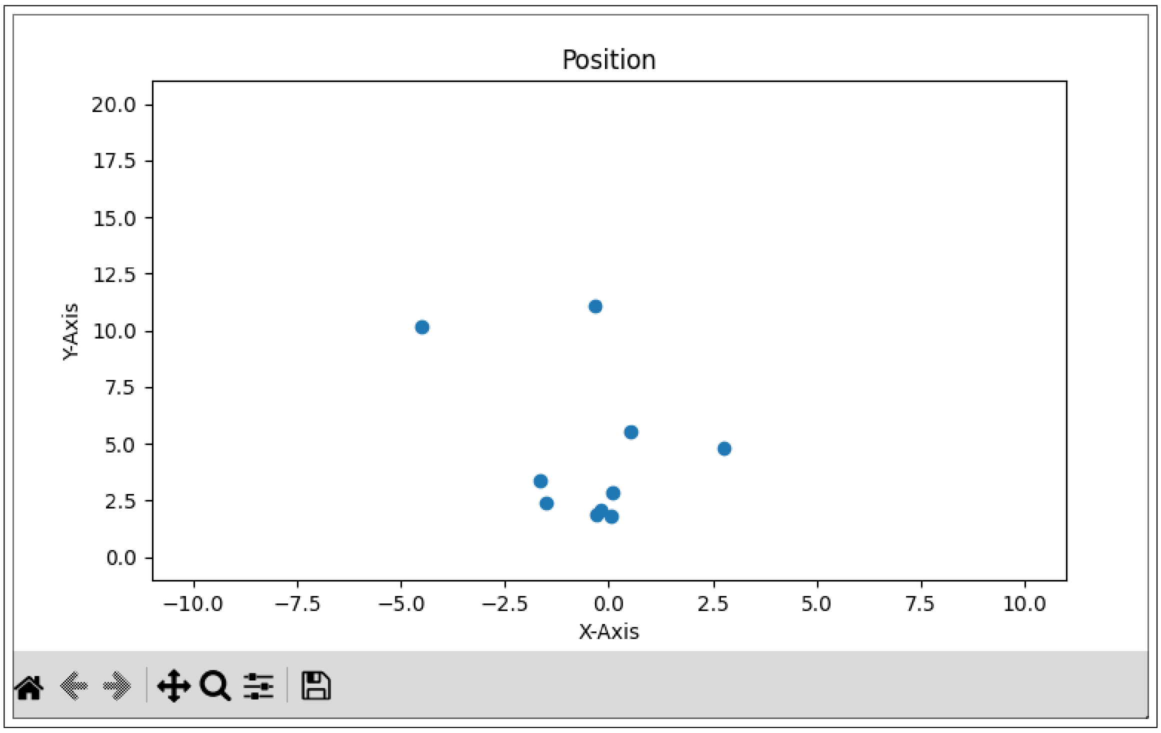

FIGURE 3.1: XY Scatter Plot

The X- and Y-axis can be controlled using the Range Width and Range Depth text boxes. The maximum value users can select is limited by the Maximum Unambiguous Range achievable by the current configuration.

3.2.2 Range Profile

This plot shows the range profile at the 0th Doppler (static objects) using the blue line and the noise profile (if enabled) using the green line (see Figure 11). By default, this graph shows the log values. The detected objects in the 0th Doppler range bin are shown as orange cross marks over the blue Range Profile plot line. For the advanced frame, this plot shows the range profile for the first subframe, which has this plot enabled in the guiMonitor command (the plot title reflects this subframe number). When the range bias is supplied using the compRangeBiasAndRxChanPhase command, the GUI internally uses the range bias to correct the range in meters, as calculated from rangeIdx, shipped by the mmWave device. The range and noise profile sent by the mmWave device are compensated for all the 1D and 2D FFT

