

UNIVERSITY OF WATERLOO



Infineon Radar Breathing Demo

Mackenzie Goodwin

Contents

Introduction	3
Radar Fundamentals	4
Radar Parameters	5
Range Parameters	6
Timing Characteristics	6
Velocity Parameter	6
Board Antenna	7
Infineon Chipset	9
BGT60TR13C Shield	10
BGT60TR13C Shield Connectors	10
BGT60TR13C Firmware	12
BGT60TR13C Antenna	12
MCU7 Baseboard	13
Installing Infineon mmWave Documentation	14
Radar SDK	18
Physical Layout	18
Range Parameters	19
Timing Characteristics	20
Velocity Parameter	20
Software Setup	21
Running the Server	22
Running Client	23
Processing	26
Auto Correlation Background	27
Auto Correlation Implementation	31
Auto Correlation Filtering	34
Test Setup	35
Results	36
Conclusion	38
Bibliography	39

Introduction

Within the last decade, mmWave radar technology operating in the frequency spectrum of 30GHz and above has become prevalent in consumer products. Back in 2019, Google released its next-generation Pixel 4 phone lineup including a Soli mmWave radar chip. In the case of the Pixel 4 smartphone, the radar was used to hand gestures called Motion Sense allowing hand gestures to skip songs, silence calls, and even interact with Pokémon.

These mmWave radars have a very wide range of applications such as in autonomous cars, gesture detection, and human breathing detection. This report will focus on the detection of breathing rate using the Infineon MCU baseboard and the BGT60TR13C shield containing the 60GHz mmWave radar system. With this radar's small size and minimal cost, these devices can be placed in many products for monitoring a user's breathing rate.

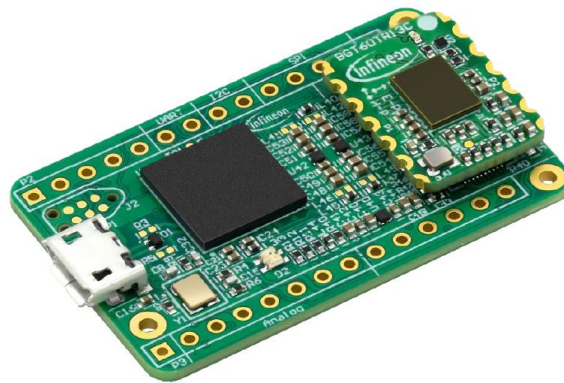


Figure 1: Baseboard and BGT60TR13C Shield [1]

In previous works such as A method of detection of respiration rate [2], methods for breathing algorithms are evaluated but the initial assumption is that the range bin is known. One of the largest challenges in detecting the breathing of a person is picking the optimal range to monitor for breathing. In a radar system, a person can exist within multiple ranges relative to the radar. These different ranges can contain many different artifacts and harmonics. An example would be a range that may contain more of the head when the area of interest is the chest. In most cases, work done in this field presents an algorithm for analyzing breathing rate time series but not how to pick the correct range pin.

This report presents a method for extracting the correct range bin to measure breathing rates. This eliminated the approximation that the range bin is already known. This algorithm can also accurately determine the breathing rate as well as the range bin. With the known range bin, other postprocessing methods can be applied.

The system consists of three sequential modules. First, the Infineon radar board performs measurements of the area and streams the data to the master device. This data is then sent to the master device which is a Raspberry PI 4B. The Raspberry PI formats the data and streams the data over a TCP connection to a

server. Finally, the server receives the data and performs the processing necessary to determine the breathing rate. Figure 2 summarises this system setup.

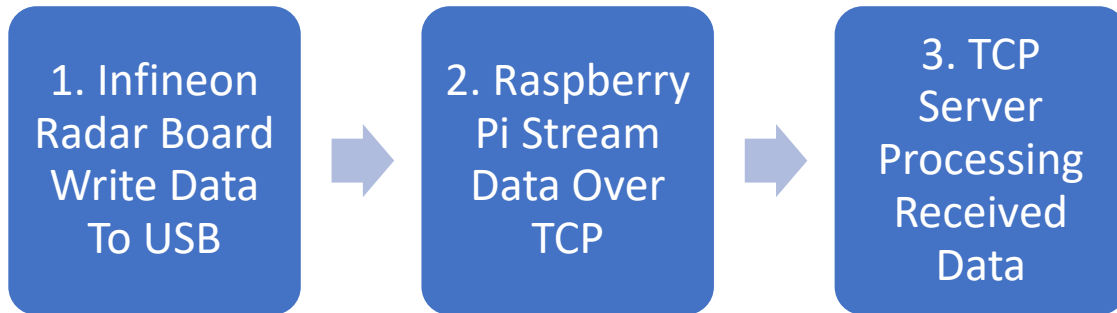


Figure 2: System Diagram

Radar Fundamentals

RADAR stands for Radio Detection and Ranging System. As the name implies a radar is a system that uses electromagnetic waves to detect objects and determine its ranges. Other information such as the speed of an object can also be determined using some extra processing.

At the most fundamental level, a RADAR is transmitting electromagnetic energy which travels towards and objects and interact with it. This object reflects some of this energy back towards the RADAR. This reflected energy then travels back towards the RADAR which is picked up by a receiver. Simple range calculations can be done since the transit time of the electromagnetic waves is known.

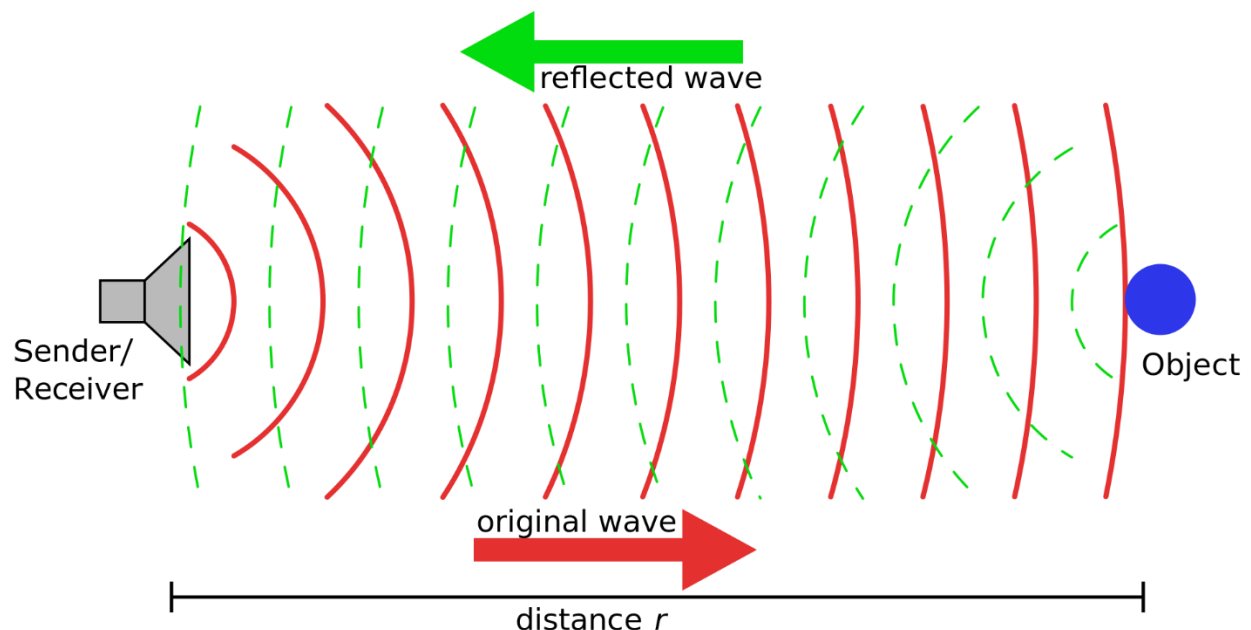


Figure 3: Radar Wave Propagation [3]

The Infineon mmWave radar uses the same principle but uses a technique called frequency modulation continues-wave signals. This works off the fact that the operating frequency is modulated in time and

linearly increases the operating frequency with time. This modulation scheme allows for fine range resolution which divides the distance in front of the radar into discrete ranges.

Radar Parameters

While the radar is running many parameters determine characteristics such as bandwidth, chirp time (length of modulated frequency), and operating frequencies that need to be set. Table 1 depicts all the required parameters for the Infineon radar chipset.

Table 1: Radar Parameters

Parameter Name	Description
Num_samples_per_chirp	This is the number of samples acquired during each chirp of a frame. The duration of a single chirp depends on the number of samples and the sampling rate.
Num_chirps_per_frame	This is the number of chirps a single data frame consists of.
Adc_samplerate_hz	This is the sampling rate of the ADC used to acquire the samples during a chirp. The duration of a single chirp depends on the number of samples and the sampling rate.
Frame_rate_us	This is the time that elapses between the beginnings of two consecutive frames. The reciprocal of this parameter is the frame rate.
Lower_frequency_kHz	This is the start frequency of the FMCW frequency ramp.
Upper_frequency_kHz	This is the end frequency of the FMCW frequency ramp.
Bgt_tx_power	This value controls the power of the transmitted RX signal. This is an abstract value between 0 and 31 without any physical meaning. Refer to the BGT60TR13AIP datasheet do learn more about the TX power BGT60TR13AIP is capable of.
Rx_antenna_mask	In this mask, each bit represents one RX antenna of BGT60TR13AIP. If a bit is set the, RX antenna is enabled during the chirps and the signal received through that antenna is captured.
Chirp_to_chirp_time_100ps	This is the time that elapses between the beginnings of two consecutive chirps in a frame.
If_gain_db	This is the amplification factor that is applied to the IF signal coming from the RF mixer before it is fed into the ADC.
Frame_end_delay_100ps	This parameter defines the delay after each frame in 100 picosecond steps. To set this value frame_period_us must be set to 0, otherwise, this value will be ignored.
Shape_end_delay_100ps	This parameter defines the delay after each shape in 100 picosecond steps. To set this value,

	chirp_to_chirp_time_100ps must be set to 0, otherwise, this value will be ignored.
--	--

Range Parameters

Two range parameters are important in setting the ADC samples per chirp and bandwidth. The parameter maximum range sets the maximum range the radar can see. Range resolution is the granularity that the ranges in front of the radar are divided into. These two parameters can be used to calculate the following:

$$samples\ per\ chirp = \frac{2 * maximum\ range}{range\ resolution}$$

Due to the Nyquist frequency, only half the range is evaluated requiring the range to be doubled. It is also important to note that in the Infineon chipset the samples per chirp must be a power of 2. This is due to the range FFT requiring a power of 2, otherwise the FFT would have to be zero-padded. Therefore,

$$range\ fft\ size = samples\ per\ chirp$$

Bandwidth is calculated off the range resolution of the radar. It can be calculated as followed,

$$bandwidth_{Hz} = \frac{c_0}{2 * range\ resolution}$$

Where c is the speed of light.

Using this information, the lower and upper frequency can also be calculated. The center frequency for the Infineon chip is set at 60.5 GHz.

$$upper\ frequency_{Hz} = center\ frequency_{Hz} + bandwidth_{Hz}$$

$$lower\ frequency_{Hz} = center\ frequency_{Hz} - bandwidth_{Hz}$$

The range per bin is also equal to the *range resolution*.

Timing Characteristics

A few timing characteristics determine the function of the radar. This is the frame rate of the radar and the chirp to chirp time. The frame rate controls how many frames happen per second. This requires that the frame time is long enough for the chirps to happen.

Chirp to chirp time depends on the center frequency of the radar and the maximum measurable velocity. This can be calculated as follow,

$$chirp\ to\ chirp\ time_{ps} = \frac{10^{10} * c_0}{4 * center\ frequency_{Hz} * maximum\ speed}$$

This equation is taken from Infineon with no source. As quoted from Infineon: "This formula was provided by algorithm team. Detailed information about this may be found in some papers. At this point, there is no documentation because the implementor of this function does not know those papers. Sorry".

Velocity Parameter

The maximum velocity and resolution determine the number of chirps performed per frame. This requires that the number of chirps is equal or smaller than 1024 since this is the maximum FFT size. It is

also important to note that num chirps per frame must be rounded to the closest power of two. A possible rounding method is the Bit Twiddling Hack by Sean Anderson [4].

$$\text{num chirps per frame} = \frac{2 * \text{maximum speed}}{\text{speed resolution}}$$

Board Antenna

As discussed, a radar system transmits and receives RF waves through a TX and RX antenna. These antennas can be specially designed for a certain purpose such as being highly directional or have a wider viewing angle. The radiation patterns of antennas can be explained using Maxwell Equations but are not relevant to this discussion.

Contained in the Infineon radar board is three receive antennas named Rx1, Rx2, and RX3 and a single transmit antenna named Tx. Figure 4 shows the layout of the Infineon board's TX and RX antennas. The RX antennas are microstrip antennas that act as the area that radiates the RF energy.

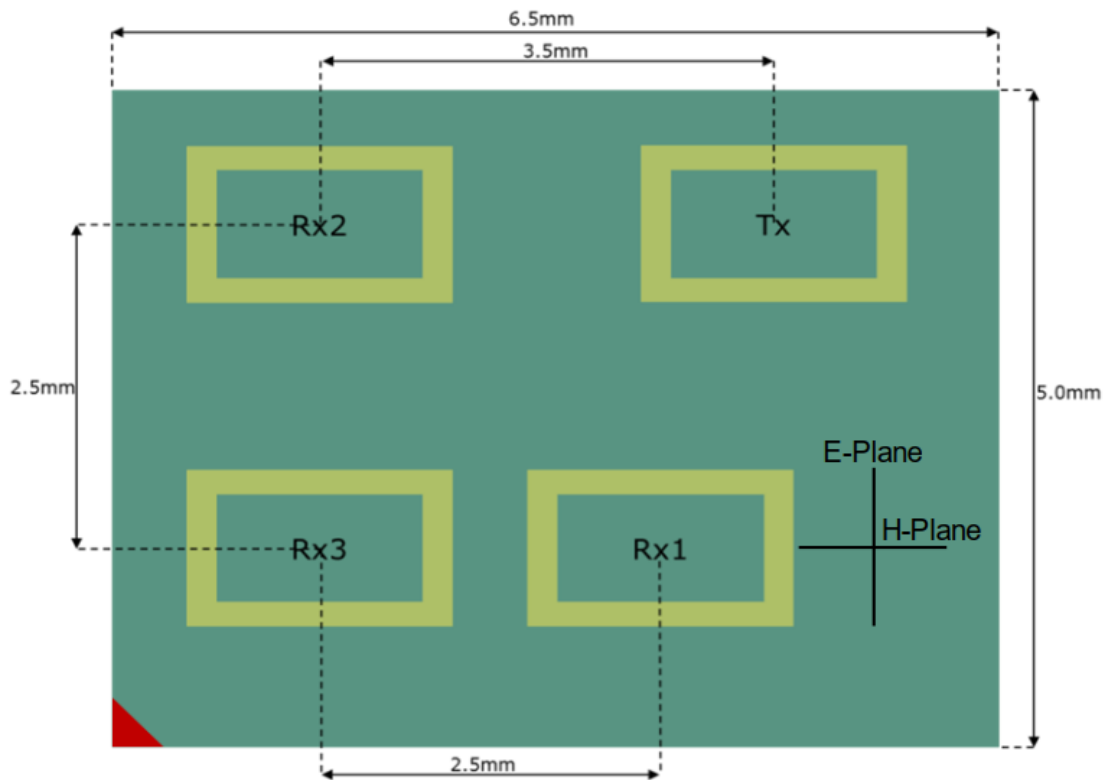


Figure 4: Infineon Antenna Layout [5]

Each of these microstrip antennas has its radiation pattern. These radiation patterns can be a complex three-dimensional structure with many lobes. In the case of the antenna for the Infineon board, this structure is roughly semi-spherical protruding normal to the PCB. Figure 5 is the radiation pattern of the Infineon chipset's RX antenna.

