Report Title:

Interfacing IWR6843AOPEVM Radar Module with Raspberry Pi: Setup, Configuration, and Results

Report Objectives:

- Interface the radar module with the Raspberry Pi and to document the steps and setup.
- Create the radar parameter's configuration file and to understand and study the implications of each parameter and the inter-relations.
- Modify the data parsing code to plot the velocity versus range plot.
- Test the data parsing code and to document the results obtained for a given scenario.

Executive Summary:

This report details the process of interfacing the radar module with the Raspberry Pi, including obtaining XY and velocity versus range plots for a given scenario. The interfacing involved setting up the radar module in the flashing mode and then on the functional mode, setting up the Raspberry Pi, verifying the recognition of the radar's port by the Raspberry Pi, downloading the required libraries and function script for the data parsing code, creating the radar configuration file and studying the parameters configured, testing the communication between the Raspberry Pi and radar module, and finally running the modified data parsing code.

Work Contribution:

This task was implemented and documented by Shayma Alteneiji

Table of Contents

I. Task Description

In this report, the steps and code used to successfully interface the radar module with the Raspberry Pi are detailed. Additionally, the obtained XY (or Range versus Cross-Range) and the Velocity versus Range plots are included for a given scenario. The following section lists the components and equipment used to implement the task.

II. Components and Equipment

1. IWR6843AOPEVM Radar Module
2. Radar Module Mounter
3. Micro USB-to-USB cable
4. Raspberry Pi 5
5. Micro SD card 64 GB
6. SD Card Reader
7. Power Supply for the Raspberry Pi
8. Micro HDMI-to-HDMI Cable
9. Monitor
10. Keyboard
11. Mouse
12. Raspberry Pi Case and Fan
13. USB

III. Methodology

The phases involved in achieving the specified task included: setting up the radar module on the flashing mode and then on the functional mode, setting up the Raspberry Pi, ports recognition verification, downloading the required libraries and function script, creating and using the radar configuration file, testing the communication between the Raspberry Pi and radar module, and finally running the IWR6843 data parsing code.

III.I Setting up the Radar Module in the Flashing and Functional Modes

In section, we detail the steps of setting up the EVM on the functional mode before it can be interfaced with the Raspberry Pi. However, this requires setting up the radar module in the flashing mode first. In the flashing mode, the EVM is flashed with a newer SDK version than the one it comes pre-installed with [1]. The loaded serial flash includes a valid MSS (Management Information System) application and DSS (Decision Support System) application that the EVM's bootloader uses in the functional mode.

To carry out this task, the instructions detailed in the "Hardware Setup for IWR6843AOP" by TI [2] were followed. The following additional steps were not thoroughly explained in the video and are detailed below.

- At minute 3:58, if the COM ports "Silicon Labs Dual CP210x USB to UART Bridge: Enhanced COM Port (COMx)" and "Silicon Labs Dual CP210x USB to UART Bridge: Standard COM Port (COMy)" do not appear in the Device Manger tab, instead, two devices appear under "Other Devices", as shown in Figure 1, it means the required drivers are not installed. To fix this problem, use the link provided in [3] to download the "CP210x VCP (Virtual COM Port) driver", then restart your PC, and the EVM's ports should appear under "Ports (COM & LPT)".

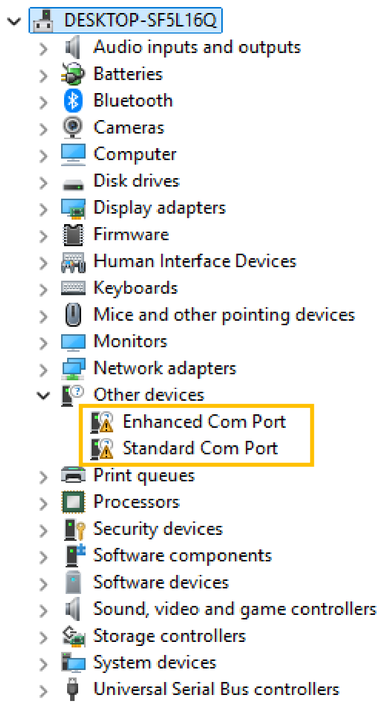


Figure 1. The device manager tab before installing the CP210x VCP (Virtual COM Port) driver.

- At minute 5:12, to install and locate the binary image file to be flashed into the EVM, download TI's Radar Toolbox folder found at this link [4]; the download button can be found at the top right side of the webpage. Use the following directory: <RADAR_TOOLBOX_INSTALL_DIR>\source\ti\examples\Out_Of_Box_Demo\prebuilt_binaries\ to locate the "out_of_box_6843_aop.bin" file in the downloaded Radar Toolbox folder [5]. The used .bin image file has been uploaded into the "Radar Module Related documents" folder in the MS teams group.

After completing the steps in the video, the radar module should be successfully set in the functional mode. The next section describes the setup and steps of setting up the Raspberry Pi.

III. II Setting Up the Raspberry Pi

This section details the steps and setup of setting up the Raspberry pi. Using the equipment and components mentioned in section II, follow the follow steps:

1) first use the SD card reader to attach the micro-SD card to the PC.
2) Search “Raspberry Pi imager” and download the package for Windows (or Mac or Linux, based on the device used).
3) When the installation is over, open the downloaded file using the “Run as administrator” option. The Raspberry Pi imager tab, shown in figure 2, should appear.
4) For the “CHOOSE DEVICE” option, select “Raspberry Pi 5” from the list of Raspberry Pi devices. For the “CHOOSE OS” option, select “Raspberry Pi OS (64-bit)” from the list of operating systems. For the “CHOOSE STORAGE” option, select the option that appears.
5) Press “Next”, and the tab shown in figure 3 should appear; Select “Edit Settings” and configure the Wireless LAN with the information of the server to connect to when using the Raspberry Pi, and then select “Save”.
6) Finally, select “yes” for the tab shown in figure 3.

After completing the previous steps, the operating system should be successfully installed into the micro-SD card, and it can be removed from the SD card reader and attached to the Raspberry Pi.

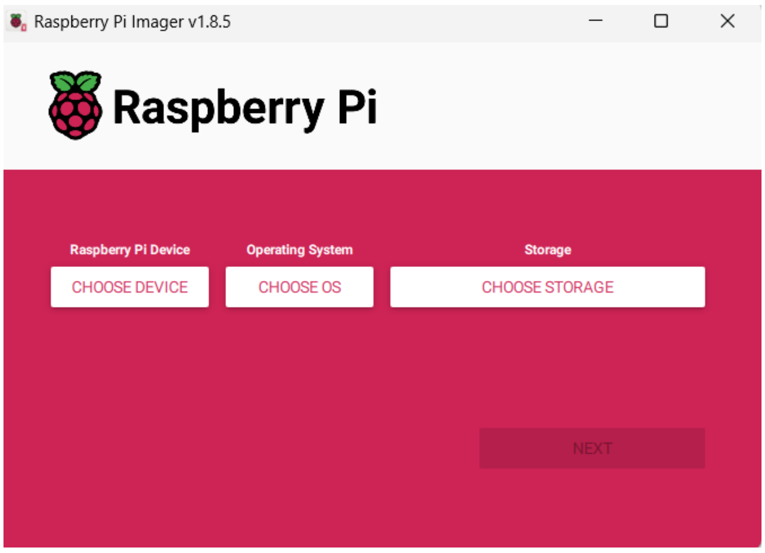
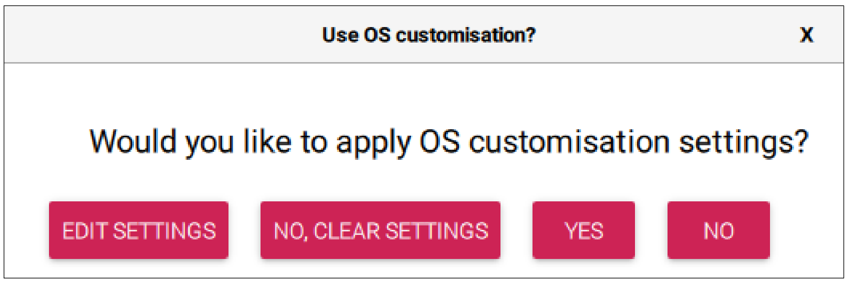


Figure 2. The Raspberry Pi imager tab.