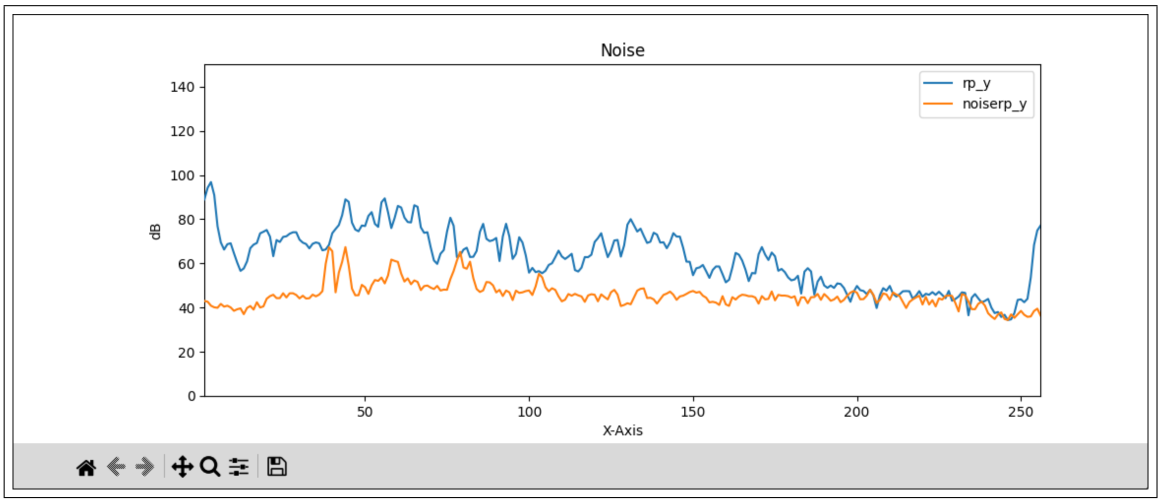

FIGURE 3.2: Range Profile

gains and the incoherent combining gain across the antennas, as per the mmWave demo processing chain, before plotting.

Note: Select Range Profile and Noise Profile in the Configure tab and configure the device accordingly to view this plot.

The linear scale for the Y-axis can be selected by unchecking the Range Profile Log Scale checkbox. The maximum limit for the Y-axis in the linear domain can be selected by the user with the Range Profile Ymax text box. Use the stop button to stop plotting and then you can change these settings. Once the settings are changed, resume plotting using the start button.

3.2.3 Range-Doppler Heatmap

This plot displays the entire radar cube matrix in Range and Doppler coordinates using the heatmap plot. For the advanced frame, this plot shows the heatmap for the first subframe which has this plot enabled in the guiMonitor command (the plot title reflects this subframe number). When the range Bias is supplied using the compRangeBiasAndRxChanPhase command, the GUI internally uses the range bias to correct the range in meters, as calculated from rangeIdx, shipped by the mmWave device.