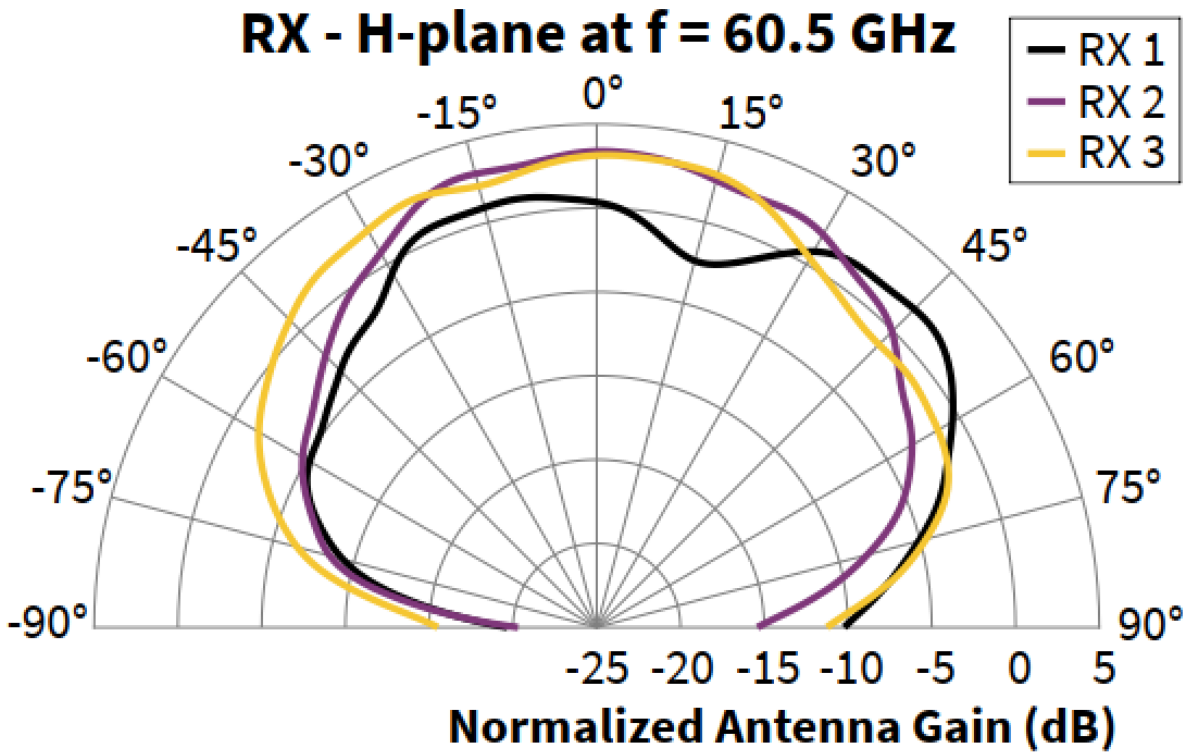
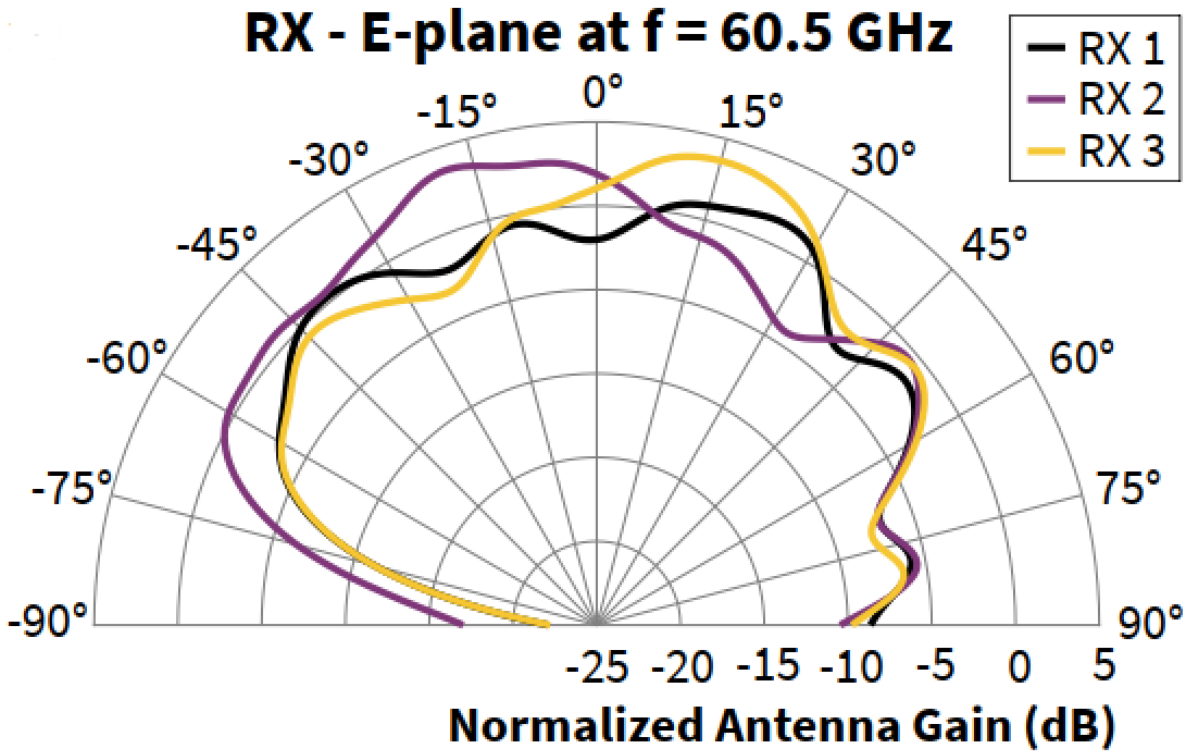


Figure 5: Patch Antenna Radiation Graph

With the addition of more antennas placed at specific distances apart, more information about the object can be obtained. These radiation patterns constructively interfere and can increase the range of the radar by increasing the gain of the antennas. Another use is the ability to determine the angle of arrival. Algorithms such as MUSIC can look at the phase relative to each antenna to determine what angle of arrival of a target. This works off the principle that the arrival time of the RF wave changes depending on the direction of arrival. Figure 6 shows an object that exists at an angle to the radar causing the RF waves to arrive at antenna 2 before antenna 1. This gives rise to a phase difference that can be measured.

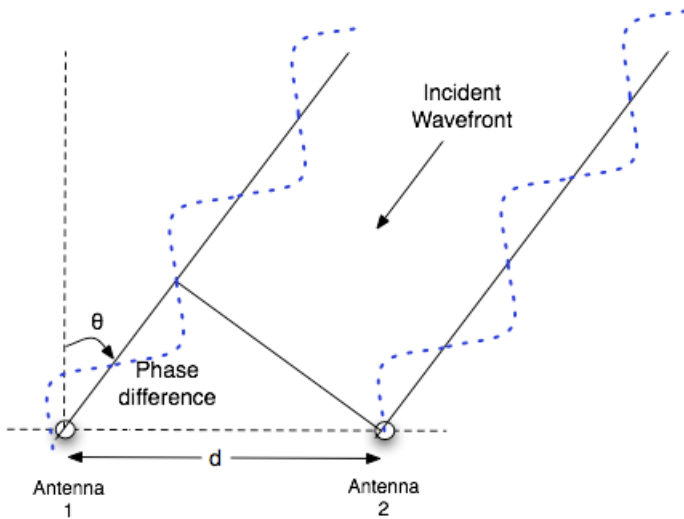


Figure 6: Direction of Arrival [6]

Infineon Chipset

The Infineon BGT60TR13C shield platform is a demonstration board for the Infineon's 60GHz radar solutions. The shield contains two PCB, one called the MCU7 baseboard and the other is the radar sensor board.

Figure 7 depicts the radar baseboard MCU7 (radar control board) in a) and the BGT60TR13C (mmWave radar) shield in b). Together these create a radar platform that can be used in many applications. Due to its flexibility, this platform can be used to develop new products quickly and to have a portable radar system for field testing. This system is targeted towards smartphones, laptops, wearables, and many others.

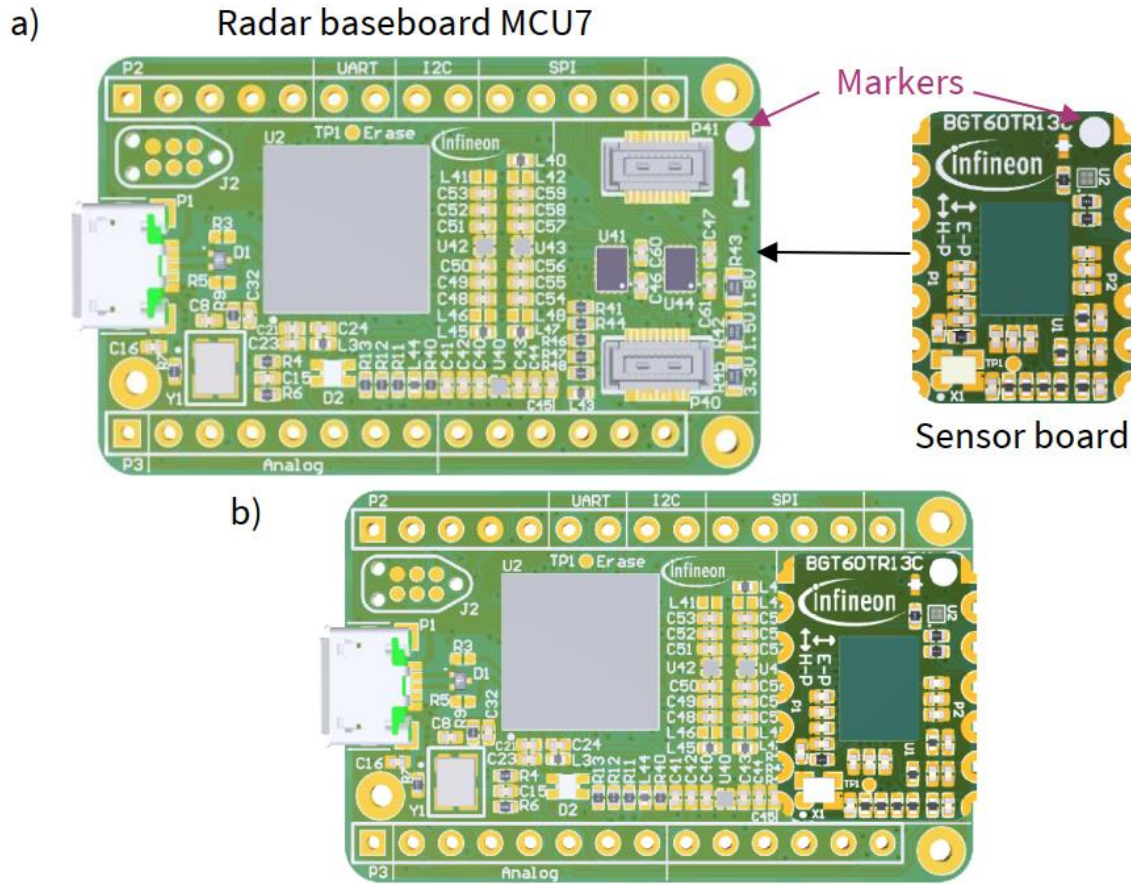


Figure 7: Infineon 60GHz Radar Platform [7]

BGT60TR13C Shield

The GT60TR13C shield is an extension board of the Infineon 60GHz radar system platform without a microcontroller. The shield can be driven by using an SPI connection and must be supplied with a few voltage rails to run the radar.

BGT60TR13C Shield Connectors

This system has two ways of connecting to the baseboard system. On the shield is castellated holes for the required signal to drive the BGT60TR13C. These can be accessible by installing make headers or other forms of through-hole connections. Figure 8 illustrates the pinout of the castellated hole connectors.

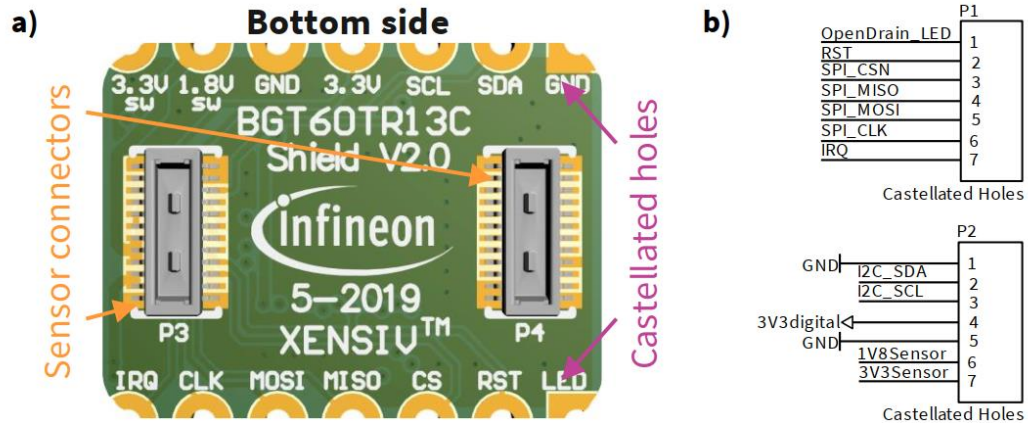


Figure 8: BGT60TR13C Shield Connectors [5]

The other form of headers for connecting to the carrier board is the sensor connectors which are surface mount high-density connections. These contain all the signals from the shield and are accessible using a DF40C-20DP-0.4V connector. Figure 9 depicts the pinout of the sensor connectors. Very carefully the shield board can be attached to the MCU7 baseboard using a specific technique as to not damage the delicate connectors.