Use OS customisation?

Would you like to apply OS customisation settings?

Figure 3. The “Use OS customization” tab.

After that, use the Hardware setup shown in figure 4 for the connections for the Raspberry Pi with the peripherals and the radar module. When turning on the Raspberry Pi's power supply, the Raspberry Pi's desktop must appear. Open the terminal window found on the top left side of the desktop screen. The next section discusses the procedures to verify that the Raspberry Pi is able to detect the connected radar module.

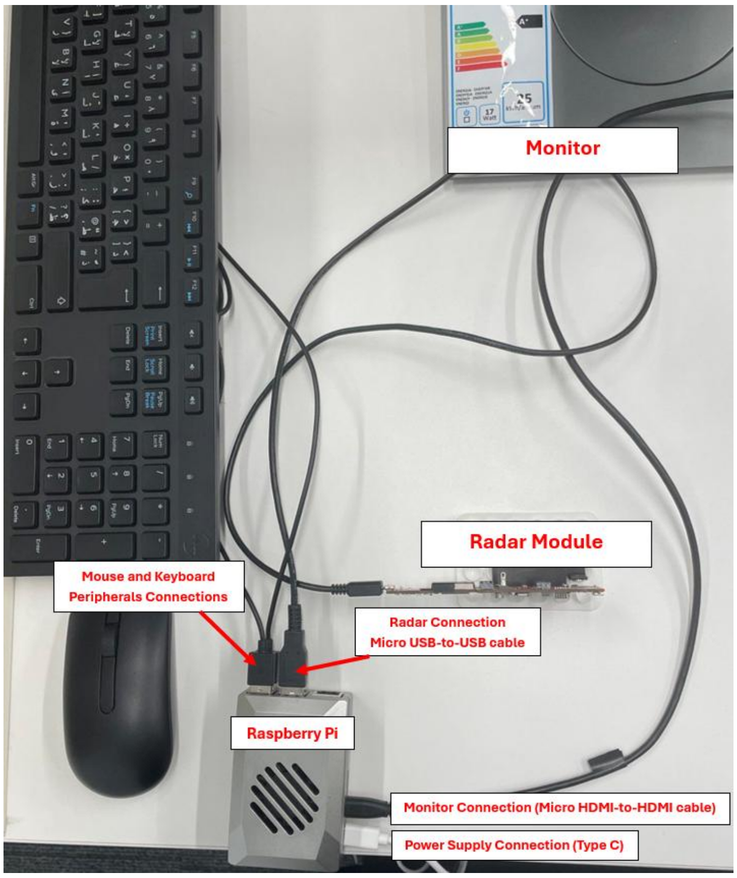

Monitor
Radar Module
Mouse and Keyboard Peripherals Connections
Radar Connection Micro USB-to-USB cable
Raspberry Pi
Monitor Connection (Micro HDMI-to-HDMI cable)
Power Supply Connection (Type C)

Figure 4. Hardware setup for the Raspberry Pi and Radar Module interfacing.

III. III Port Recognition Verification

To verify that the Raspberry Pi can recognize and detect the radar module, run the following code on terminal:

```
ls /dev/ttyUSB*
```

If the radar module is successfully detected, the terminal would print the following:

```
/dev/ttyUSB0    /dev/ttyUSB1
```

To ensure that the Raspberry Pi can access the ports, run the following code on Terminal:

```
sudo usermod -aG dialout $(whoami)
```

Then reboot the Rassberry Pi using the following code on Terminal:

```
sudo reboot
```

The next section discusses the terminal codes for downloading the libraries and the function script required for the data parsing code.

III. IV Downloading the Required Libraries and Function Script

Initially it was planned to create a virtual environment to run the data parsing code, however due to the difficulty of downloading the PyQt5 library on the virtual environment, instead all libraries were installed system wide. To install the numpy, run the following code on terminal:

```
sudo apt upgrade
sudo apt install numpy
```

To install the pyserial librabry:

```
sudo pip install --break-system-packages pyserial
```

To install PyQtGraph and PyQt5:

```
sudo apt install python3-pyqtgraph
sudo apt install python3-pyqt5
```

To install the parser_mmw_demo.py function script, we do the following:

1) To install the function script into the Raspberry Pi, use the link to the Github file and run the following code:

```
wget https://raw.githubusercontent.com/kirkster96/IWR6843-Read-Data-Python-MMWAVE-SDK/main/parser_mmw_demo.py
```

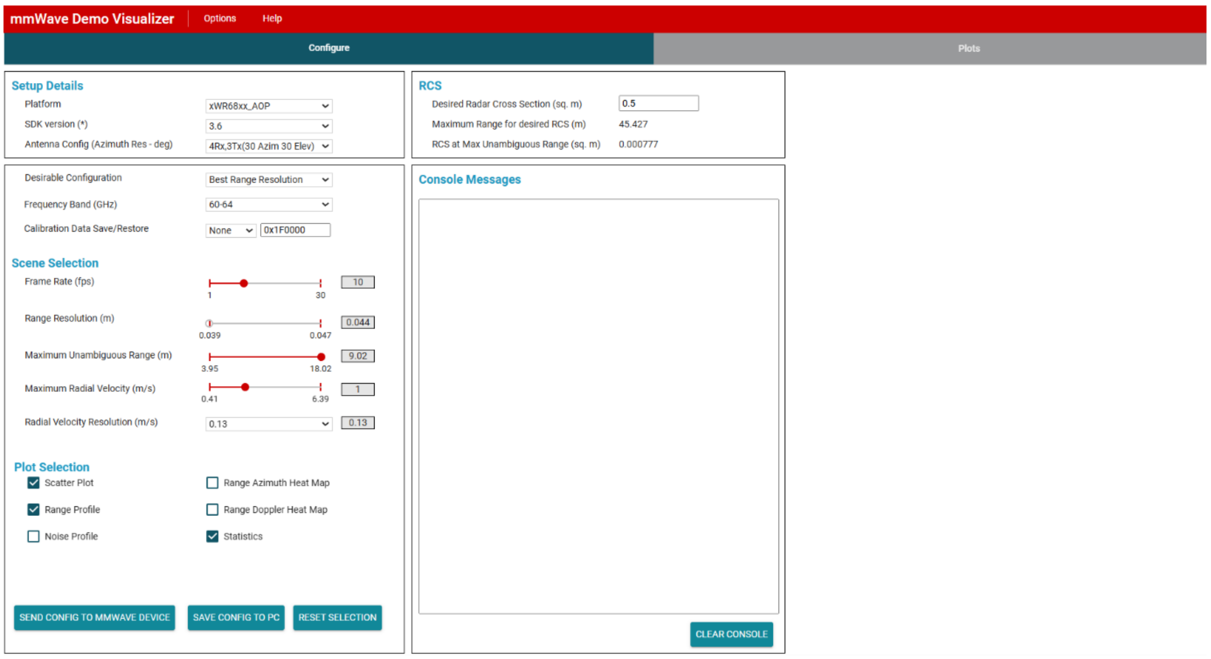
2) To move the parser function file to the same directory as that of the data parsing script (/home/Shayma/), run the following code:

```
mv parser_mmw_demo.py /home/Shayma/
```

In the following section we discuss creating the configuration file and studying the parameters configured.