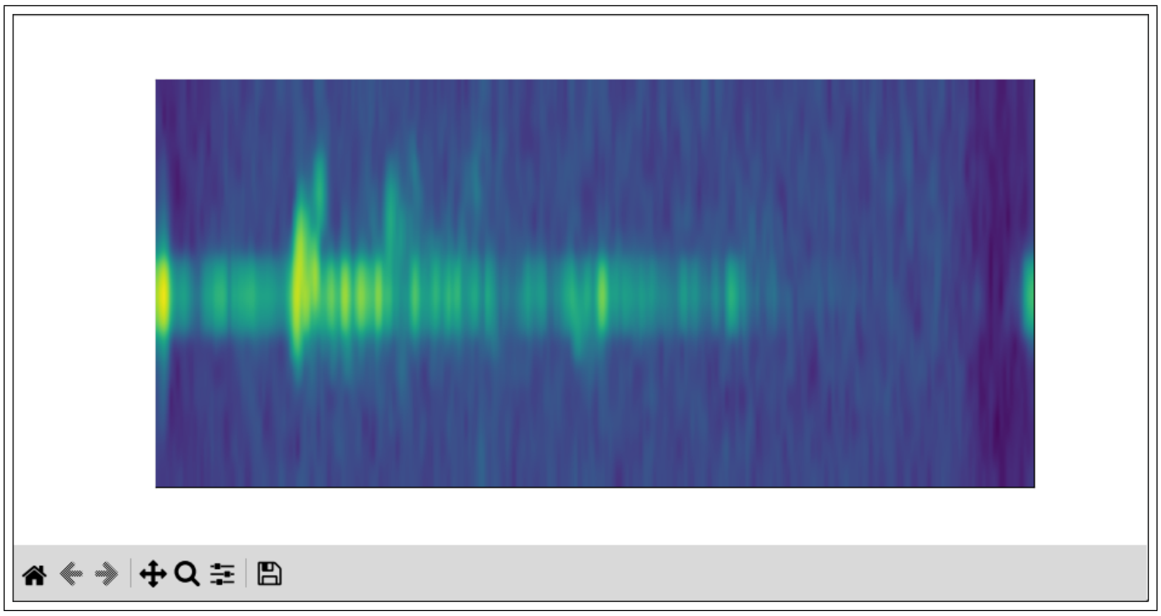

Figure 3.3: Range Doppler Heatmap

Note: Select Range Doppler Heat Map in the Configure tab and configure the device accordingly to view this plot.

Chapter 4

Implementation

Figure 4.1 shows the Implementation flow diagram of our demo visualizer software. Next we discuss the implementation in more detail.

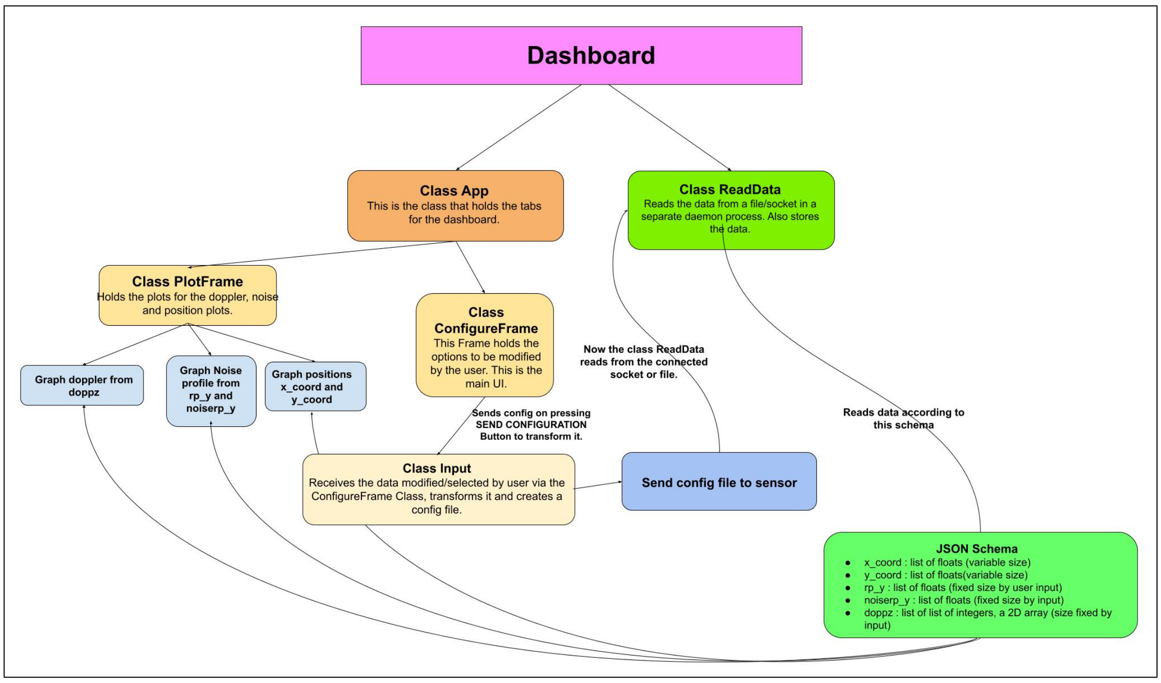
Dashboard
Class App
This is the class that holds the tabs for the dashboard.
Class ReadData
Reads the data from a file/socket in a separate daemon process. Also stores the data.
Class PlotFrame
Holds the plots for the doppler, noise and position plots.
Class ConfigureFrame
This Frame holds the options to be modified by the user. This is the main UI.
Graph doppler from doppz
Graph Noise profile from rp_y and noiserp_y
Graph positions x_coord and y_coord
Now the class ReadData reads from the connected socket or file.
Sends config on pressing SEND CONFIGURATION Button to transform it.
Reads data according to this schema
Class Input
Receives the data modified/selected by user via the ConfigureFrame Class, transforms it and creates a config file.
Send config file to sensor
JSON Schema
x_coord : list of floats (variable size)
y_coord : list of floats(variable size)
rp_y : list of floats (fixed size by user input)
noiserp_y : list of floats (fixed size by input)
doppz : list of list of integers, a 2D array (size fixed by input)