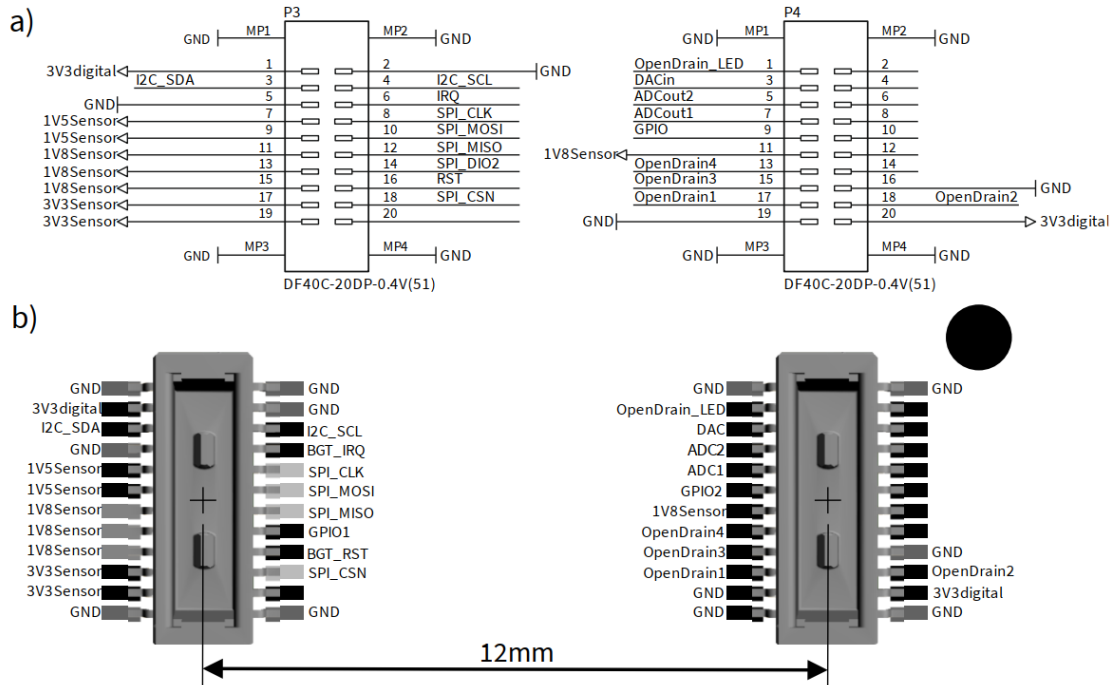


Figure 9: Sensor Connector Pinout [5]

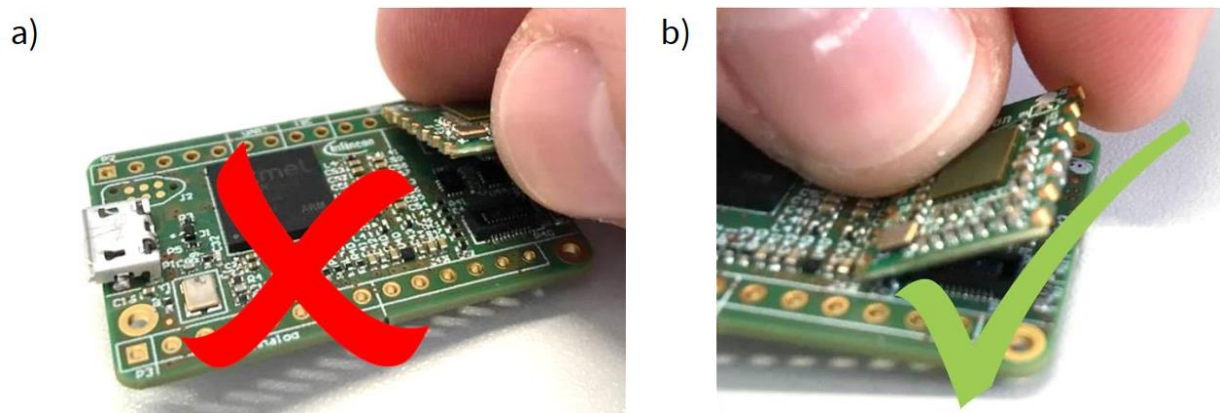


Figure 10: Inserting BGT60TR13C Shield [5]

BGT60TR13C Firmware

Contained on the BGT60TR13C shield is an EEPROM used to store data. This will contain information such as a board identifier.

When the MCU7 baseboard detects a shield attached, the driver layer is automatically configured for the shield. This includes configuring the chip to initiate SPI transfer and when BGT signals the availability of data via the IRQ line. More information such as SPI programming can be found under the IFX_AN600_BGT60TR13C Shield and IFX_BGT60TR13C_Preliminary_Datasheet datasheet.

BGT60TR13C Antenna

The BGT60TR13C contains a single TX antenna and three RX antennas. Each consists of a microstrip antenna that radiates the RF energy with a given radiation pattern. Figure 11 depicts the radiation pattern of each TX and RX microstrip antenna in the E and H-plane at 60.5GHz.

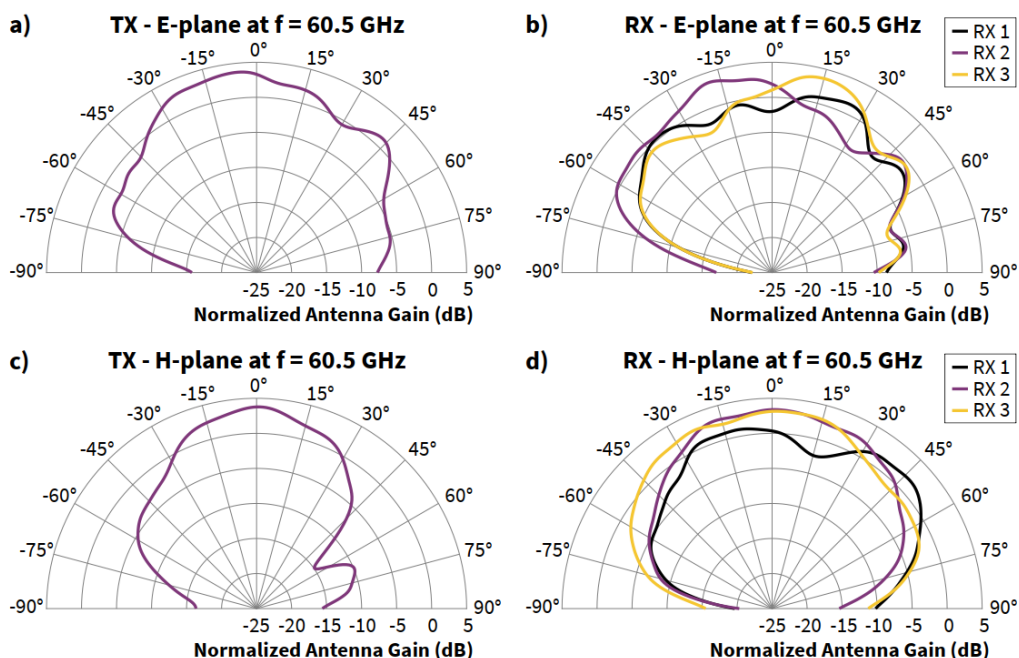


Figure 11: BGT60TR13C Radiation Pattern [8]

This shows the directionality of the radar with the specifications of the antenna package contained in Table 2.

Table 2: Antenna Specification

Spec	Unit	Value			Condition
Parameter		Min	Typ	Max	
HPBW_RX_H	Deg	20	35	50	Half-power beamwidth of a single RX antenna in the H-plane direction.
HPBW_TX_E	Deg	50	65	80	Half-power beamwidth of a single TX antenna in the E-plane direction
HPBW_TX_H	Deg	25	40	55	Half-power beamwidth of a single TX antenna in the H-plane direction
D_Rx_RX	λ_0		0.5		Distance between RX antennas at 60 GHz

MCU7 Baseboard

The radar baseboard MCU7 is the carrier board for the BGT60TR13C radar sensor. This board is for translating the SPI interface from the radar to a USB interface and other functions such as programming the radar sensor.

Onboard the device is an ARM Cortex-M7 MCU which acts as the central processor for the board. Its main functions are to manage the radar sensor and translate the data from SPI to the USB interface. Figure 12 illustrates the function of the ARM Cortex M7 in the system and peripherals it drives.

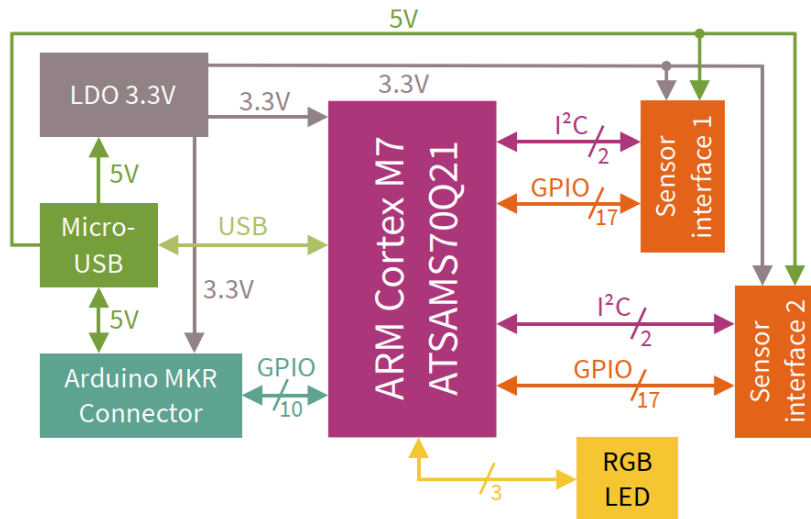


Figure 12: MCU7 Baseboard Block Diagram [7]

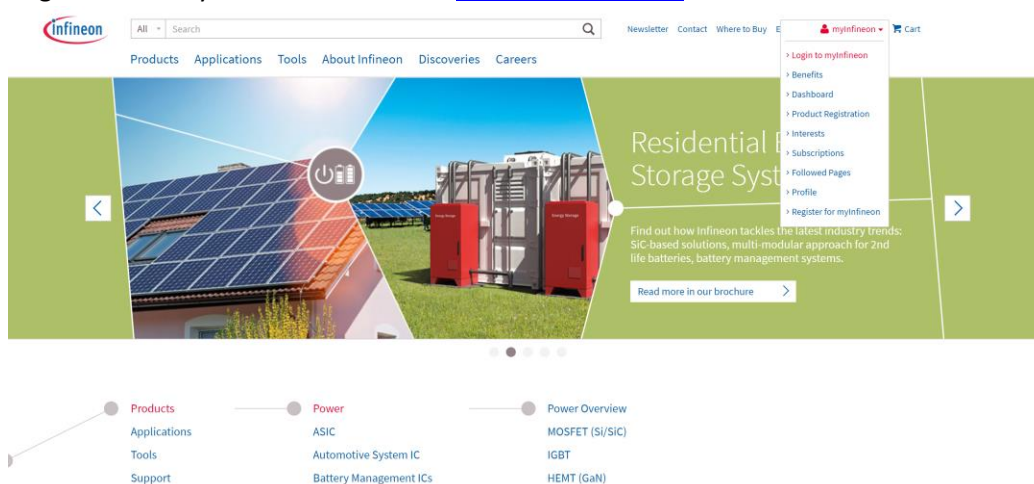
The board will not provide power to the sensor shield if the incorrect sensor is detected. This is done by the MCU reading the EEPROM off the BGT60TR13C shield to detect the type of device that is connected. If the sensor that is plugged in is correct the power supply will be enabled starting the programming process for the shield.

More information for the MCU7 carrier board can be found in the datasheet IFX_AN599 - Radar Baseboard MCU7.

Installing Infineon mmWave Documentation

To have access to datasheets and extra documentation the Infineon Toolbox must be installed. The instructions below describe this process.

1. Register for a myInfineon account on www.infineon.com



2. Once registered the Infineon Toolbox will be installed using the link below:
<https://www.infineon.com/cms/en/tools/landing/infineontoolbox.html?redird=102781>
3. Once Infineon Toolbox has finished installing launch the program and press login. This will be the account you registered in step 1

- Once logged in at the top right press the setting. In the default update site change the URL to: <https://itoolspriv.infineon.com/toolbox/updatesite/radarsdk>

Configuration menu ✕

Updatesite Proxy

Default update site

▼

Reset Clear History

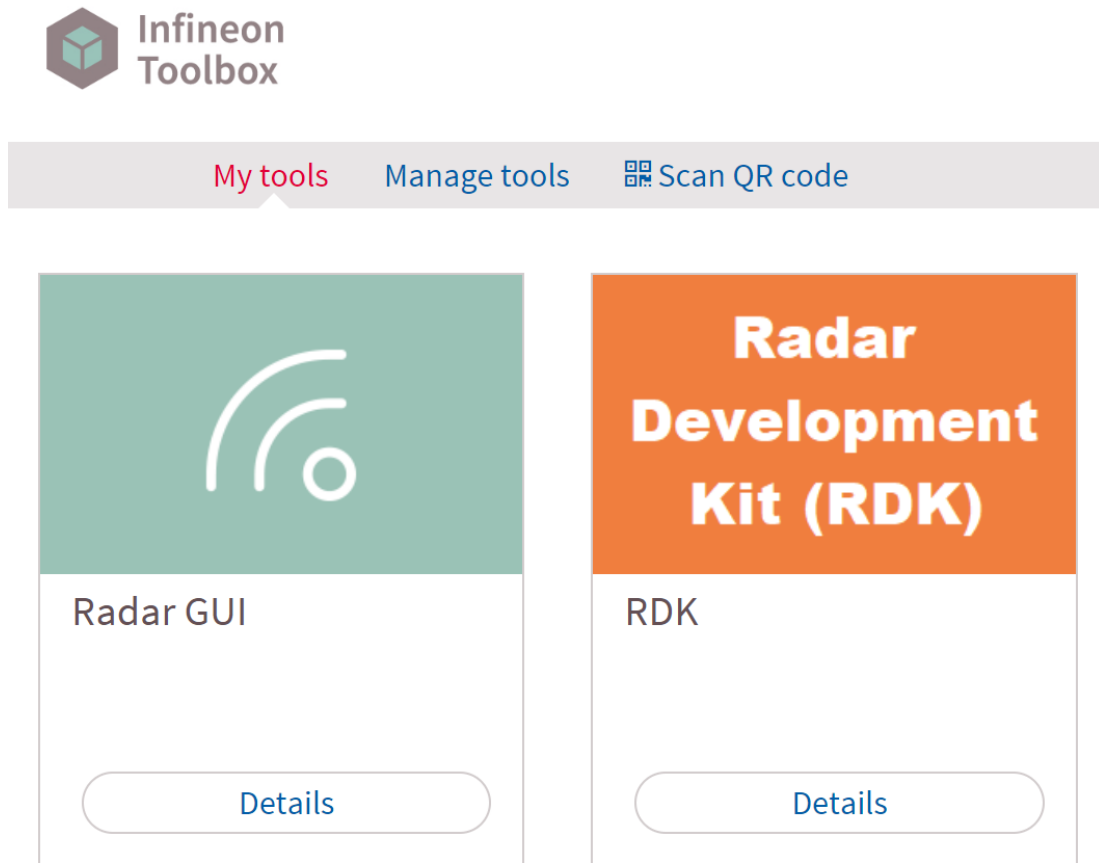
Additional update site

Load data from additional updatesites

Manage Update sites

OK

- Next, click on Manager Tools and install the RDK and Radar GUI



- These programs can then be run

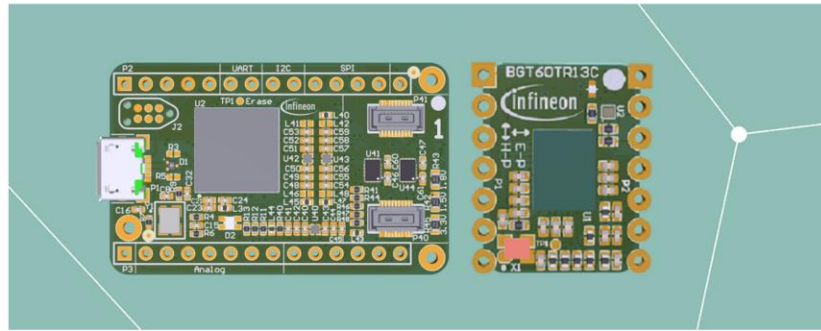
Provided are two programs called RDK and Radar GUI. Figure 13 shows the RDK program for retrieving documentation and demos from Infineon. The Radar GUI is a provided interactive GUI depicted in Figure 14 for controlling the radar, plotting range doppler, and other such information.