III. V Creating the Radar Configuration File

The configuration file can be created and customized to the EVM's application using the Texas Instrument's mmWave_Demo_Visualizer browser-based application. Figure 5 shows the "Configure" tab of the mmWave_Demo_Visualizer webpage. Figure 6 shows an example of the content of a configuration file. The details of the parameters included in the configuration is discussed below.



Setup Details
Platform: xWR68xx_AOP
SDK version (*): 3.6
Antenna Config (Azimuth Res - deg): 4Rx,3Tx(30 Azim 30 Elev)

RCS
Desired Radar Cross Section (sq. m): 0.5
Maximum Range for desired RCS (m): 45.427
RCS at Max Unambiguous Range (sq. m): 0.000777

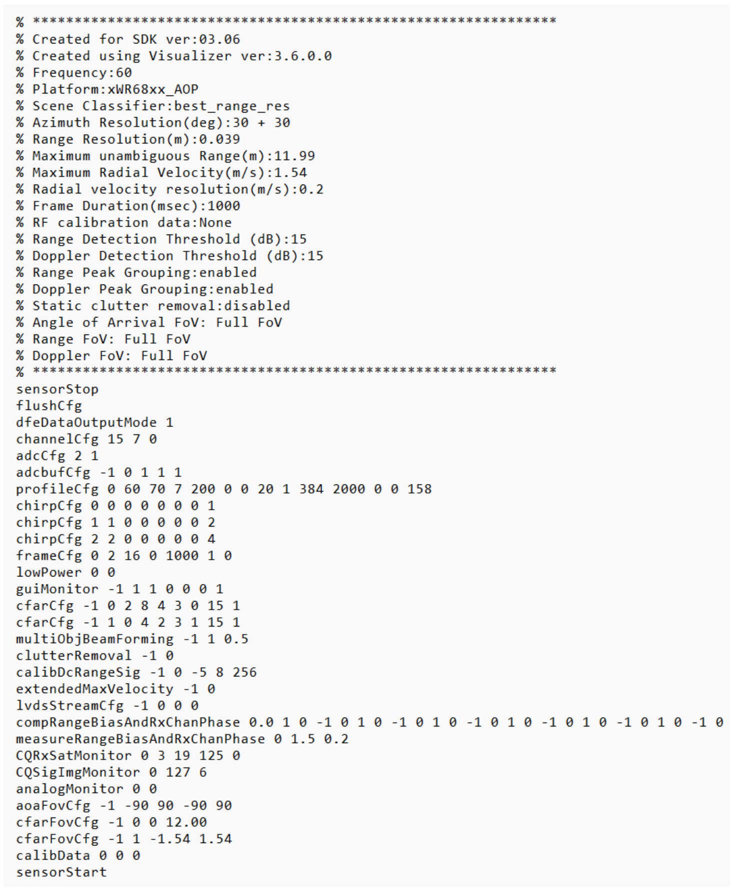
Desirable Configuration: Best Range Resolution
Frequency Band (GHz): 60-64
Calibration Data Save/Restore: None 0x1F0000

Console Messages

Scene Selection
Frame Rate (fps): 1 – 30: 10
Range Resolution (m): 0.039 – 0.047: 0.044
Maximum Unambiguous Range (m): 3.95 – 18.02: 9.02
Maximum Radial Velocity (m/s): 0.41 – 6.39: 1
Radial Velocity Resolution (m/s): 0.13: 0.13

Plot Selection
☑ Scatter Plot
☐ Range Azimuth Heat Map
☑ Range Profile
☐ Range Doppler Heat Map
☐ Noise Profile
☑ Statistics

Figure 5. Webpage screen of the mmWave_Demo_Visualizer browser-based application, Configure tab.



```
% ***************************************************************
% Created for SDK ver:03.06
% Created using Visualizer ver:3.6.0.0
% Frequency:60
% Platform:xWR68xx_AOP
% Scene Classifier:best_range_res
% Azimuth Resolution(deg):30 + 30
% Range Resolution(m):0.039
% Maximum unambiguous Range(m):11.99
% Maximum Radial Velocity(m/s):1.54
% Radial velocity resolution(m/s):0.2
% Frame Duration(msec):1000
% RF calibration data:None
% Range Detection Threshold (dB):15
% Doppler Detection Threshold (dB):15
% Range Peak Grouping:enabled
% Doppler Peak Grouping:enabled
% Static clutter removal:disabled
% Angle of Arrival FoV: Full FoV
% Range FoV: Full FoV
% Doppler FoV: Full FoV
% ***************************************************************
sensorStop
flushCfg
dfeDataOutputMode 1
channelCfg 15 7 0
adcCfg 2 1
adcbufCfg -1 0 1 1 1
profileCfg 0 60 70 7 200 0 0 20 1 384 2000 0 0 158
chirpCfg 0 0 0 0 0 0 0 1
chirpCfg 1 1 0 0 0 0 0 2
chirpCfg 2 2 0 0 0 0 0 4
frameCfg 0 2 16 0 1000 1 0
lowPower 0 0
guiMonitor -1 1 1 0 0 0 1
cfarCfg -1 0 2 8 4 3 0 15 1
cfarCfg -1 1 0 4 2 3 1 15 1
multiObjBeamForming -1 1 0.5
clutterRemoval -1 0
calibDcRangeSig -1 0 -5 8 256
extendedMaxVelocity -1 0
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 19 125 0
CQSigImgMonitor 0 127 6
analogMonitor 0 0
aoaFovCfg -1 -90 90 -90 90
cfarFovCfg -1 0 0 12.00
cfarFovCfg -1 1 -1.54 1.54
calibData 0 0 0
sensorStart
```

Figure 6. Example of the content of a configuration file.

In figure 5, for the “Setup Details”, the xWR68xx_AOP was selected as the platform, matching the TI IWR6843AOP radar sensor used. The “SDK version” is a drop- down menu that asks the user to select the SDK version that the mmWave device was flashed with. In case there is a mismatch in the selected SDK version, the graphical user interface (GUI) will show an error when using the “Send Config to mmWave Device button” or “Load config from PC and send” buttons [6]. The antenna configuration is also a drop- down menu, figure 5 shows the correct configuration to be selected (4 Rx, 3 Tx).

The “Desirable Configuration” is a drop – down menu that allows the user to select the specification that is ensured to be met above the other parameters. This is because certain parameters affect the values of other parameters. When a particular desirable configuration is selected, the range or values of the parameters given under the “Scene Selection” (figure 5) are adjusted accordingly. Moreover, how different parameters affect other parameters also change. Table 1 summarizes each parameter’s dependent parameters and the relation between them [6]. As the table shows, the desirable configuration options include best range resolution, best velocity resolution, or best range. The default settings uses the best range resolution option (figure 5).

The frequency band is set by default depending on the selected platform. Under “Scene Selection,” the parameter frame rate describes the rate at which the frames are shipped out of the mmWave radar sensor [6]. A frame consists of N chirps, each frame is separated by a duration called the inter frame time. Figure 7 shows a typical FMCW chirp, and figure 8 shows a typical frame structure, showing N chirps (frame) and M chirps (anther frame) separated by the inter frame time duration.