FIGURE 4.1: Class function and call flowchart

4.1 Package Dependencies and Usage

Not too many packages were required to build this app, but the interoperability of packages demanded a standard way to install packages so that the same can be reproduced in all machines. Now, there were various ways to make a graph or plot something in Python, thanks to its extensive number of packages, but the problem is with embedding animation. Not only should the animation be in real-time, the application demanded that each frame should be plotted in about 200 ms, as that is how fast the data is polled. Moreover, the application should take as low resources as possible, as it is required to run it on a Raspberry Pi. While there are certainly prettier and more modern Python libraries than Tkinter to build a Graphical user interface or a GUI, all were extremely heavy in resources or required too many packages to be installed. However, Tkinter already comes with every Python, as it is in Python's standard library. All it was required was to install tk and tcl and build Python with them. As the features vary amongst different versions of Python, Python was built with pyenv, a Python version manager. Pyenv was extremely useful in this case, as in different host machines we built, we had different versions of Python installed, and a standardized Python version was required in all the machines, namely Python 3.10.11.
Next up, we had to install the dependencies required to build the app. We opted for the more modern dependency manager, PDM (Python Dependency Manager). PDM uses pep582 to speed up dependency resolution by only importing the attributes needed to resolve the dependencies of a project, rather than importing the entire module. Also, PDM uses a lockfile to ensure reproducible builds and faster installation. The lockfile guarantees that the dependencies of a project are always the same, even if the dependencies themselves change over time. This allows PDM to skip unnecessary dependency resolution and installation steps. We just had to enable pep582, and add the packages we needed, like matplotlib(to graph), numpy(for calculations), ttkthemes(the default theme offered by tkinter is extremely clunky and frankly, ugly), and msgspec(for JSON deserialization using a predefined schema).
The rationale to use Matplotlib, instead of the more modern Seaborn or Plotly is to reduce memory usage. While Plotly produces much more interactive and "modern" graphs in Python, to implement a live plot or live animation inside of a tkinter window required the usage of dash in another subprocess to reload the Plotly graph, so the simpler, but effective matplotlib graphing tool was used. TTkthemes was used

Table 4.1: Schema

Variable Name	**Data Type**
`x_coord`	`list[float]`
`y_coord`	`list[float]`
`rp_y`	`list[float]`
`noiserp_y`	`list[float]`
`doppz`	`list[list[int]]`

purely for aesthetic reasons, and so is the use of the Noto Mono fonts. Msgspec was used for speed. Since we know beforehand the Schema of our JSON data, we can define a Schema as given in Table 4.1. This makes data deserializing much faster than just JSON decoding using the Python json package.

4.2 Technical Implementation

There are three parts to the implementation, namely the main app, the daemon that runs in the background, polling and collecting data from the sensor, and the input transformation, which transforms the input from the app, generates a config file and sends it to the sensor. The main 'body' of the dashboard first calls the daemon to start, with the polling of data initially set to false. Then it calls for the main Tkinter dashboard to display, and simultaneously calls the animation functions that run in the background, but are paused for the time being.

4.2.1 Animation Functions

There are three animation functions, plotting separately and updating every 200 ms:

- Position Animation - This shows the position of each detected object. This is plotted from the obtained x coordinate and y coordinate from the data reading class, explained in subsection 4.2.3. These coordinate lists are variable in size, as the sensor can detect any number of objects/people in a given time.
- Doppler Animation - This shows the output of the Doppler vector as a heatmap. This has a fixed size once initialized, i.e., it depends on the constraints set by

the inputs from the Configuration Tab, explained in Section 4.2.2. This animation *needs* to be blitted, as upgrading the image every time will severely degrade the performance.

- Noise Animation - This plots the `noiserp_y` and the `rp_y` from the data obtained in a single graph. These two plots also have a set size once initialized.