Radar Development Kit (RDK)

Contents

- Overview
- Software & Documentation
- Gesture Demo
- Getting started
- Information



Overview

The Radar Development Kit (RDK) offers fully developed and tested use cases as well as access to several abstraction layers via API for Infineon's 60GHz antenna-in-package radar sensor.

Summary of features

- > Presence Detection: Detects stationary and moving targets
- > Segmentation: Indicates the pre-defined segment in which presence was detected
- > GUI: Visualize use cases and several radar features such as range-doppler. Change use case specific settings which will be translated into the relevant radar settings
- > Modular and portable approach: Access the device, radar features such as the FFT or use case specific features such as the tracking layer AP
- > Multiple platforms: We offer access through C, Matlab and Python

Target applications

- > Smart speakers and audio
- > Smart Home
- > Lighting & Surveillance
- > Laptops
- > Television
- > Mobile Phones
- > Wearables

Figure 13: Radar Development Kit Program

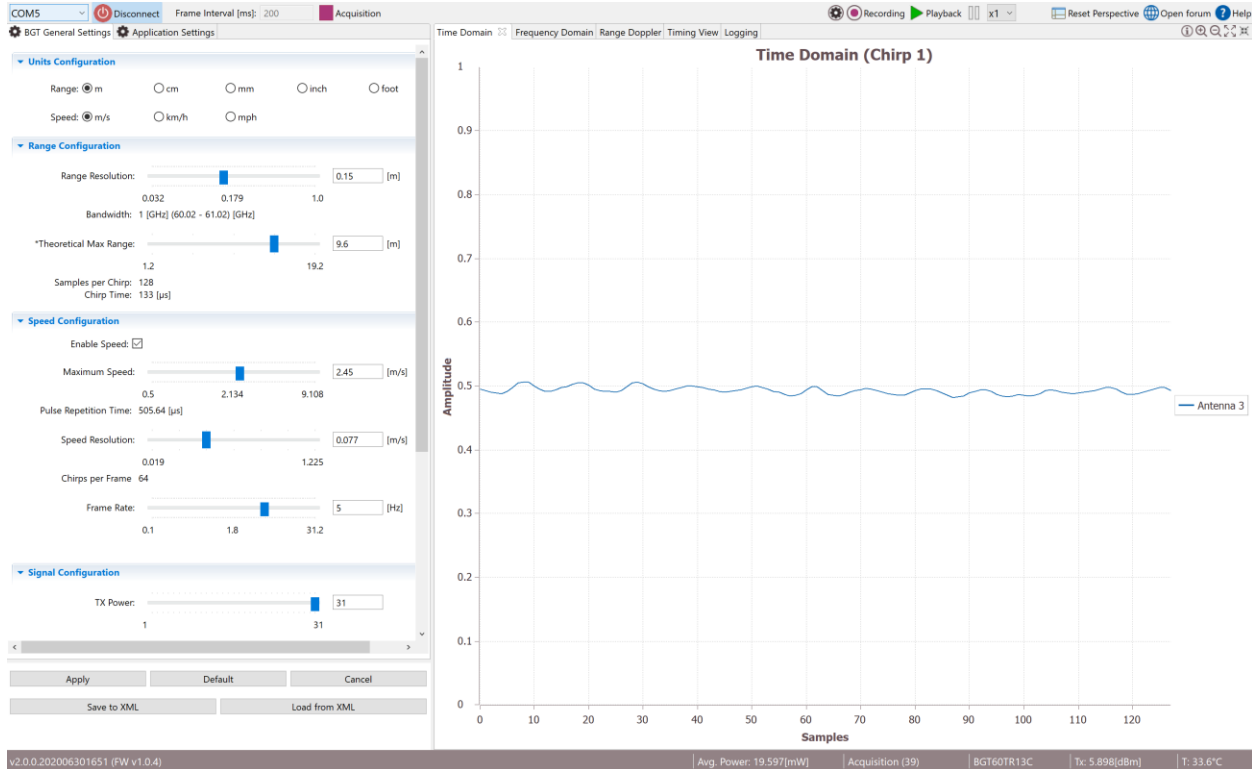


Figure 14: Radar GUI Program

Radar SDK

Provided by Infineon is a Radar SDK environment used for interfacing a USB capable host to communicate with the MCU7 baseboard. This library is written in C and can be linked to communication with the board. Documentation on the SDK can be found through the Infineon Toolbox RDK program described in section Installing Infineon mmWave Documentation.

Physical Layout

As discussed in the Introduction the system is comprised of the Infineon board, Raspberry Pi, and a server. Starting with the radar board the Infineon chipset connects to the host device using a USB connection. Contained on the Infineon board is an ARM cortex responsible for translating the SPI data coming from to the radar to stream data to a host device over the USB connection. The Radar SDK provided by Infineon allows for direct access to the streamed data as well as configuration for the radar. Figure 15 depicts the connection between the Infineon Radar board and the Raspberry Pi.

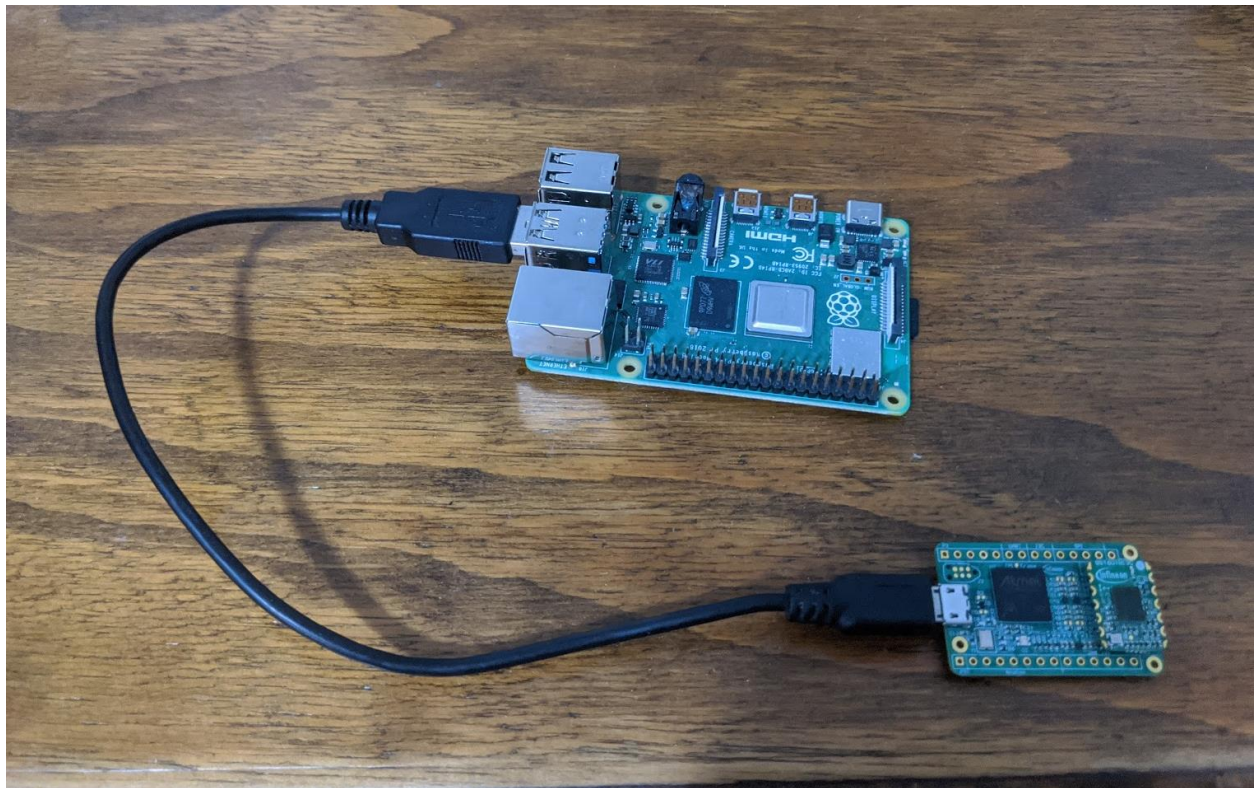


Figure 15: Infineon and Raspberry Pi Connection

The physical setup of the radar relative to the target is also important. In the Radar Parameters

While the radar is running many parameters determine characteristics such as bandwidth, chirp time (length of modulated frequency), and operating frequencies that need to be set. Table 1 depicts all the required parameters for the Infineon radar chipset.

Table 1: Radar Parameters

Parameter Name	Description
----------------	-------------

Num_samples_per_chirp	This is the number of samples acquired during each chirp of a frame. The duration of a single chirp depends on the number of samples and the sampling rate.
Num_chirps_per_frame	This is the number of chirps a single data frame consists of.
Adc_samplerate_hz	This is the sampling rate of the ADC used to acquire the samples during a chirp. The duration of a single chirp depends on the number of samples and the sampling rate.
Frame_rate_us	This is the time that elapses between the beginnings of two consecutive frames. The reciprocal of this parameter is the frame rate.
Lower_frequency_kHz	This is the start frequency of the FMCW frequency ramp.
Upper_frequency_kHz	This is the end frequency of the FMCW frequency ramp.
Bgt_tx_power	This value controls the power of the transmitted RX signal. This is an abstract value between 0 and 31 without any physical meaning. Refer to the BGT60TR13AIP datasheet to learn more about the TX power BGT60TR13AIP is capable of.
Rx_antenna_mask	In this mask, each bit represents one RX antenna of BGT60TR13AIP. If a bit is set the, RX antenna is enabled during the chirps and the signal received through that antenna is captured.
Chirp_to_chirp_time_100ps	This is the time that elapses between the beginnings of two consecutive chirps in a frame.
If_gain_db	This is the amplification factor that is applied to the IF signal coming from the RF mixer before it is fed into the ADC.
Frame_end_delay_100ps	This parameter defines the delay after each frame in 100 picosecond steps. To set this value frame_period_us must be set to 0, otherwise, this value will be ignored.
Shape_end_delay_100ps	This parameter defines the delay after each shape in 100 picosecond steps. To set this value, chirp_to_chirp_time_100ps must be set to 0, otherwise, this value will be ignored.

Range Parameters

Two range parameters are important in setting the ADC samples per chirp and bandwidth. The parameter maximum range sets the maximum range the radar can see. Range resolution is the granularity that the ranges in front of the radar are divided into. These two parameters can be used to calculate the following:

$$\text{samples per chirp} = \frac{2 * \text{maximum range}}{\text{range resolution}}$$

Due to the Nyquist frequency, only half the range FFT is evaluated requiring the range to be doubled. It is also important to note that in the Infineon chipset the samples per chirp must be a power of 2. This is due to the range FFT requiring a power of 2, otherwise the FFT would have to be zero-padded. Therefore,

$$\text{range fft size} = \text{samples per chirp}$$

Bandwidth is calculated off the range resolution of the radar. It can be calculated as followed,

$$\text{bandwidth}_{\text{Hz}} = \frac{c_0}{2 * \text{range resolution}}$$

Where c is the speed of light.

Using this information, the lower and upper frequency can also be calculated. The center frequency for the Infineon chip is set at 60.5 GHz.

$$\text{upper frequency}_{\text{Hz}} = \text{center frequency}_{\text{Hz}} + \text{bandwidth}_{\text{Hz}}$$

$$\text{lower frequency}_{\text{Hz}} = \text{center frequency}_{\text{Hz}} - \text{bandwidth}_{\text{Hz}}$$

The range per bin is also equal to the *range resolution*.

Timing Characteristics

A few timing characteristics determine the function of the radar. This is the frame rate of the radar and the chirp to chirp time. The frame rate controls how many frames happen per second. This requires that the frame time is long enough for the chirps to happen.

Chirp to chirp time depends on the center frequency of the radar and the maximum measurable velocity. This can be calculated as follow,

$$\text{chirp to chirp time}_{ps} = \frac{10^{10} * c_0}{4 * \text{center frequency}_{\text{Hz}} * \text{maximum speed}}$$

This equation is taken from Infineon with no source. As quoted from Infineon: "This formula was provided by algorithm team. Detailed information about this may be found in some papers. At this point, there is no documentation because the implementor of this function does not know those papers. Sorry".

Velocity Parameter

The maximum velocity and resolution determine the number of chirps performed per frame. This requires that the number of chirps is equal or smaller than 1024 since this is the maximum FFT size. It is also important to note that num chirps per frame must be rounded to the closest power of two. A possible rounding method is the Bit Twiddling Hack by Sean Anderson .

$$\text{num chirps per frame} = \frac{2 * \text{maximum speed}}{\text{speed resolution}}$$

Board Antenna section, the directionality of the antennas was discussed. The direct of the highest gain of the antennas is perpendicular to the PCB. This gives the optimal direction of the radars normal in the same direction as the object of interest. Figure 16 has the radar's normal setup facing the object of interest.



Figure 16: Radar Position

Once the Infineon board has been connected to the Raspberry PI the data can be streamed to a server for processing. This server for processing can be an external computer or the Raspberry PI itself. In this example, a laptop will be used for the server, but the Raspberry PI has successfully been tested as the server.