The frame period (T_f) can therefore be expressed as N*Tc + $T_{IFT,}$ where Tc is the chirp time or sweep time and T_{IFT} is the inter frame time. During T_{IFT}, the radar processes and transmits the captured data. A higher frame rate reduces the frame periodicity (T_f) and hence T_{IFT}, which may lead to data loss if the resulting T_{IFT} is insufficient to transmit the large amount of captured data over the relatively low baud rate of the transmission protocol [6]; it is therefore suggested to select a relatively low frame rate (3 – 10 fps) [6].

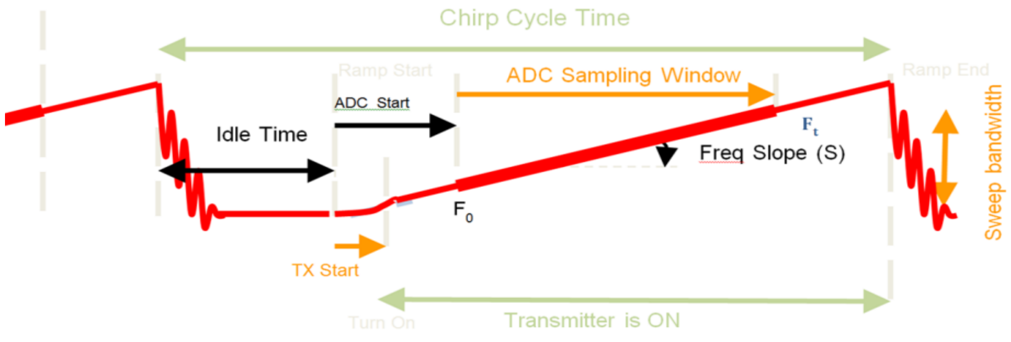


Figure 7. Typical FMCW chirp [6].

Table 1. Summary of the Dependency on the Relations Between the Radar Configuration Parameters for a given Desired Configuration

	Scene Selected											
	Best Range Resolution				Best Range Resolution						Best Range Resolution	
	Dependent Parameters		Relation		Dependent Parameters			Relation			Dependent Parameters	Relation
Configuration Parameters	P1	P2	P1	P2	P1	P2	P3	P1	P2	P3	P1	P1
Frame Rate (fps)	The minimum value of the maximum radial velocity	-	Proportional relation. The greater the frame rate, the larger is the minimum maximum radial velocity.	-	The minimum value of the maximum radial velocity	Radial Velocity Resolution	Range Resolution	Proportional relation. The greater the frame rate, the larger is the minimum maximum radial velocity.	The lower the frame rate, the finer the radial velocity resolution.	Higher frame rates provide more finer range resolution options.	The minimum value of the maximum radial velocity	Proportional relation. The greater the frame rate, the larger is the minimum maximum radial velocity.
Range Resolution (m)	Maximum Unambiguous Range	Maxim-um Radial Velocity	Proportional. The finer the range resolution is, the longer is the maximum unambiguous range.	Inverse relation. The finer the range resolution, the lower the Max radial velocity.	Maximum Unambig-uous Range	-	-	Proportional. The finer the range resolution, the shorter the maximum unambiguous range.	-	-	Maximum Radial Velocity	The coarser the selected range resolution, the more and higher the options for the maximum radial velocity.
Maximum Unambiguous Range (m)	Radial Velocity Resolution (drop – down menu)	-	The longer the maximum unambiguous range, the lesser the number of better options for the radial velocity resolution.	-	Range Resolution	-	-	The longer the Max unambiguous range, the finer the range resolution.	-	-	Range Resolution	The lower the maximum unambiguous range, the lower but finer options for range resolution.
Maximum Radial Velocity (m/s)	Radial Velocity Resolution	-	Inverse relation. The lower the maximum radial velocity, the finer the radial velocity resolution.	-	Range Resolution	Maximum Unambiguous Range	-	Inverse relation. The greater the maximum radial velocity, the coarser the range resolution.	Inverse relation. The greater the maximum radial velocity, the shorter the maximum unambig-uous range.	-	Radial Velocity Resolution	The lower Max. radial velocity, the finer the radial velocity resolution.
Radial Velocity Resolution (m/s)	Drop – down menu, constrained by the other parameters.				A direct function of the frame rate. The radial velocity resolution is fixed to the optimal possible value for the selected frame rate.						Drop – down menu, constrained by the other parameters.	

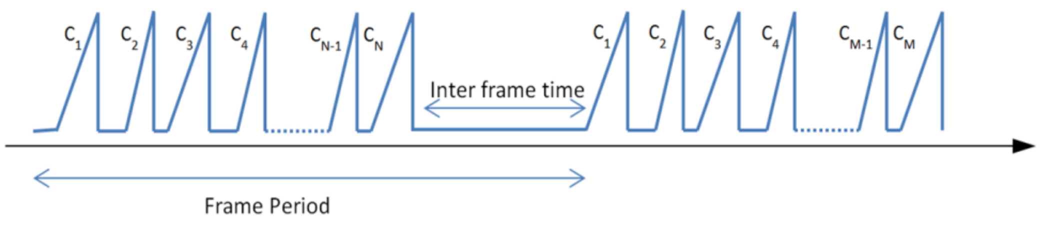


Figure 8. Typical frame structure [6].

The equations of the other four parameter under "Scene Selection" is given in Table 2 (the derivation of the equations is planned to be discussed in a separate report) [7]. Moreover, due to the complexity in the dependency of the parameters as a function of the selected desirable configuration, the relation between the parameters cannot be easily understood using the equations given. We may instead use Table 1 to study the relations and fine – tune the parameters to achieve the desired specifications. For that reason, for the current stage, the default values of the parameters have been used.

The "Plot Selection" is specified when interfacing the radar module with the PC. After having understood the mmWave device configuration parameters and how to control them to obtain a particular requirement, next we explore using sending the configuration file to the sensor via the Raspberry Pi to test the success of the communication between it and the EVM.

Table 2. Radar Configuration Parameters Equations

	Resolution	Maximum
Range	$\frac{c}{2B}$	$\frac{cF_s}{2\mu}$
Radial Velocity	$\frac{\lambda}{2T_f}$	$\frac{\lambda}{4T_C}$

III. VI Testing the Communication between the mmWave device and the Raspberry Pi

After having configured the radar parameters according to a given application specification, click on "SAVE CONFIG TO PC" found at the bottom of the screen of the configure tab (figure 5). This saves a file with .cfg extension. Upload this file into the Raspberry Pi using a USB. Run the command given below on terminal to determine the directory to the uploaded configuration file.

```
find ~ -name "*.cfg"
```

The Python code in Appendix A verifies access to the mmWave device's serial ports, sends the configuration file, and reads incoming data. Use the output of the terminal command given above (e.g. /home/Shayma/sdp/bin/config_1.cfg) and copy it in place of the configuration file name in the code. Use the output of the terminal command given in section III. III for lines five and six of the code, respectively.

Since the code does not parse data, we expect the Raspberry Pi to print gibberish, as shown in Figure 9. Once the Raspberry Pi is verified to successfully communicate with the sensor module, we may upload the data parsing code with lesser potential issues to debug. The modifications and the running of the data parsing code is explored in the next section.