4.2.2 Implementation of the Dashboard

The 'app' or the dashboard was created using Tkinter. We first set the title, the icon, the geometry, and the theme using Tkinter. This was created as a Tkinter Notebook, i.e., with two Tabs or Frames - The first frame being the 'Configuration' frame, and the other being the 'Plot' Frame. While the plot frame holds the three animation plots described in section 4.2.1, along with their toolbars for interaction with the plot, the 'Configuration' Frame has the Labels, Buttons, and Comboboxes in three separate labeled Frames, namely setup, scene, and plot. This also consists of a 'send' button, which actually sends the data to be transformed, covered in Section 4.2.4, and then the config file is sent to the device, which, in turn, sends back the data, and is read by the ReadData Class, as covered in section 4.2.3. The setup LabelFrame contains the combo box to set the platform, the SDK version, the Antenna Configuration, the Subprofile type, and the Frequency band. The scene Labelframe contains Sliders to set the FPS Rate, the Range Resolution, the Maximum Unambiguous Range, the Maximum Radial velocity, and a Combo box to change the Radial Velocity Resolution. The plot contains options regarding the plot options, i.e., checkboxes for enabling/disabling showing the scatter plot, range profile, noise profile, range azimuth heat map, the Range Doppler heat map, and the overall statistics. There is also an option to import any file for replaying the data plotting. Any file, from which you can read json line-by-line, is supported.

4.2.3 Implementation of Animation Graphs

Since we can calculate the size of the receiving data, from the inputs in the dashboard, we can fix the graph size before the animation begins. However, we cannot say the same when we are reading from a file, as we are not storing the input size of each graph. We might have stored it and made a simple implementation, but the fact that the data is coming at a fixed speed, and we cannot plot the data as fast because of the Doppler Animation, we made a "follow-along" UI model. Here, once we receive data, we don't actually plot it. Rather, we store it in a queue. The advantage to this method is that:

- We can make a graph equal to the size of the data, once we receive the first frame.
- We can customize the update time for each graph. Since we know that some graphs are more difficult to render than others, and some data we receive have more fps than others, we can model the problem such that we store each data for a graph in a queue, then from the data from the first frame we set the size of the graph, and push each data in separate queues. Since we know the fps for each data received, we can render the data on the graph, and then release it from the queue once done. This way, we can avoid frame drops and buggy rendering at the same time. This is modeled to be similar to the signal architecture in Javascript frameworks like SolidJS, with caching.

4.2.4 Implementation for Reading Data from Sensor

Currently, the data is read from the sensor using another process, via USB, and read as a binary format file. The same is kept in a system file, or any of /dev devices, and the data is read and converted to json by a separate process. This is the data that is currently read by the polling daemon, as described in the section below. Currently, work is being done to integrate reading the binary data directly, without running a separate process.

4.2.5 Implementation of the Daemon for Reading Data

The ReadData class actually reads the data from either a socket or a file. In either case, it runs as a background daemon process. A threading daemon is made, as after extensive profiling, a threading process was faster than a multiprocessing one. Also, using a threading daemon, sharing data between processes becomes easier. We just read data when the thread is locked, and then the data is pushed to a queue, as described in Section 4.2.3. The ReadData class also has a play Event, which, when 'switched', enables data reading and the animations described in Section 4.2.1. Then it creates a daemon process, but doesn't start it. When started, it continually polss and updates the data. The read data can now be accessed from anywhere using an object of the ReadData class.

4.2.6 Implementation of Input Transformation

This is implemented as a single monolithic file. This reads the data as set by the user in the 'Configuration' File, and a various number of transformations are performed on the data, as per specification, to generate a config file. This file is what is sent to the device next, and the data then sent by the device is read by the ReadData class as explained in Section 4.2.3.

Chapter 5

Model Training and Predictions

5.1 Overview

This section delves into the intricate process of training a machine learning model tailored for activity classification, leveraging range Doppler data captured by mm-wave sensors. The ultimate goal is to deploy the trained model in real-time, enabling the prediction of various activities, especially focusing on identifying behaviors that could pose risks during driving scenarios.

5.1.1 Data Collection

The initial phase involves the meticulous collection of range Doppler data utilizing mm-wave sensors. These sensors are strategically positioned to capture signals reflecting activities such as clapping and jumping. The collected data forms the fundamental dataset crucial for subsequent model training.