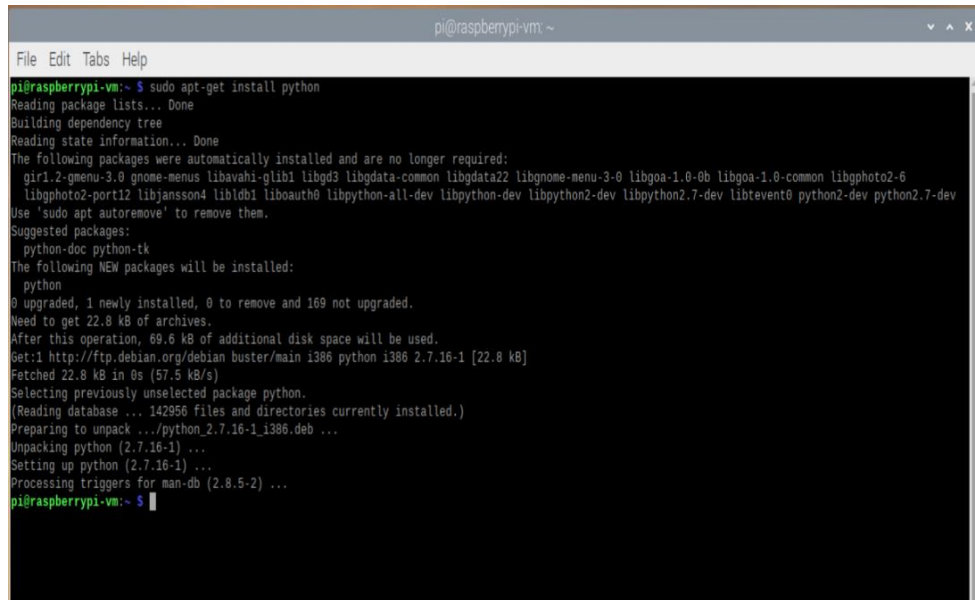
Software Setup

This section will describe how to install and run the client and server application.

Running the Server

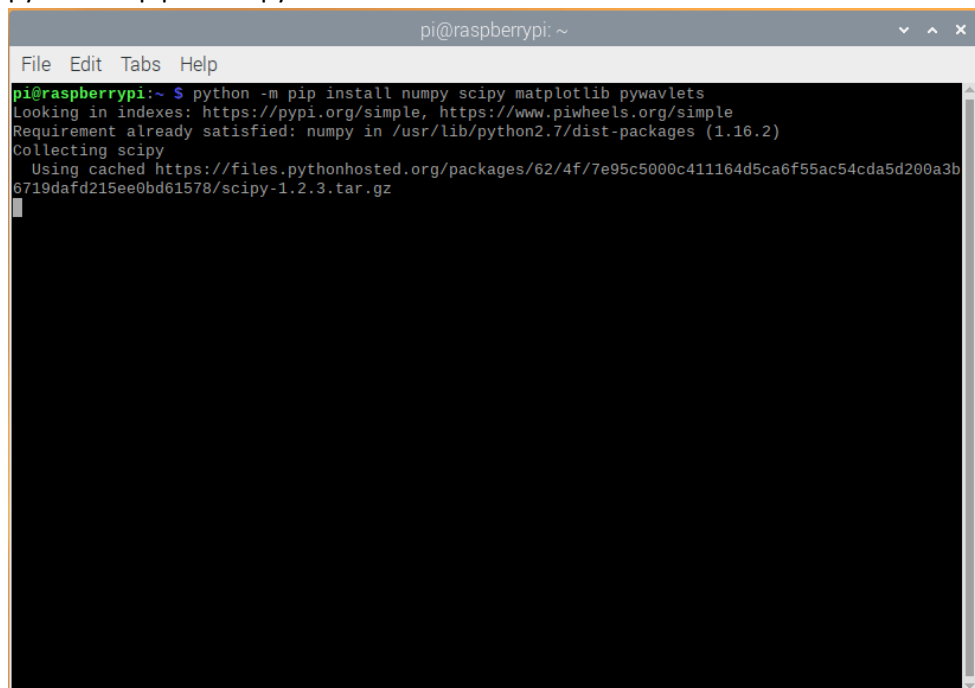
The server can be run using the command line. Some libraries are required before being able to run the system.

1. Install python
 - a. `sudo apt-get install python`



```
pi@raspberrypi-vm: ~  
File Edit Tabs Help  
pi@raspberrypi-vm:~$ sudo apt-get install python  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
The following packages were automatically installed and are no longer required:  
  gir1.2-gmenu-3.0 gnome-menus libavahi-glib1 libgd3 libgdata-common libgdata22 libgnome-menu-3.0 libgoa-1.0-0b libgoa-1.0-common libgphoto2-6  
  libgphoto2-port12 libjansson4 libldb1 liboauth0 libpython-all-dev libpython-dev libpython2-dev libpython2.7-dev libtevent0 python2-dev python2.7-dev  
Use 'sudo apt autoremove' to remove them.  
Suggested packages:  
  python-doc python-tk  
The following NEW packages will be installed:  
  python  
0 upgraded, 1 newly installed, 0 to remove and 169 not upgraded.  
Need to get 22.8 kB of archives.  
After this operation, 69.6 kB of additional disk space will be used.  
Get:1 http://ftp.debian.org/debian buster/main i386 python i386 2.7.16-1 [22.8 kB]  
Fetched 22.8 kB in 0s (57.5 kB/s)  
Selecting previously unselected package python.  
(Reading database ... 142956 files and directories currently installed.)  
Preparing to unpack .../python_2.7.16-1_i386.deb ...  
Unpacking python (2.7.16-1) ...  
Setting up python (2.7.16-1) ...  
Processing triggers for man-db (2.8.5-2) ...  
pi@raspberrypi-vm:~$
```

- b.
2. Install python libraries such as numpy, scipy, matplotlib, pywavelets
 - a. `python -m pip install numpy`
 - b. `python -m pip install scipy`
 - c. `python -m pip install matplotlib`
 - d. `python -m pip install pywavelets`

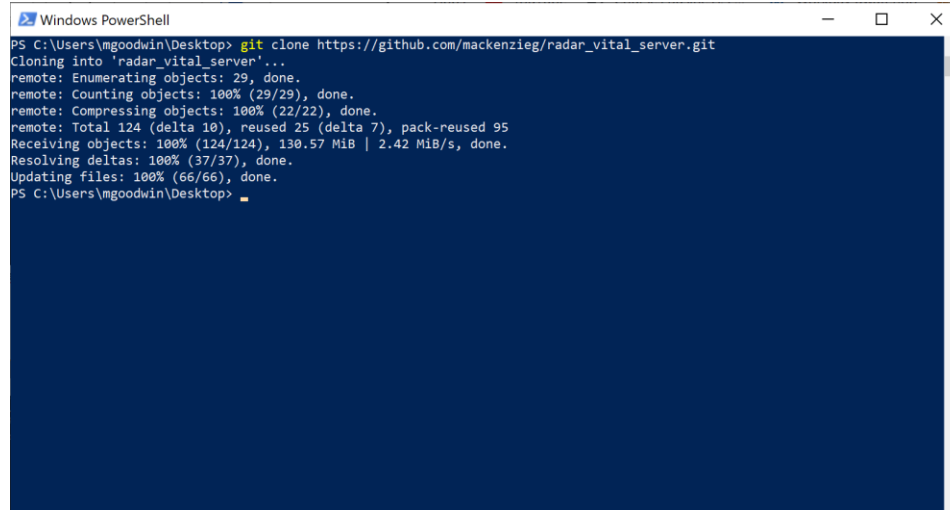


```
pi@raspberrypi: ~  
File Edit Tabs Help  
pi@raspberrypi:~$ python -m pip install numpy scipy matplotlib pywavlets  
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple  
Requirement already satisfied: numpy in /usr/lib/python2.7/dist-packages (1.16.2)  
Collecting scipy  
  Using cached https://files.pythonhosted.org/packages/62/4f/7e95c5000c411164d5ca6f55ac54cda5d200a3b6719dafd215ee0bd61578/scipy-1.2.3.tar.gz  
pi@raspberrypi:~$
```

e.

3. Clone radar_vital_server repository

- a. `git clone https://github.com/mackenzieg/radar_vital_server.git`

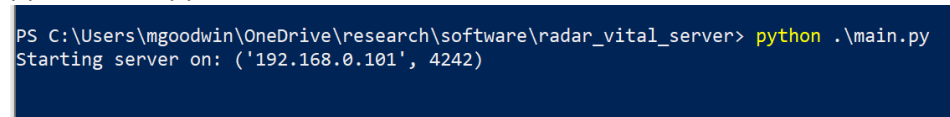


```
Windows PowerShell
PS C:\Users\mgoodwin\Desktop> git clone https://github.com/mackenzieg/radar_vital_server.git
Cloning into 'radar_vital_server'...
remote: Enumerating objects: 29, done.
remote: Counting objects: 100% (29/29), done.
remote: Compressing objects: 100% (22/22), done.
remote: Total 124 (delta 10), reused 25 (delta 7), pack-reused 95
Receiving objects: 100% (124/124), 130.57 MiB | 2.42 MiB/s, done.
Resolving deltas: 100% (37/37), done.
Updating files: 100% (66/66), done.
PS C:\Users\mgoodwin\Desktop>
```

b.

4. Run the code with the following command

- a. `python main.py`



```
PS C:\Users\mgoodwin\OneDrive\research\software\radar_vital_server> python .\main.py
Starting server on: ('192.168.0.101', 4242)
```

5. Once the server is running the client can be launched. Follow the section Running Client for information on starting the client

Running Client

Below will describe all the requirements before running the code.

1. Download CLion zip and decompress

- a. Link: <https://www.jetbrains.com/toolbox-app/download/download-thanks.html?platform=linux>

Thank you for downloading Toolbox App!

Your download should start shortly. If it doesn't, please use the [direct link](#).

Download and verify the file [SHA-256 checksum](#).

b.

2. Navigate to the directory where the tar file was placed. The tar file will then be unzipped

- a. `cd ~/Downloads`
b. `tar -xzf clion-2020.2.tar.gz`
c. `cd CLion-2020.2/bin`

3. Run Clion installation script

- a. `./clion.sh`

4. Install boost library and cmake, g++ tools

- a. sudo apt-get install cmake build-essential libboost-all-dev

```

pi@raspberrypi-vm: ~/Desktop
File Edit Tabs Help
pi@raspberrypi-vm:~/Desktop $ sudo apt-get install cmake build-essential libboost-all-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
build-essential is already the newest version (12.6).
The following packages were automatically installed and are no longer required:
  gir1.2-gmenu-3.0 gnome-menus libavahi-glib1 libgd3 libgdata-common libgdata22 libgnome-menus-3.0 libgoa-1.0-0b
  libgoa-1.0-common libgphoto2-6 libgphoto2-port12 libjansson4 libldb1 liboauth0 libpython-all-dev libvent0
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  autoconf automake autotools-dev cmake-data gfortran gfortran-8 ibverbs-providers icu-devtools libboost-atomic-dev
  libboost-atomic1.67-dev libboost-chrono-dev libboost-chrono1.67-dev libboost-container-dev libboost-container1.67-dev
  libboost-context-dev libboost-context1.67-dev libboost-context1.67-dev libboost-coroutine-dev
  libboost-coroutine1.67-dev libboost-date-time-dev libboost-date-time1.67-dev libboost-dev
  libboost-exception-dev libboost-exception1.67-dev libboost-fiber-dev libboost-fiber1.67-dev libboost-fiber1.67.0
  libboost-filesystem-dev libboost-filesystem1.67-dev libboost-graph-dev libboost-graph-parallel-dev
  libboost-graph-parallel1.67-dev libboost-graph-parallel1.67.0 libboost-graph1.67-dev libboost-graph1.67.0
  libboost-iostreams-dev libboost-iostreams1.67-dev libboost-locale-dev libboost-locale1.67-dev libboost-log-dev
  libboost-log1.67-dev libboost-log1.67.0 libboost-math-dev libboost-math1.67-dev libboost-math1.67.0 libboost-mpi-dev
  libboost-mpi-python-dev libboost-mpi-python1.67-dev libboost-mpi-python1.67.0 libboost-mpi1.67-dev libboost-mpi1.67.0
  libboost-numpy-dev libboost-numpy1.67-dev libboost-numpy1.67.0 libboost-program-options-dev
  libboost-program-options1.67-dev libboost-program-options1.67.0 libboost-python-dev libboost-python1.67-dev
  libboost-python1.67.0 libboost-random-dev libboost-random1.67-dev libboost-random1.67.0 libboost-regex-dev
  libboost-regex1.67-dev libboost-regex1.67.0 libboost-serialization-dev libboost-serialization1.67-dev
  libboost-serialization1.67.0 libboost-signals-dev libboost-signals1.67-dev libboost-signals1.67.0 libboost-stacktrace-dev
  libboost-stacktrace1.67-dev libboost-stacktrace1.67.0 libboost-system-dev libboost-system1.67-dev libboost-test-dev
  libboost-test1.67-dev libboost-test1.67.0 libboost-thread-dev libboost-thread1.67-dev libboost-timer-dev
  libboost-timer1.67-dev libboost-timer1.67.0 libboost-tools-dev libboost-type-erasure-dev libboost-type-erasure1.67-dev
  libboost-type-erasure1.67.0 libboost-wave-dev libboost-wave1.67-dev libboost-wave1.67.0 libboost1.67-dev
  libboost1.67-tools-dev libcaf-openmpi-3 libcoarrays-dev libcoarrays-openmpi-dev libfabric1 libgfortran-8-dev libhwloc-dev
  libhwloc-plugins libhwloc5 libibverbs-dev libibverbs1 libicu-dev libicu63 libltdl-dev libnl-3-dev libnl-route-3-dev
  libnuma-dev libopenmpi-dev libopenmpi3 libpmix2 libpsm-infinipath1 librdmacm1 librdash0 libsigsegv2 libtool libuv1 m4
  mpi-default-bin mpi-default-dev ocl-icd libopencl1 openmpi-bin openmpi-common python-dev
Suggested packages:
  autoconf-archive gnu-standards autoconf-doc gettext cmake-doc ninja-build gfortran-multilib gfortran-doc
  gfortran-8-multilib gfortran-8-doc libgfortran5-dbg libboost-doc graphviz libboost1.67-doc gccxml libmpfrc++-dev libntl-dev
  xsltproc docbook docbook-xml docbook-xsl default-jdk fop libhwloc-contrib-plugins icu-doc libtool-doc openmpi-doc gcj-jdk
  m4-doc opencl-icd
The following NEW packages will be installed:
  autoconf automake autotools-dev cmake cmake-data gfortran gfortran-8 ibverbs-providers icu-devtools libboost-all-dev
  libboost-atomic-dev libboost-atomic1.67-dev libboost-chrono-dev libboost-chrono1.67-dev libboost-container-dev
  libboost-container1.67-dev libboost-context-dev libboost-context1.67-dev libboost-context1.67-dev libboost-coroutine-dev
  libboost-coroutine1.67-dev libboost-date-time-dev libboost-date-time1.67-dev libboost-dev
  libboost-exception-dev libboost-exception1.67-dev libboost-fiber-dev libboost-fiber1.67-dev libboost-fiber1.67.0
  libboost-filesystem-dev libboost-filesystem1.67-dev libboost-graph-dev libboost-graph-parallel-dev
  libboost-graph-parallel1.67-dev libboost-graph-parallel1.67.0 libboost-graph1.67-dev
  libboost-graph1.67.0 libboost-iostreams-dev libboost-iostreams1.67-dev libboost-locale-dev libboost-locale1.67-dev
  libboost-log-dev libboost-log1.67-dev libboost-log1.67.0 libboost-math-dev libboost-math1.67-dev libboost-math1.67.0