III. VII The Data Parsing Code

The used data parsing code is given in Appendix B, which was taken from the github source given at source [8]. The code has been modified to display the velocity versus range plot. As with the previous section, use the output of the terminal command given in section III. III for the (CLI) and (DATA) ports directory, and the output of the terminal command given in section III. VI for the configuration file directory. The following section shows the XY and velocity versus range plots obtained for a particular scenario.

```
Serial ports are open
Configuration file sent
Radar Data:Ch
Radar Data: 0
Radar Data: 2=\S<s!=-0!>\=
                    ::t!
Radar Data: a
Radar Data:
Radar Data: Q
Radar Data: |                    -     :
Radar DatY□ U              @
Radar Data:
Radar Data:
□y<L*VKXETlJ'8YIN@I-CGR^m          □"
                  '$□@O35YH2LQIzGYo>o
                                ?]u.b}'EX6VM:?4SnQ
e]8                                              NZJ;U}4
Radar Data:                                               D:c]*=59       d3U
Radar Data: =c.q0>2(/tWqL">Vyi)n
$@pkE~j1!*0 Mg=@V>|,DSl:I□LLH(UfQ>/6,E\R3N,M?<r~$?Q)FL`9\.w2J~$@<;&[
```

Figure 9. Terminal output when running the Python code given in appendix A.

IV. Results

Figures 10(a) – 10(l) shows the data parsing code output plots for the given scenario. When our team member (Engy) is not moving (figure 10(a)), the XY plot (figure 10(b)) shows a total of six detected objects, and the velocity vs. range plot (figure 10(c)) shows the detected objects to have a velocity of zero. When the team member starts to move (figure 10(d)), the velocity of three dots (at range = 4 m) becomes non – zero (figure f), and the XY plot (figure 10(e)) displays new points around the same range (points with coordinates (~4 m, ~1m)).

As the team member moves closer and closer (figure 10(g) and 10(j)), the range of the nonzero velocity points and the x- coordinates of the XY plot decreases, as can be observed in figures 10(h) and 10(k) for the XY plots, and figures 10(i) and 10(l) for the velocity versus range plots. On terminal, the following is outputted continuously for each of the data packets parsed.

```
{'numObj': 6, 'range': [0.6104857094483974, 1.0901529643987067, 1.2209713293732982,
2.6599731940321663, 8.459586957524877, 0.17442446723933996], 'x': [0.12879982590675354,
0.8871416449546814, 0.8831987977027893, 0.16034263372421265, -2.2947397232055664,
0.015771405771374702], 'y': [0.594185471534729, 0.6301546096801758, 0.8027286529541016,
1.12474524974823, 5.371487140655518, 0.11358580738306046], 'z': [0.05519992485642433, -
0.06571419537067413, 0.2575996518135071, -2.405139446258545, 6.1193060874938965,
0.13142839074134827], 'v': [0.0, 0.0, 0.0, 0.0, 0.0, 0.36213648319244385]}
Bytes read: 768
```

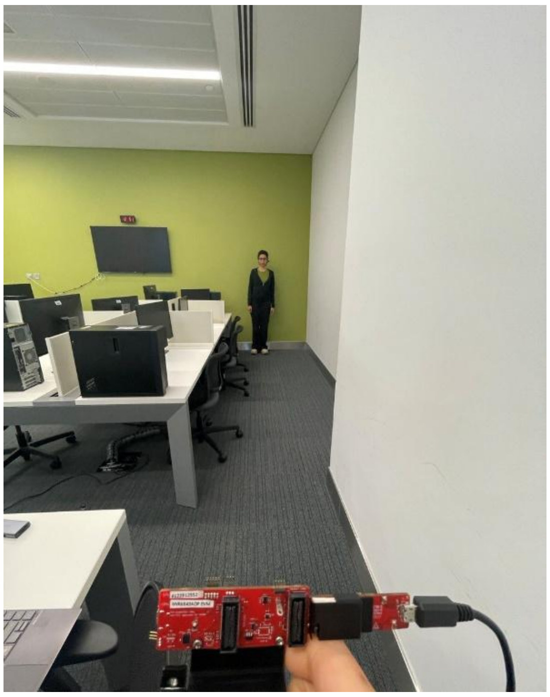


Figure 10(a). First scenario frame. Team member is static.

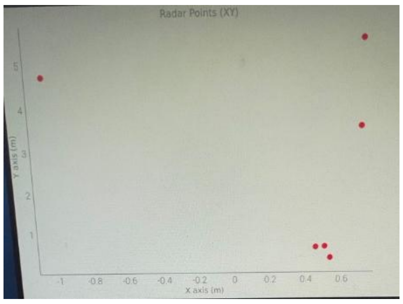


Figure 10(b). XY plot corresponding to the first scenario frame.

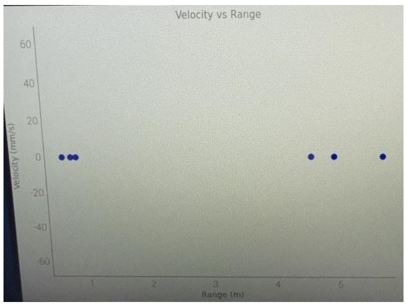


Figure 10(c). Velocity versus Range plot corresponding to the first scenario frame.

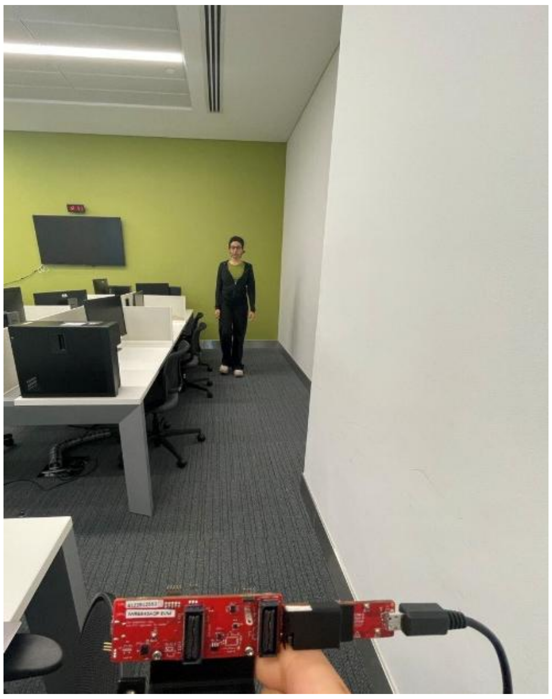


Figure 10(d). Second scenario frame.

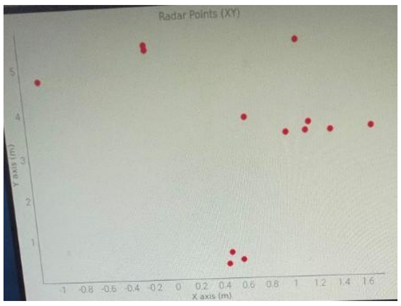


Figure 10(e). XY plot corresponding to the second scenario frame.

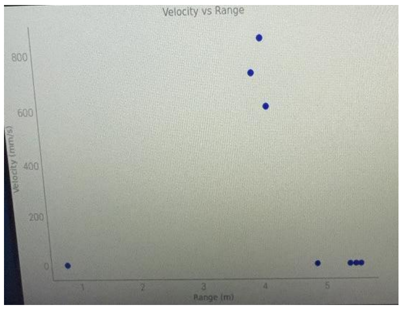


Figure 10(f). Velocity versus Range plot corresponding to the second scenario frame.