5.1.2 Model Training

With the dataset in hand, the next crucial step is to train a machine learning model dedicated to activity classification. This intricate process entails feeding the meticulously curated range Doppler data into the model architecture. Through iterative optimization and parameter tuning, the model learns to discern and classify between different activities with a high degree of accuracy.

5.1.3 Prediction Process

Upon successful training, the model transitions into deployment for real-time predictions. Live data streams emanating from the mm-wave sensors are seamlessly integrated into the prediction pipeline. As new data flows in, the model's robust algorithms meticulously process each stream, swiftly identifying and categorizing observed activities. Whether it's detecting instances of clapping, jumping, or any other predefined activity, the model swiftly provides insights into the ongoing behavioral dynamics.

5.2 Implementation Details

This section provides an exhaustive breakdown of the intricate steps involved in implementing the machine learning model for activity classification and facilitating real-time predictions.

5.2.1 Model Training Procedure

The model training procedure is a meticulously orchestrated process designed to harness the potential of machine learning algorithms in accurately classifying activities based on range Doppler data. It commences with data preprocessing, where the raw data collected from mm-wave sensors undergoes a series of transformations to extract meaningful features and remove noise. This preprocessing step is paramount in ensuring the efficacy and reliability of the subsequent model training process.

Following data preprocessing, the next crucial phase entails model selection. Various machine learning algorithms such as Support Vector Machines (SVM), Random Forests, or Convolutional Neural Networks (CNNs) may be considered based on the nature of the data and the complexity of the classification task. The selection process involves evaluating the performance of different algorithms on a subset of the data through techniques like cross-validation.

Once an appropriate model architecture is chosen, hyperparameter tuning becomes the focal point. Hyperparameters are the configuration settings of the model that govern its learning process. Through systematic exploration of hyperparameter space using techniques like grid search or random search, optimal configurations are identified, maximizing the model's performance.

Lastly, comprehensive performance evaluation is conducted to assess the efficacy of the trained model. Metrics such as accuracy, precision, recall, and F1-score are computed to gauge the model's ability to correctly classify activities. Additionally, techniques like confusion matrices and ROC curves provide valuable insights into the model's strengths and weaknesses, aiding in further refinement if necessary.

5.2.2 Real-time Prediction Setup

The setup for enabling real-time predictions using the trained model involves a sophisticated integration of various components and technologies. At the forefront is the graphical user interface (GUI), which serves as the interface between the user and the underlying prediction system. The GUI provides intuitive controls for initiating and monitoring the prediction process, enhancing user experience and usability.

In parallel, the data input configuration plays a pivotal role in seamlessly streaming live data from the mm-wave sensors to the prediction system. This entails establishing robust data pipelines capable of handling high-volume data streams while ensuring minimal latency. Techniques such as buffering, parallel processing, and data compression may be employed to optimize data throughput and efficiency.

Central to the real-time prediction setup is the integration of the prediction logic within the system architecture. The trained machine learning model, along with its associated inference algorithms, is seamlessly embedded into the prediction pipeline. As new data streams in, the model swiftly processes each data point, generating real-time predictions with unparalleled speed and accuracy.

5.2.3 Alarm System Integration

To augment the predictive capabilities of the model and enhance safety measures, an alarm system is integrated into the prediction pipeline. This system is designed to detect instances of potentially hazardous activities, such as drowsiness or distraction, based on the model's predictions. Defined thresholds are established to trigger alarms when the model identifies activities that may pose risks to the driver or passengers.

The alarm system is equipped with sophisticated mechanisms to alert the driver promptly and effectively. Visual alerts, such as flashing lights or warning messages displayed on the dashboard, serve as immediate indicators of potential danger. Additionally, auditory alarms, such as beeps or voice alerts, provide audible cues to capture the driver's attention and prompt corrective action.

Furthermore, the alarm system may incorporate adaptive features to tailor alerts based on the severity of the detected activity. For instance, mild instances of distraction may trigger subtle alerts to remind the driver to refocus attention, while more critical situations, such as imminent drowsiness or inattention, may prompt urgent and assertive warnings to prevent accidents.

In summary, the integration of an alarm system adds an additional layer of safety and vigilance to the predictive capabilities of the model, safeguarding against potential risks and enhancing overall driving experience and safety.