```

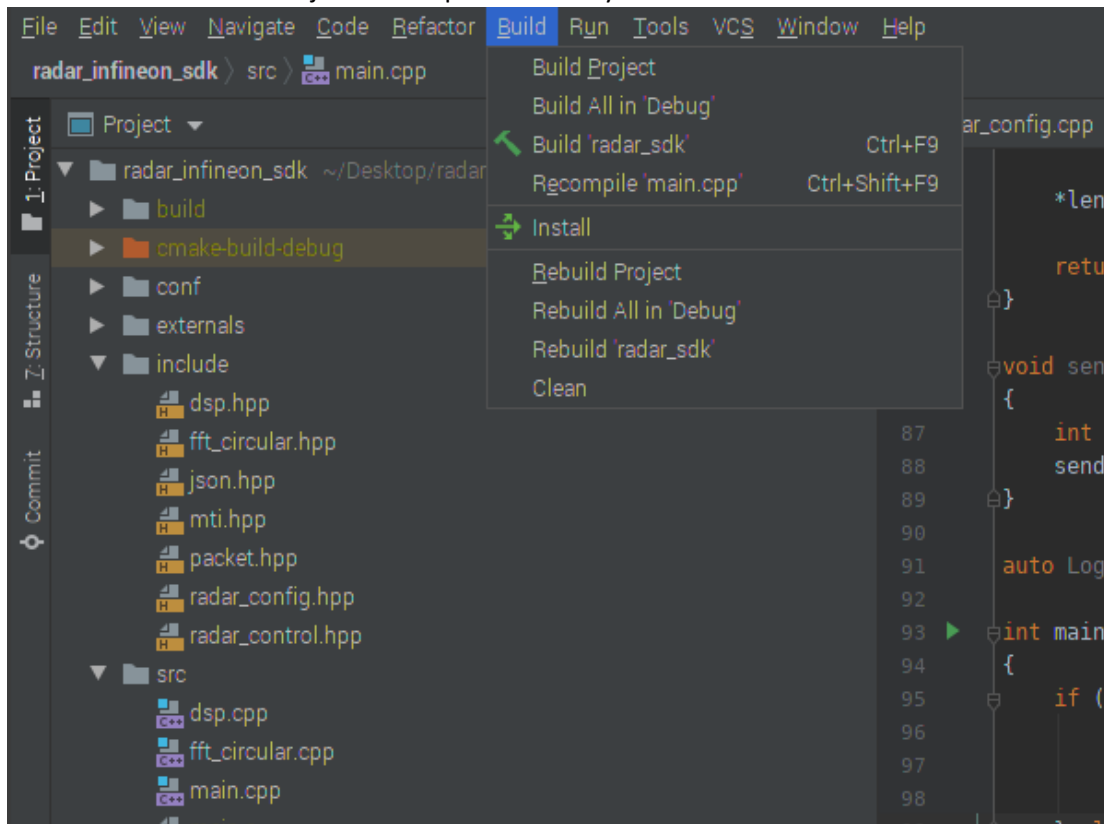
- b.

5. Clone radar_infinion_sdk repository

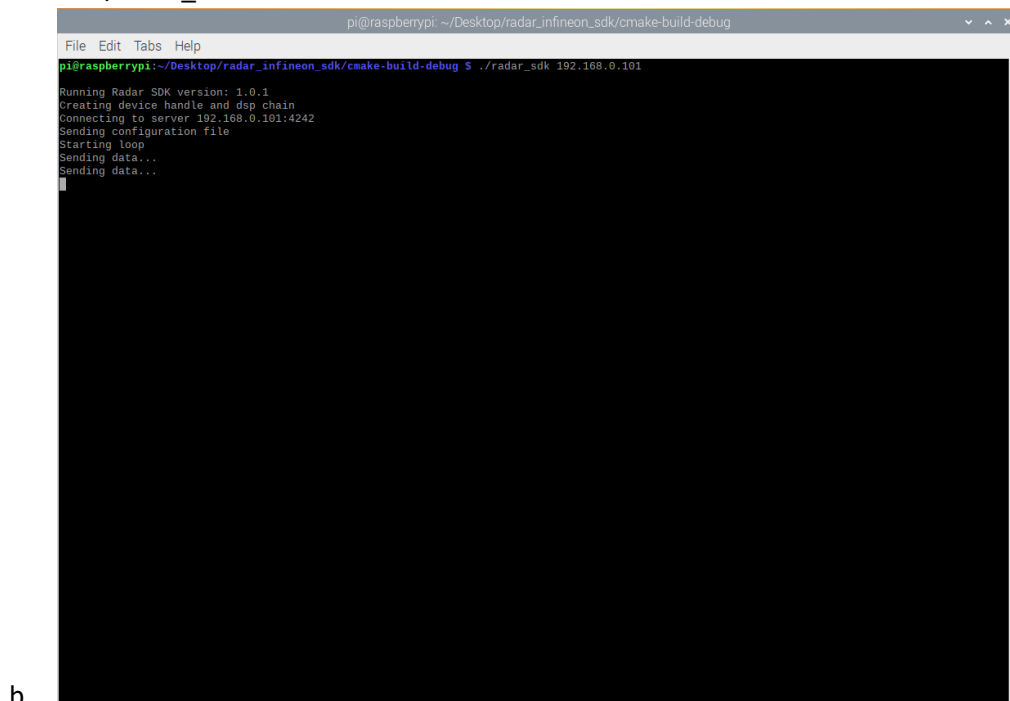
- a. git clone https://github.com/mackenzieg/radar_infineon_sdk.git

6. Open the project inside the directory of the cloned repository using CLion

- Click on Build -> Build Project to compile the binary



- Once the binary is built navigate to directory cmake-build-debug where a runnable binary named radar_sdk exists
- Run the command `./radar_sdk <ip>`
 - Ex: `./radar_sdk 192.168.0.101`



b.

It is important to note that the server must be running before the client as the client does not have any reconnect functionality. This is a future feature.

Processing

On start-up, the Raspberry PI first generates a configuration for the Infineon board. This configuration is sent over the server for the server to configure itself. Once the configuration has been sent to the server the Raspberry PI forwards the configuration to the Infineon board and initiates data streaming over the USB connection. This raw data is the data from the ADC on board and is sent over as a float format to the server. On the server-side a range FFT is performed on the raw ADC data to get the power at each range. This range FFT is then stored in a buffer to be later processed. Figure 17 describes the flow of the client and server communication.

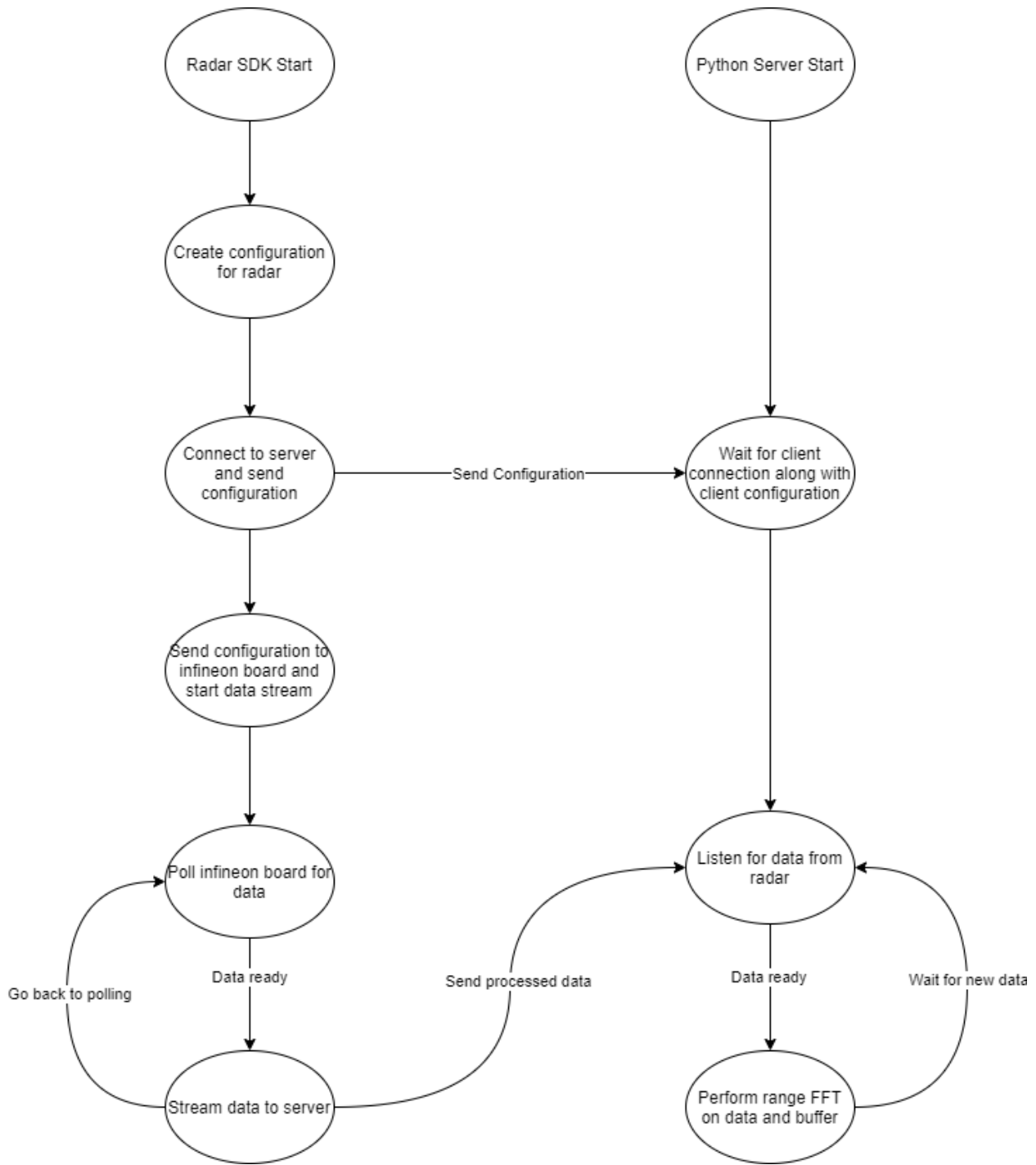


Figure 17: Client-Server Flow Chart

Once the server has filled up a buffer (20 seconds was used for testing) an autocorrelation algorithm is run on the buffer to determine the best range to perform analysis on.

Auto Correlation Background

Autocorrelation is a process that gives metrics on the correlation with itself. It works off the principle of delaying a copy of itself and taking the convolution between the two. This processing is used to determine

how periodic a signal is and can be used to determine the period of the signal. Autocorrelation is defined as:

Suppose signal $y[t]$ is a discrete signal,

$$y[t] = y[t] \quad 0 \leq t \leq N - 1 \quad (1)$$

$$r[k] = \frac{1}{N} \sum_{i=1}^{N-k} (y[i] - \bar{y})(y[i+k] - \bar{y}) \quad 0 \leq k \leq N - 1 \quad (2)$$

Where N is the length of the entire discrete time series $y[t]$.

In this equation $r[k]$ is the correlation spectrum where r is the correlation coefficient and k is the lag time. Essentially a delayed copy is convoluted with the none delayed signal and summed giving a correlation value.

It is hypothesized that a periodic signal produces a periodic response in the coefficient spectrum of the autocorrelation. The periodic response in the autocorrelation will have the same period as the signal on the input. Using the periodicity of the autocorrelation the period of the original signal can be determined. This can be proven by again looking at equation (2) but with a periodic input,

Suppose signal $y[t]$ is a periodic signal with a period of P , then

$$y[t] = y[t + P] \quad 0 \leq t \leq N - 1 \quad (3)$$

And,

$$y[t] = y[t - P] \quad 0 \leq t \leq N - 1 \quad (4)$$

Note that the dc bias of the signal is also removed before processing,

$$y[t] = y[t] - \bar{y} \quad (5)$$

Taking the normalized autocorrelation spectrum of the periodic signals (3) and (4) with k as the lag coefficient and r as the correlation coefficient,

$$r[k] = \frac{1}{N} \sum_{i=1}^{N-k} (y[i] - \bar{y})(y[i+k] - \bar{y}) \quad 0 \leq k \leq N - 1 \quad (6)$$

Since the dc bias removed in (5), \bar{y} can be assumed to be zero,

$$r[k + P] = \frac{1}{N} \sum_{i=1}^{N-(k+P)} (y[i])(y[i + (k + P)])$$

$$r[k + P] = \frac{1}{N} \sum_{i=1}^{(N-k)+P} (y[i])(y[(i + k) + P])$$

Note that since $y[t]$ is periodic then $y[(i + k) + P] = y[i + k]$ therefore,

$$r[k + P] = \frac{1}{N} \sum_{i=1}^{(N-k)+P} (y[i])(y[i + k])$$

$$r[k + P] = r[k] \tag{7}$$

And following the same proof for $r[k - P]$

$$r[k - P] = r[k] \tag{8}$$

From formula (7) and (8), the period of the autocorrelation sequence P represents the same period of the original signal $y[t]$. This will give rise to a sinusoidal response in the autocorrelation spectrum with a period P . Finding the peaks of this sinusoid can give information on the period of the original signal and the correlation.

Using python's scipy library an example of a periodic function can be autocorrelated and the coefficient spectrum graphed to see the response. In this example, a sinusoid with white noise is applied and the autocorrelation spectrum is graphed on the bottom. Figure 18 shows the periodicity in the coefficient spectrum matching the period of the input sin wave. There is also a spike in the center of the spectrum. Firstly, from (2) the coefficient spectrum is an even function meaning Figure 18 contains the mirrored spectrum on the left side of the center and the none mirror spectrum on the right side of the center. This means everything to the left of the center is negative lag coefficients.