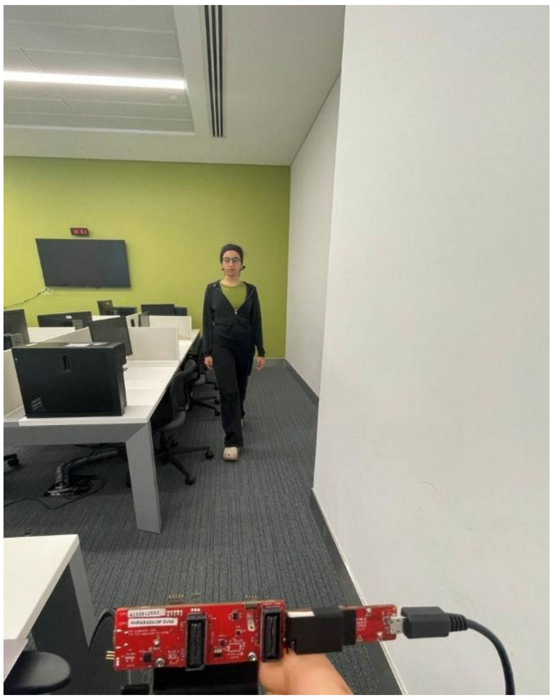


Figure 10(g). Third scenario frame.

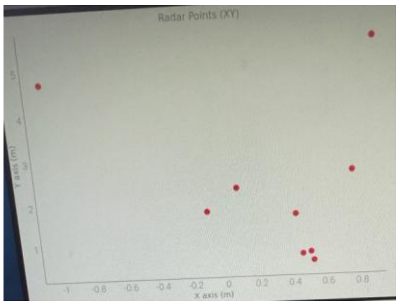


Figure 10(h). XY plot corresponding to the third scenario frame.

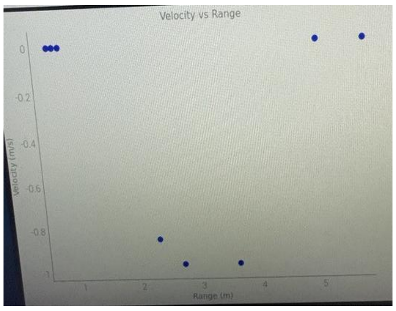


Figure 10(i). Velocity versus Range plot corresponding to the third scenario frame.

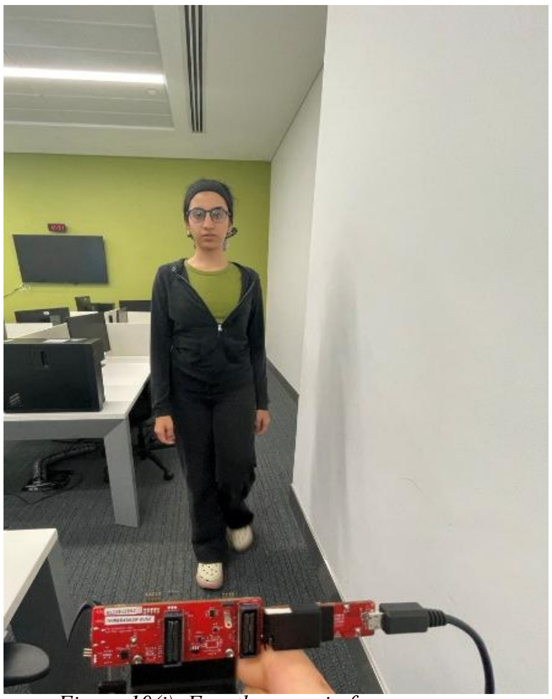


Figure 10(j). Fourth scenario frame.

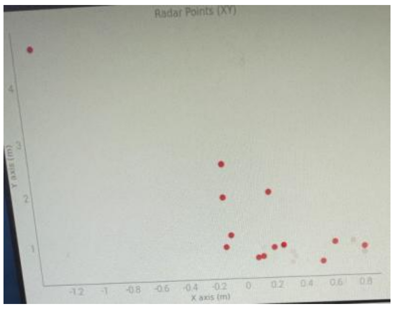


Figure 10(k). XY plot corresponding to the fourth scenario frame.

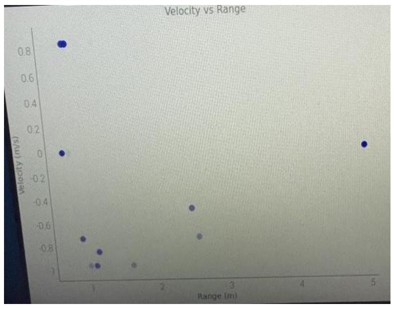


Figure 10(l). Velocity versus Range plot corresponding to the fourth scenario frame.

V. References

[1]. sgs-weather-and-environmental-systems, “TI-mmWave-SDK/mmwave_sdk_01_02_00_05/docs/mmwave_sdk_user_guide.pdf at master · sgs-weather-and-environmental-systems/TI-mmWave-SDK,” *GitHub*, 2018. https://github.com/sgs-weather-and-environmental-systems/TI-mmWave-SDK/blob/master/mmwave_sdk_01_02_00_05/docs/mmwave_sdk_user_guide.pdf (accessed Mar. 20, 2025).

[2]. “Hardware Setup for IWR6843AOP,” *Ti.com*, 2023. https://www.ti.com/video/6205846491001 (accessed Mar. 20, 2025).

[3]. “Silicon Labs,” *Silabs.com*, 2024. https://www.silabs.com/developer-tools/usb-to-uart-bridge-vcp-drivers?tab=downloads (accessed Mar. 20, 2025).

[4]. “Radar Toolbox for mmWave Sensors,” *TI Resource Explorer* . https://dev.ti.com/tirex/explore/node?node=A__AEIJm0rwIeU.2P1OBWwlaA__radar_toolbox__1AslXXD__LATEST (accessed Mar. 20, 2025).

[5]. “Out Of Box Demo User Guide,” *Ti.com*, 2025. https://dev.ti.com/tirex/explore/node?node=A__AXAenV2u4woV.FhTlAk68Q__radar_toolbox__1AslXXD__LATEST (accessed Mar. 20, 2025).

[6]. “mmWave Demo Visualizer User’s Guide mmWave Demo Visualizer,” 2017. Accessed: Mar. 20, 2025. [Online]. Available: https://www.ti.com/lit/ug/swru529c/swru529c.pdf?ts=1742455933180&ref_url=https%253A%252F%252Fwww.google.com%252F

[7]. Q. Chaudhari, “FMCW Radar Part 3 - Design Guidelines | Wireless Pi,” *Wireless Pi*, Nov. 30, 2023. https://wirelesspi.com/fmcw-radar-part-3-design-guidelines/ (accessed Mar. 20, 2025).

[8]. kirkster96, “IWR6843-Read-Data-Python-MMWAVE-SDK/parser_mmw_demo.py at main · kirkster96/IWR6843-Read-Data-Python-MMWAVE-SDK,” *GitHub*, 2021. https://github.com/kirkster96/IWR6843-Read-Data-Python-MMWAVE-SDK/blob/main/parser_mmw_demo.py (accessed Mar. 20, 2025).