Chapter 6

Code Flow

6.1 Overview

The code architecture comprises two pivotal components: `only_read.py` and `main.py`. These components collectively orchestrate the functionality of the application. `only_read.py` is primarily tasked with real-time data acquisition, while `main.py` governs the graphical user interface (GUI) and orchestrates the initialization of `only_read.py`. Additionally, the configuration files stored within the `Configurations` directory play a crucial role in setting parameters for point cloud processing, ensuring adaptability and versatility in the application's functionality.

6.1.1 Data Handling

Within the realm of data handling, `only_read.py` emerges as the cornerstone. This module is intricately designed to efficiently manage the influx of data, whether sourced from live feeds or retrieved from stored files within the `data` directory. The provision of a dedicated file for testing purposes not only streamlines development but also facilitates seamless toggling between live and static data sources, bolstering the application's versatility and adaptability to varied testing scenarios.

6.1.2 Graphical User Interface (GUI)

`main.py` stands as the epitome of user interaction, housing the graphical user interface (GUI) components. Through an intuitive and aesthetically pleasing interface, users can effortlessly engage with the application, leveraging an array of interactive elements such as buttons, sliders, and dropdown menus. These GUI components serve as conduits for visualization control, data interaction, and parameter adjustment, empowering users with a seamless and immersive experience.

6.1.3 Threading for Live Data

The seamless integration of live data streams into the application is facilitated by harnessing the power of the Python `threading` library. Leveraging threading capabilities allows the application to execute multiple tasks concurrently, thereby ensuring that the GUI remains responsive and fluid even as data is continuously read in the background. This asynchronous execution paradigm not only enhances the application's responsiveness but also optimizes resource utilization, laying the foundation for a robust and efficient data processing pipeline.

6.1.4 ML Models for Prediction

At the heart of the application lies the utilization of machine learning (ML) models for activity prediction. Python serves as the primary language for model creation and training, leveraging the rich ecosystem of ML libraries such as TensorFlow, PyTorch, or Scikit-learn. These ML models are meticulously trained on range Doppler data sourced from sensors, enabling them to discern and classify various activities with a high degree of accuracy. Upon deployment, the trained models seamlessly integrate into the application's prediction pipeline, where they dynamically analyze live data streams and provide real-time predictions regarding the observed activities. This fusion of advanced machine learning techniques with real-time data processing capabilities underscores the application's sophistication and efficacy in addressing complex activity classification tasks.

6.2 Configuration Management

Configuration files stored in the `Configurations` directory provide settings for processing point cloud data. These settings may include parameters for macro and micro configurations, such as frame rates, resolution, or other relevant options.

6.3 Folder Structure

The folder structure is organized as follows:

```
assets
   data
    data_file.json

 src
     _pycache__

     Configurations
        macro_5fps.cfg
        micro_2fps.cfg
        pointcloud_con.

     __init__.py
     main.py
     only_read.py
     .gitignore
     pdm.lock
     pyproject.toml
     README.md
```

This structure separates data, code, and configuration files, providing clarity and organization to the project.

For the GitHub code repository, you can include the link as follows:

6.3.1 GitHub Repository Link

The code for this project is available on GitHub at: `https://github.com/arghasen10/mmWave-Demo-App`

Chapter 7

Future Work and Conclusion

In this chapter, we discuss the future directions for our software and conclude with reflections on its current state and potential impact.

Our software represents a significant achievement in real-time data processing from sensors. It excels in collecting live data files and promptly plotting essential data visualizations such as range profiles, noise profile range-doppler, and point cloud representations. The software's ability to decode binary data directly from the sensor devices ensures efficient data retrieval, decoding, and plotting without unnecessary delays.

By open-sourcing the software on GitHub, we have extended its accessibility to a broader audience of researchers, developers, and enthusiasts. This decision not only fosters collaboration but also encourages contributions towards further improvements and enhancements.

Looking ahead, our roadmap includes the integration of machine learning models for live data analysis. This strategic move aims to unlock new opportunities for extracting valuable insights from sensor data in real-time. By leveraging machine learning algorithms, we anticipate enhancing the software's analytical capabilities, enabling it to perform tasks such as anomaly detection, object recognition, and predictive maintenance.

In summary, our software stands as a testament to our commitment to innovation and collaboration in the field of sensor data processing. Its current capabilities, coupled with the future integration of machine learning models, position it as a versatile tool with vast potential for applications in various domains, including autonomous systems, robotics, and environmental monitoring. [1].