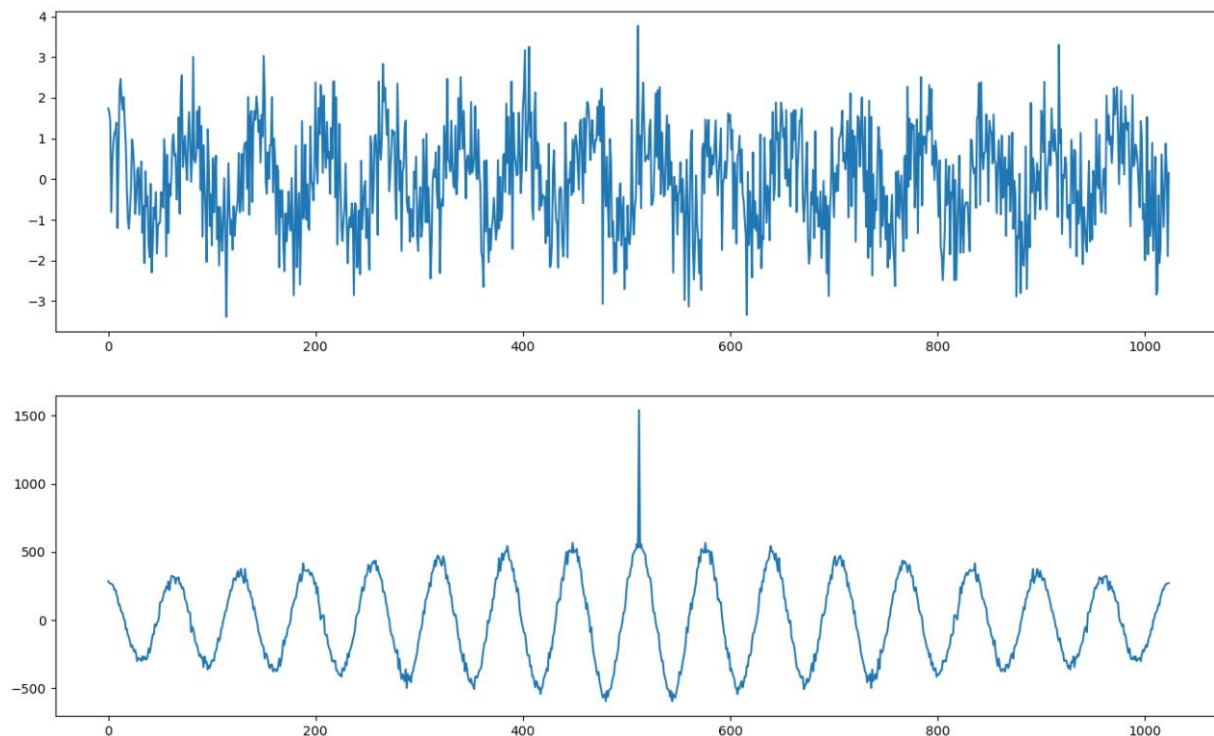


Figure 18: Sin Wave with Noise Correlation

This mirrored characteristic of the spectrum can be ignored for now. This mirrored portion can be removed to get the proper coefficient spectrum starting with zero lag.

Since the spectrum is symmetrical, we can normalize the spectrum to get a correlation value between 0 and 1 along with removing the symmetric half of the signal. Figure 19 shows the mirrored part removed. Notice the spike that exists at a lag of zero. This phenomenon is the average power of the signal and can be derived from (2)

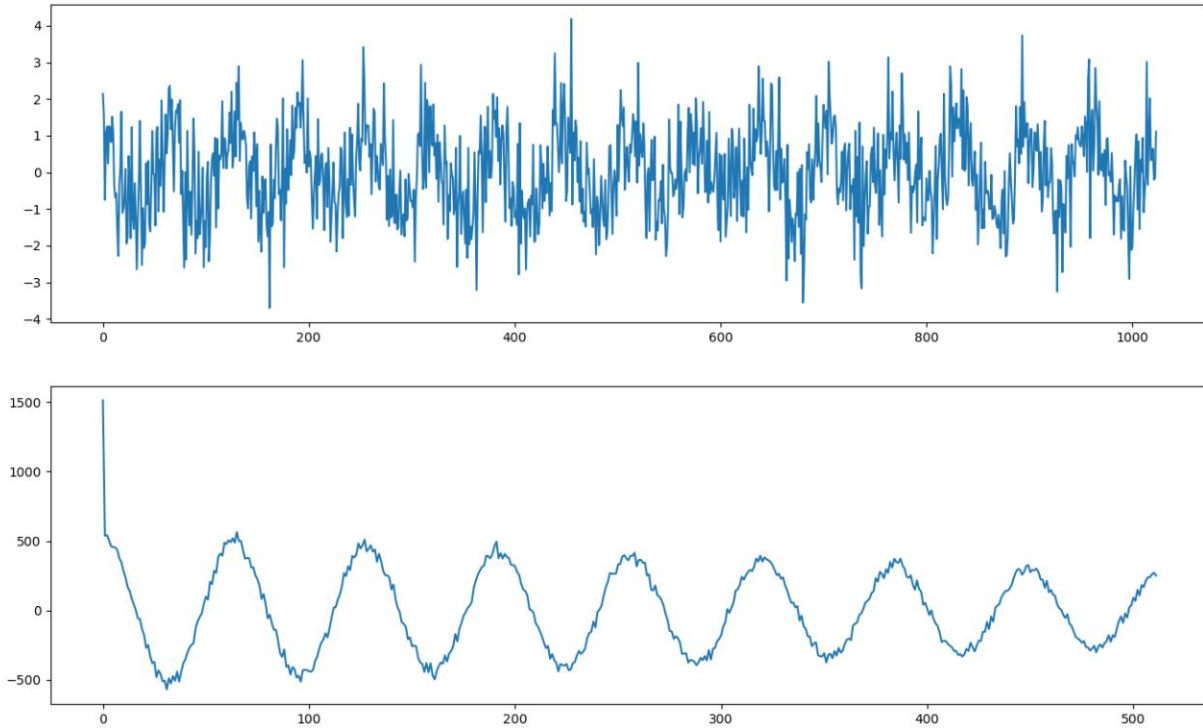


Figure 19: Autocorrelation Symmetry Removed

$$\begin{aligned}
 r[0] &= \frac{1}{N} \sum_{i=1}^{N-0} (y[i])(y[i + 0]) \\
 r[0] &= \frac{1}{N} \sum_{i=1}^N (y[i])^2 = P_y
 \end{aligned}
 \tag{9}$$

Therefore equation (9) shows that a lag of zero corresponds to the average power of the signal. This is due to the average dc value not being removed from the input signal.

The autocorrelation spectrum is also not normalized yet. After normalization, the correct correlation can be determined. Figure 20 is the spectrum normalized. The first harmonic exists at a lag value of 64 which is the period of the original input signal. The first harmonic also has a peak at ~ 0.4 which can easily be detected using a peak searching algorithm.

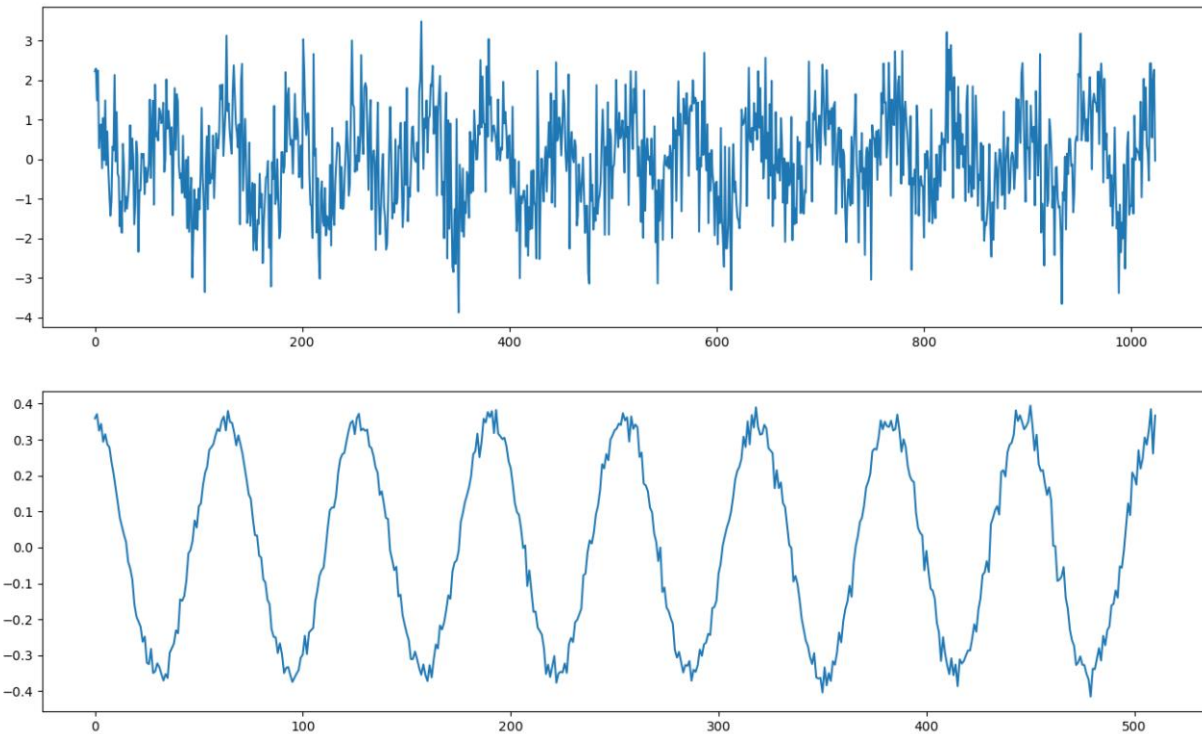


Figure 20: Autocorrelation Normalized

It is also important to note that the correlation factor changes with different SNR values. In Figure 20 the SNR is 0dB which is well below what a normal use case would be. With an SNR of 6dB, the above example has a correlation of 0.7.

The autocorrelation algorithm with peak detection is presented as a method of testing if a range bin contains

Auto Correlation Implementation

A series of N samples are performed within a single chirp. Using a range FFT these N samples are turned into N samples of spectral power at each specific range. These N range powers are then buffered into a matrix of $N \times M$ samples where M is the buffer time. With a chirp rate of 32 chirps per second the size of M is picked to be 1024. This gives a buffer time of 32 seconds. Analyses can then be performed on this matrix.

Since autocorrelation is a slow process, minimizing the amount of analyzed signal is required. To start the process, an estimated best range bin is performed using a computational minimal algorithm, then the predicted is refined using autocorrelation. This minimal algorithm for the initial prediction can be simple. In this example, an algorithm that takes the highest peak to peak value was used. This predicted range bin will be referred to as x .

Using this predicted range bin (referred do as index x) a better range bin is extracted using an autocorrelation method. First, a range of interest is done around the range bin x . This range to search is adjustable with a range of $x \pm 4$ being used. Range bins $x - 4$ to $x + 4$ is iterated over with an index n . The

real part of the time series n is taken which has a length of M samples. The DC bias of the series is removed and then an autocorrelation is performed on the time series. This gives a spectrum of coefficients for the autocorrelation which is symmetrical about the $M/2$ index and the mirrored portion is removed. If the time series n is largely noise, the expected autocorrelation response is a large peak at lag time 0 with the rest being a flat response. To determine if bin n is a bin containing purely noise a linear regression is performed to determine the linearity of the response. If the r -value of the linear regression is above 0.5 range bin n is ignored and the next range bin the range of interest is evaluated. Figure 21 is an example of a radar range bin with effectively only noise with autocorrelation which is highly linear.

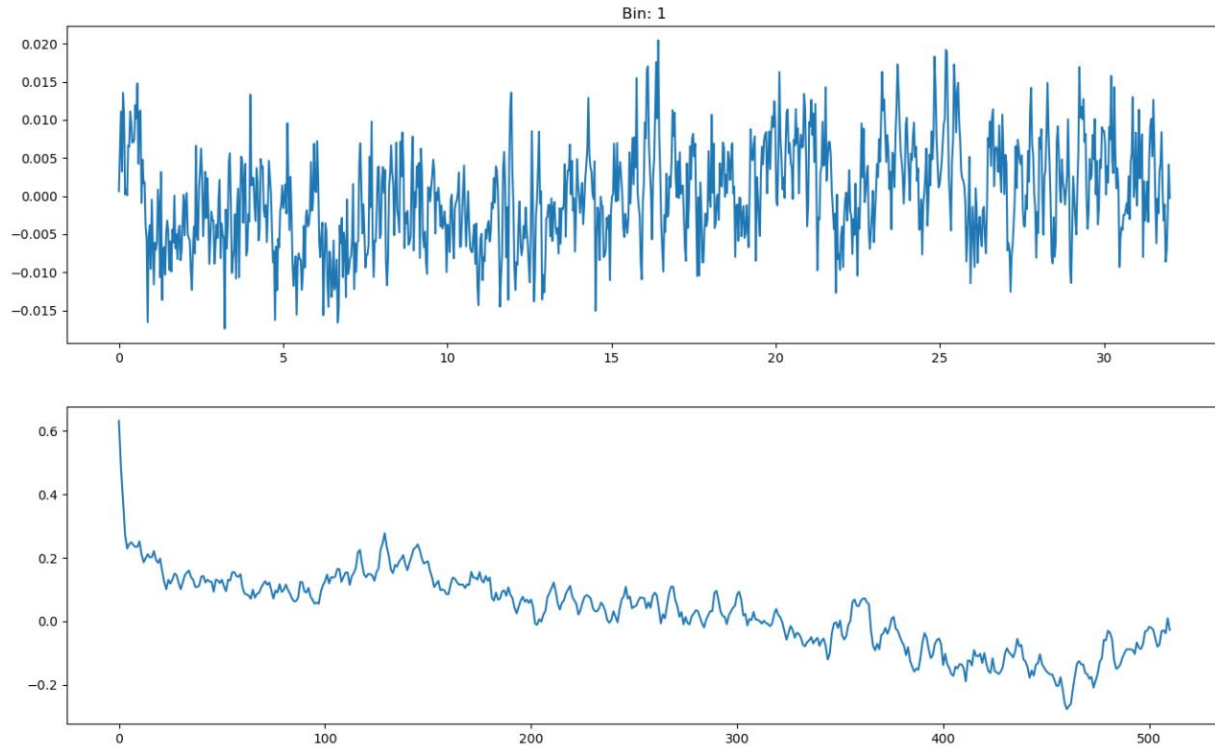


Figure 21: Noise Auto Correlation r_value 0.8

After linear regression is performed and highly linear signals are eliminated, peak detection can be performed. To perform peak detection some data preparation is performed. The autocorrelation spectrum is mirrored by the $M/2$ index with the zero-lag coefficient removed. Mirroring the signal helps with the peak detection finding all the peaks.

Since the correlation spectrum is mirrored the central peak is the lag 1 peak. Anything following it on the right are the harmonics. In Figure 22, the first harmonic peak exists at 164 lag and r value 0.7. With a lag time of 164 and samples per second of 32, the period is calculated to be 5.1 seconds. In most cases, the lag time of the first harmonic can be used to determine the frequency of the oscillations. This assumption falls apart with larger buffer times as the oscillation frequency can change over time causing a lower correlation.