VI. Appendices

VI. I Appendix A – Python code for Verifying the Communication between the mmWave device and the Raspberry Pi

```
# Generated using the assistance of AI
import serial
import time

CMD_PORT = "/dev/ttyUSB0"
DATA_PORT = "/dev/ttyUSB1"

# Open command and data ports
cmd_uart = serial.Serial(CMD_PORT, baudrate=115200, timeout=1)
data_uart = serial.Serial(DATA_PORT, baudrate=921600, timeout=1)

if cmd_uart.is_open and data_uart.is_open:
    print("Serial ports are open")
else:
    print("Failed to open serial ports")

def send_config_file(filename):
    """Send a configuration file line by line to the radar."""

    with open(filename, 'r') as file:

        for line in file:

            cmd_uart.write(line.encode() + b'\r\n')  # Send each line

            time.sleep(0.05)  # Delay for processing

    print("Configuration file sent")

def read_data():

    """Read and print data from the radar."""

    while True:

        if data_uart.in_waiting > 0:

            data = data_uart.readline().decode(errors="ignore").strip()

            print("Radar Data:", data)

# Send the config file before reading data

send_config_file("/home/Shayma/config_2.cfg")

# Start reading radar data

read_data()
```

VI. II Appendix B – Data Parsing Code

```
# Adapted from a cited Github source and edited using the assistance of an AI model


import serial
import time
import numpy as np
import os
import sys
from PyQt5 import QtWidgets, QtCore
import pyqtgraph as pg
from pyqtgraph.Qt import QtGui
# import the parser function
from parser_mmw_demo import parser_one_mmw_demo_output_packet

# Change the configuration file name
configFileName = '/home/Shayma/sdp/config_2.cfg'

# Change the debug variable to use print()
DEBUG = False

# Constants
maxBufferSize = 2**15;
CLIport = {}
Dataport = {}
byteBuffer = np.zeros(2**15,dtype = 'uint8')
byteBufferLength = 0;
maxBufferSize = 2**15;
magicWord = [2, 1, 4, 3, 6, 5, 8, 7]
detObj = {}
frameData = {}
currentIndex = 0
# word array to convert 4 bytes to a 32 bit number
word = [1, 2**8, 2**16, 2**24]

# Function to configure the serial ports and send the data from
# the configuration file to the radar
def serialConfig(configFileName):

    global CLIport
    global Dataport
    # Open the serial ports for the configuration and the data ports

    # Raspberry pi
    CLIport = serial.Serial('/dev/ttyUSB0', 115200)
    Dataport = serial.Serial('/dev/ttyUSB1', 921600)

    print("CLIport open:", CLIport.isOpen())  # Debugging line
    print("Dataport open:", Dataport.isOpen())  # Debugging line

    # Read the configuration file and send it to the board
    config = [line.rstrip('\r\n') for line in open(configFileName)]
    for i in config:
        CLIport.write((i+'\n').encode())
        print(i)
        time.sleep(0.01)

    return CLIport, Dataport

# Function to parse the data inside the configuration file
def parseConfigFile(configFileName):
    configParameters = {} # Initialize an empty dictionary to store the configuration parameters

    # Read the configuration file and send it to the board
    config = [line.rstrip('\r\n') for line in open(configFileName)]
    for i in config:

        # Split the line
        splitWords = i.split(" ")

        # Hard code the number of antennas, change if other configuration is used
        numRxAnt = 4
        numTxAnt = 3

        # Get the information about the profile configuration
        if "profileCfg" in splitWords[0]:
            startFreq = int(float(splitWords[2]))
            idleTime = int(splitWords[3])
            rampEndTime = float(splitWords[5])
            freqSlopeConst = float(splitWords[8])
            numAdcSamples = int(splitWords[10])
            numAdcSamplesRoundTo2 = 1;

            while numAdcSamples > numAdcSamplesRoundTo2:
                numAdcSamplesRoundTo2 = numAdcSamplesRoundTo2 * 2;

            digOutSampleRate = int(splitWords[11]);

        # Get the information about the frame configuration
        elif "frameCfg" in splitWords[0]:

            chirpStartIdx = int(splitWords[1]);
            chirpEndIdx = int(splitWords[2]);
            numLoops = int(splitWords[3]);
            numFrames = int(splitWords[4]);
            framePeriodicity = int(splitWords[5]);


    # Combine the read data to obtain the configuration parameters
    numChirpsPerFrame = (chirpEndIdx - chirpStartIdx + 1) * numLoops
    configParameters["numDopplerBins"] = numChirpsPerFrame / numTxAnt
    configParameters["numRangeBins"] = numAdcSamplesRoundTo2
    configParameters["rangeResolutionMeters"] = (3e8 * digOutSampleRate * 1e3) / (2 *
freqSlopeConst * 1e12 * numAdcSamples)
    configParameters["rangeIdxToMeters"] = (3e8 * digOutSampleRate * 1e3) / (2 * freqSlopeConst *
1e12 * configParameters["numRangeBins"])
    configParameters["dopplerResolutionMps"] = 3e8 / (2 * startFreq * 1e9 * (idleTime +
rampEndTime) * 1e-6 * configParameters["numDopplerBins"] * numTxAnt)
```

```
    configParameters["maxRange"] = (300 * 0.9 * digOutSampleRate)/(2 * freqSlopeConst * 1e3)
    configParameters["maxVelocity"] = 3e8 / (4 * startFreq * 1e9 * (idleTime + rampEndTime) * 1e-6
* numTxAnt)

    return configParameters

####################################################################################
# USE parser_mmw_demo SCRIPT TO PARSE ABOVE INPUT FILES
####################################################################################
def readAndParseData14xx(Dataport, configParameters):
    #load from serial
    global byteBuffer, byteBufferLength

    # Initialize variables
    magicOK = 0 # Checks if magic number has been read
    dataOK = 0 # Checks if the data has been read correctly
    frameNumber = 0
    detObj = {}

    readBuffer = Dataport.read(Dataport.in_waiting)
    byteVec = np.frombuffer(readBuffer, dtype = 'uint8')
    byteCount = len(byteVec)
    print(f"Bytes read: {byteCount}")

    # Check that the buffer is not full, and then add the data to the buffer
    if (byteBufferLength + byteCount) < maxBufferSize:
        byteBuffer[byteBufferLength:byteBufferLength + byteCount] = byteVec[:byteCount]
        byteBufferLength = byteBufferLength + byteCount

    # Check that the buffer has some data
    if byteBufferLength > 16:

        # Check for all possible locations of the magic word
        possibleLocs = np.where(byteBuffer == magicWord[0])[0]
        print(f"Byte buffer: {byteBuffer[:byteBufferLength]}")

 # Confirm that is the beginning of the magic word and store the index in startIdx
        startIdx = []
        for loc in possibleLocs:
            check = byteBuffer[loc:loc+8]
            if np.all(check == magicWord):
                startIdx.append(loc)

        # Check that startIdx is not empty
        if startIdx:

            # Remove the data before the first start index
            if startIdx[0] > 0 and startIdx[0] < byteBufferLength:
                byteBuffer[:byteBufferLength-startIdx[0]] = byteBuffer[startIdx[0]:byteBufferLength]
                byteBuffer[byteBufferLength-startIdx[0]:] = np.zeros(len(byteBuffer[byteBufferLength-
startIdx[0]:]),dtype = 'uint8')
                byteBufferLength = byteBufferLength - startIdx[0]

            # Check that there have no errors with the byte buffer length
            if byteBufferLength < 0:
```

```
            byteBufferLength = 0

        # Read the total packet length
        totalPacketLen = np.matmul(byteBuffer[12:12+4],word)
        # Check that all the packet has been read
        if (byteBufferLength >= totalPacketLen) and (byteBufferLength != 0):
            magicOK = 1

    # If magicOK is equal to 1 then process the message
    if magicOK:
        # Read the entire buffer
        readNumBytes = byteBufferLength
        if(DEBUG):
            print("readNumBytes: ", readNumBytes)
        allBinData = byteBuffer
        if(DEBUG):
            print("allBinData: ", allBinData[0], allBinData[1], allBinData[2], allBinData[3])

        # init local variables
        totalBytesParsed = 0;