[1]`https://github.com/arghasen10/mmWave-Demo-App`, (Accessed: April 21, 2024)

Bibliography

[1] Ahmad, A., Roh, J. C., Wang, D., and Dubey, A. (2018). Vital signs monitoring of multiple people using a fmcw millimeter-wave sensor. In *2018 IEEE Radar Conference (RadarConf18)*, pages 1450–1455. IEEE.

[2] Alizadeh, M., Abedi, H., and Shaker, G. (2019a). Low-cost low-power in-vehicle occupant detection with mm-wave fmcw radar. In *2019 IEEE SENSORS*, pages 1–4. IEEE.

[3] Alizadeh, M., Shaker, G., De Almeida, J. C. M., Morita, P. P., and Safavi-Naeini, S. (2019b). Remote monitoring of human vital signs using mm-wave fmcw radar. *IEEE Access*, 7:54958–54968.

[4] Li, W., Zhang, D., Li, Y., Wu, Z., Chen, J., Zhang, D., Hu, Y., Sun, Q., and Chen, Y. (2022a). Real-time fall detection using mmwave radar. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 16–20. IEEE.

[5] Li, Y., Zhang, D., Chen, J., Wan, J., Zhang, D., Hu, Y., Sun, Q., and Chen, Y. (2022b). Towards domain-independent and real-time gesture recognition using mmwave signal. *IEEE Transactions on Mobile Computing.*

[6] Liu, H., Wang, Y., Zhou, A., He, H., Wang, W., Wang, K., Pan, P., Lu, Y., Liu, L., and Ma, H. (2020). Real-time arm gesture recognition in smart home scenarios via millimeter wave sensing. *Proceedings of the ACM on interactive, mobile, wearable and ubiquitous technologies*, 4(4):1–28.

[7] Petkie, D. T., Benton, C., and Bryan, E. (2009). Millimeter wave radar for remote measurement of vital signs. In *2009 IEEE Radar Conference*, pages 1–3. IEEE.

[8] Sen, A., Das, A., Karmakar, P., and Chakraborty, S. (2023a). mmassist: Passive monitoring of driver's attentiveness using mmwave sensors. In *2023 15th International Conference on COMmunication Systems & NETworkS (COMSNETS)*, pages 545–553. IEEE.

[9] Sen, A., Mandal, A., Karmakar, P., Das, A., and Chakraborty, S. (2023b). mmdrive: mmwave sensing for live monitoring and on-device inference of dangerous driving. *arXiv preprint arXiv:2301.08188*.

[10] Smith, K. A., Csech, C., Murdoch, D., and Shaker, G. (2018). Gesture recognition using mm-wave sensor for human-car interface. *IEEE sensors letters*, 2(2):1–4.

[11] van Berlo, B., Elkelany, A., Ozcelebi, T., and Meratnia, N. (2021). Millimeter wave sensing: A review of application pipelines and building blocks. *IEEE Sensors Journal*, 21(9):10332–10368.

[12] Wang, F., Zeng, X., Wu, C., Wang, B., and Liu, K. R. (2021). Driver vital signs monitoring using millimeter wave radio. *IEEE Internet of Things Journal*, 9(13):11283–11298.

[13] Wang, Y., Wang, W., Zhou, M., Ren, A., and Tian, Z. (2020). Remote monitoring of human vital signs based on 77-ghz mm-wave fmcw radar. *Sensors*, 20(10):2999.

[14] Yang, Z., Pathak, P. H., Zeng, Y., Liran, X., and Mohapatra, P. (2016). Monitoring vital signs using millimeter wave. In *Proceedings of the 17th ACM international symposium on mobile ad hoc networking and computing*, pages 211–220.

[15] Yu, J., Wang, P., Koike-Akino, T., Wang, Y., Orlik, P. V., and Sun, H. (2020a). Human pose and seat occupancy classification with commercial mmwave wifi. In *2020 IEEE Globecom Workshops (GC Wkshps*, pages 1–6. IEEE.

[16] Yu, J.-T., Yen, L., and Tseng, P.-H. (2020b). mmwave radar-based hand gesture recognition using range-angle image. In *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, pages 1–5. IEEE.