        numFramesParsed = 0;

        # parser_one_mmw_demo_output_packet extracts only one complete frame at a time
        # so call this in a loop till end of file
        #
        # parser_one_mmw_demo_output_packet function already prints the
        # parsed data to stdio. So showcasing only saving the data to arrays
        # here for further custom processing
        parser_result, \
        headerStartIndex,  \
        totalPacketNumBytes, \
        numDetObj,  \
        numTlv,  \
        subFrameNumber,  \
        detectedX_array,  \
        detectedY_array,  \
        detectedZ_array,  \
        detectedV_array,  \
        detectedRange_array,  \
        detectedAzimuth_array,  \
        detectedElevation_array,  \
        detectedSNR_array,  \
        detectedNoise_array = parser_one_mmw_demo_output_packet(allBinData[totalBytesParsed::1],
readNumBytes-totalBytesParsed,DEBUG)

        # Check the parser result
        if(DEBUG):
            print ("Parser result: ", parser_result)
        if (parser_result == 0):
            totalBytesParsed += (headerStartIndex+totalPacketNumBytes)
            numFramesParsed+=1
            if(DEBUG):
                print("totalBytesParsed: ", totalBytesParsed)

#####################################################################################
```

```
            # TODO: use the arrays returned by above parser as needed.
            # For array dimensions, see help(parser_one_mmw_demo_output_packet)
            # help(parser_one_mmw_demo_output_packet)

################################################################################


            # For example, dump all S/W objects to a csv file

            import csv
            if (numFramesParsed == 1):
                democsvfile = open('mmw_demo_output.csv', 'w', newline='')
                demoOutputWriter = csv.writer(democsvfile, delimiter=',', quoting=csv.QUOTE_NONE)
                demoOutputWriter.writerow(["frame","DetObj#","x","y","z","v","snr","noise"])

            for obj in range(numDetObj):
                demoOutputWriter.writerow([numFramesParsed-1, obj, detectedX_array[obj],\
                                           detectedY_array[obj],\
                                           detectedZ_array[obj],\
                                           detectedV_array[obj],\
                                           detectedSNR_array[obj],\
                                           detectedNoise_array[obj]])

            detObj = {"numObj": numDetObj, "range": detectedRange_array, \
                      "x": detectedX_array, "y": detectedY_array, "z": detectedZ_array, "v":
detectedV_array}
            dataOK = 1
        else:
            # error in parsing; exit the loop
            print("error in parsing this frame; continue")

        shiftSize = totalPacketNumBytes
        byteBuffer[:byteBufferLength - shiftSize] = byteBuffer[shiftSize:byteBufferLength]
        byteBuffer[byteBufferLength - shiftSize:] = np.zeros(len(byteBuffer[byteBufferLength -
shiftSize:]),dtype = 'uint8')
        byteBufferLength = byteBufferLength - shiftSize

        # Check that there are no errors with the buffer length
        if byteBufferLength < 0:
            byteBufferLength = 0
        # All processing done; Exit
        if(DEBUG):
            print("numFramesParsed: ", numFramesParsed)

    return dataOK, frameNumber, detObj
class MyWidget(pg.GraphicsView):

    def __init__(self, parent=None):
        super().__init__(parent=parent)

        self.mainLayout = QtWidgets.QVBoxLayout(self)

        # First Plot Layout (for x-y plot)
        self.plotView1 = pg.GraphicsView(self)
        self.setLayout(self.mainLayout)
        self.noDataLabel = QtWidgets.QLabel("No Data to Plot", self)
        self.noDataLabel.setAlignment(QtCore.Qt.AlignCenter)
        self.noDataLabel.setVisible(False)  # Initially hide the label
        self.mainLayout.addWidget(self.noDataLabel)  # Add the label to the layout

        self.timer = QtCore.QTimer(self)
        self.timer.setInterval(100) # in milliseconds
        self.timer.start()
        self.timer.timeout.connect(self.onNewData)
        self.plotItem1 = pg.PlotItem(title="Radar Points (XY)")
        self.plotView1.setScene(pg.GraphicsScene())  # Set a new scene for the first plot
        self.plotView1.scene().addItem(self.plotItem1)
        self.plotView1.setBackground('w')  # Set background color of the GraphicsView

        self.plotView1.setMinimumSize(400, 300)

        self.mainLayout.addWidget(self.plotView1)
        self.plotItem1.setRange(QtCore.QRectF(-10, -10, 40, 40))  # Adjust plot range for the first plot

        # Create data item for first plot
        self.plotDataItem1 = self.plotItem1.plot([], pen=None, symbolBrush=(25, 125, 0),
symbolSize=5, symbolPen=None)

        self.plotView2 = pg.GraphicsView(self)  # Create a GraphicsView for the second plot
        self.plotItem2 = pg.PlotItem(title="Velocity vs Range")
        self.plotView2.setScene(pg.GraphicsScene())  # Set a new scene for the second plot
        self.plotView2.scene().addItem(self.plotItem2)
        self.plotView2.setBackground('w')  # Set background color of the GraphicsView
        self.plotView2.setMinimumSize(400, 300)

        self.mainLayout.addWidget(self.plotView2)
        self.plotItem2.setRange(QtCore.QRectF(0, 0, 100, 50))  # Adjust plot range for the second plot

        # Create data item for second plot
        self.plotDataItem2 = self.plotItem2.plot([], pen=None, symbolBrush=(0, 0, 255), symbolSize=5,
symbolPen=None)
    def setData(self, x, y):
        self.plotDataItem1.setData(x, y)

    def setData2(self, velocity, range_):
        self.plotDataItem2.setData(range_, velocity)
    # Funtion to update the data and display in the plot
    def update(self):
        dataOk = 0
        global detObj
        x = []
        y = []
        velocity = []
        range_ = []
        # Read and parse the received data
        dataOk, frameNumber, detObj = readAndParseData14xx(Dataport, configParameters)
        if dataOk:
            print("detObj:", detObj)  # Debugging line
        else:
            print("No data to parse.")  # Debugging line
        if dataOk and len(detObj["x"]) > 0:
            print(detObj)
            x = detObj["x"]
            y = detObj["y"]
            velocity = detObj["v"]
            range_ = detObj["range"]

        return dataOk, x, y, velocity, range_

    def onNewData(self):

        # Update the data and check if the data is okay
        dataOk,newx,newy,velocity,range_  = self.update()

        #if dataOk:
            # Store the current frame into frameData
            #frameData[currentIndex] = detObj
            #currentIndex += 1

        if dataOk and len(newx) > 0 and len(newy) > 0:
            # If data is valid and not empty, update the plot with new data
            self.setData(newx, newy)
            self.noDataLabel.setVisible(False)  # Hide the "No Data" label if data is available
        else:
            # If no data or invalid data, clear the plot and show "No Data to Plot"
            self.setData([], [])  # Clear the plot by passing empty lists
            self.noDataLabel.setVisible(True)  # Show the "No Data" label

        if dataOk and len(velocity) > 0 and len(range_) > 0:
            # Update the second plot with velocity vs range data
            self.setData2(velocity, range_)

def main():
    # Configurate the serial port
    CLIport, Dataport = serialConfig(configFileName)

    # Get the configuration parameters from the configuration file
    global configParameters
    configParameters = parseConfigFile(configFileName)

    app = QtWidgets.QApplication([])

    pg.setConfigOptions(antialias=False) # True seems to work as well
    win = MyWidget()
    win.show()
    win.resize(1500,1000)
    win.raise_()
    app.exec_()
    CLIport.write(('sensorStop\n').encode())
    CLIport.close()
    Dataport.close()

if __name__ == "__main__":
    main()
```