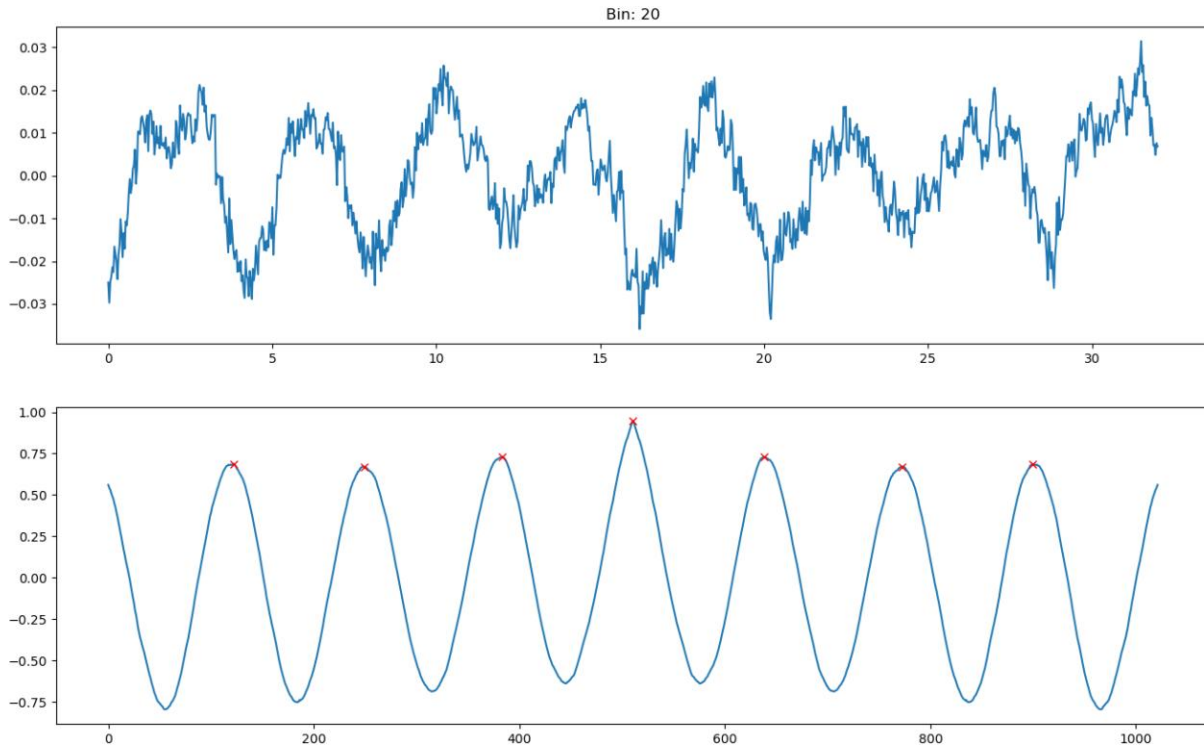


Figure 22: Auto Correlated Peak Detection

The described algorithm has been summarized in a simple flow graph. Figure 23 depicts this summarized flow graph.

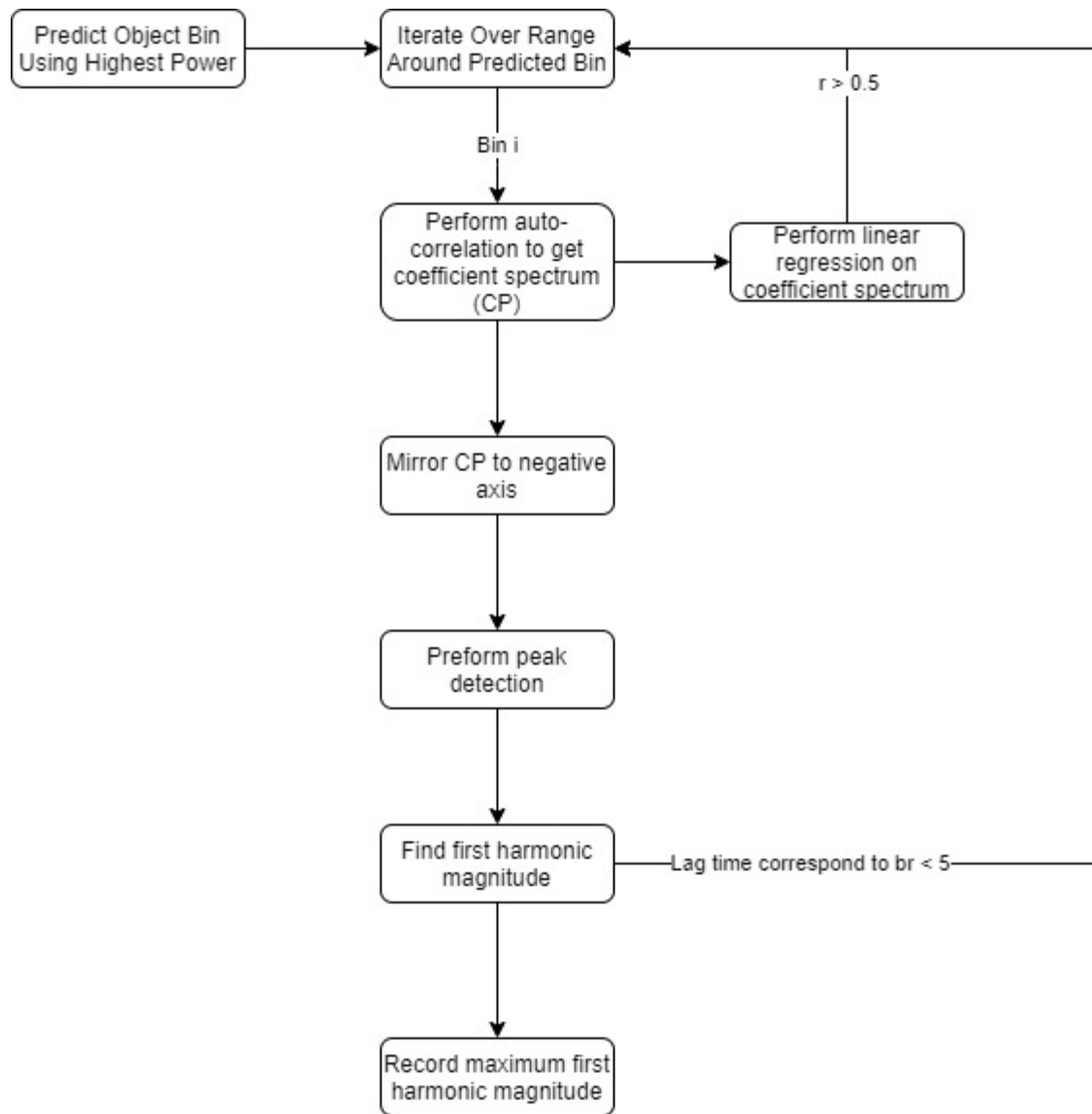


Figure 23: Auto Correlation Flow Graph

Auto Correlation Filtering

Due to real-world noise in the system, some preprocessing is required before running the autocorrelation method. This helps eliminate mispredictions. Before the specific range row is processed a Butterworth low pass filter can be run on the data. Since higher frequency data does not want to be lost to help with seeing the difference in correlation between range bins, a higher cut-off of 4Hz was picked from testing. This filter could be replaced with a few different methods such as a median filter or singular value decomposition.

It is also noted that looking at a coefficient spectrum from an autocorrelation if there is simple respiration with no other activities it is expected that there is a very sinusoidal response. When there is a higher number of other harmonics it causes ripples in the coefficient spectrum. This is reduced by again applying a Butterworth low pass filter set to 3Hz applied to the coefficient spectrum. This can be optionally applied in the signal processing chain if required.

Some parameters can be tuned to change the behavior of the autocorrelation described below.

Delta Bin

As described in Auto Correlation Implementation, an estimated prediction is performed first and the autocorrelation algorithm iterates over that range. The delta bin parameter determines how large the range is. The highest bin that is checked is the estimate bin plus the delta bin and the lowest is the estimate bin minus the delta bin. The final number of bins that are checked are $\text{delta_bin} * 2 + 1$.

This parameter is a function of how large the object is and the range resolution of the radar. With a larger object, the number of bins must be larger. For a smaller range resolution, the object will also take up more bins requiring more of the bins to be checked.

Linear Correlation Threshold

As described in Auto Correlation Implementation, highly linear range bins are eliminated by performing linear regression and checking the r-value. If the r-value is above a certain threshold, the signal is considered too linear and is ignored. In current testing, a threshold of 0.5 is sufficient to eliminate most highly linear signals.

Max-Min RPM

Maximum and minimum rpm is used to adjust the peak detection and reject some of the detected peaks. The maximum breath rate is converted to samples per period which sets the minimum peak width for peak detection. The minimum rpm is used to reject first harmonic peaks in the autocorrelation spectrum that corresponds to too slow of an rpm.

Test Setup

For testing purposes, an automated breathing doll was used. This doll used a plastic pushing plate that would oscillate to mimic the breathing of a person. The oscillation speed can be adjusted to represent different breathing rates. The autocorrelation algorithm could then be used to extract the breathing rate from the radar data and evaluated relative to the doll's breathing rate.



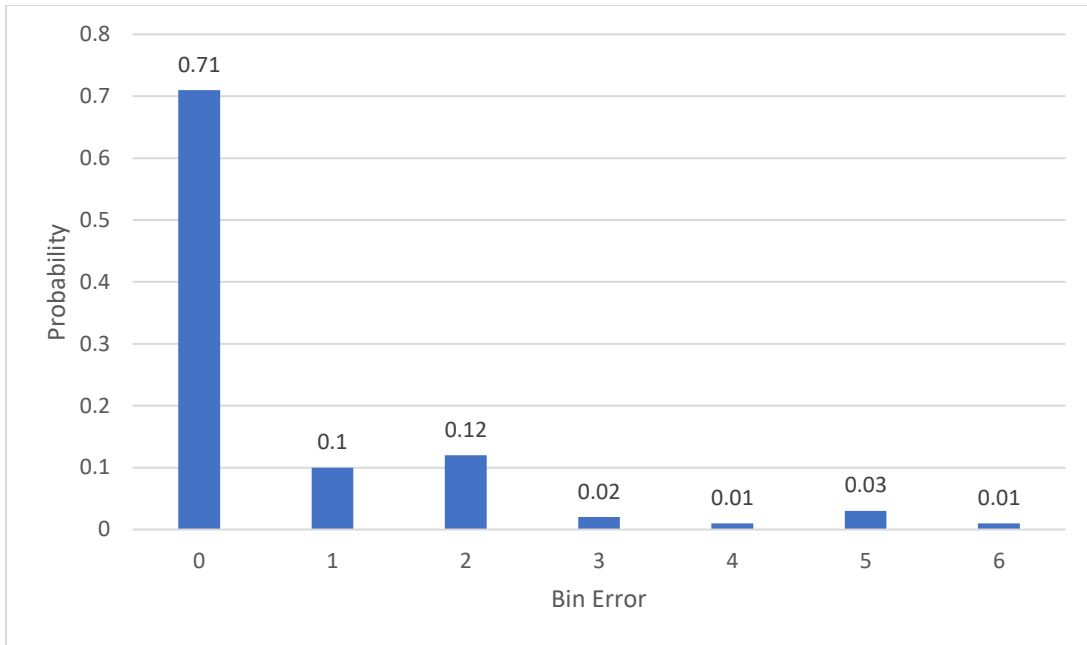
Results

The autocorrelation algorithm was tested using an automated doll with a calibrated breathing rate. This allowed for a reference object for the algorithm to detect a breathing rate. Included in the testing were trials with the doll and person breathing in front of the radar. For access to the full set of demos please contact <email>.

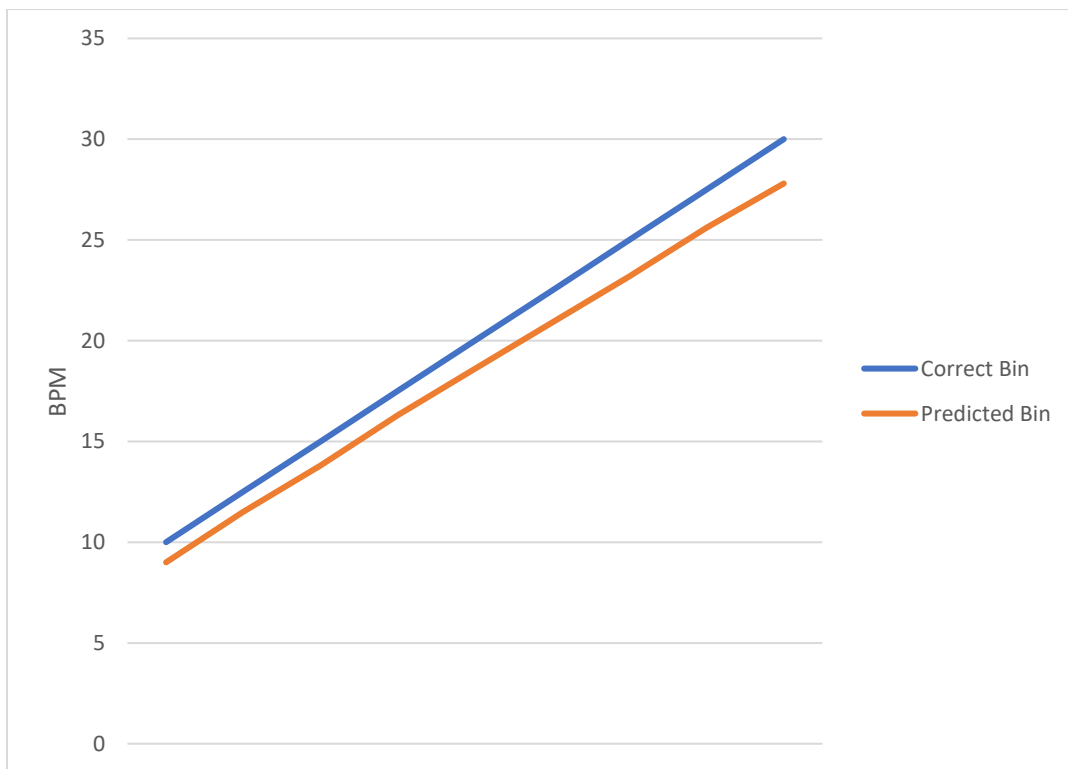
Provided in this document are some of the results from the doll breathing in front of the radar. An image of the test setup is attached below depicting the doll situated in front of the Infineon radar system.



The accuracy of the bin predictions was tested by using a breathing rate of 15bpm with different distances from the radar. The predicted bin was then collected over 5 minutes every with a predicted made every second. The correct bin was manually selected by looking at a heat map of the distance FFT over time. The predicted bin was then compared to the manually selected bin to determine its accuracy. This data was then compiled to give each data point a bin error or distance from the correct bin. Below is a graph of the data.



The ability of the algorithm to determine the correct BPM was then tested by keeping the doll at a constant distance and varying the breathing rate of the doll. The doll was tested at different breathing rates ranging from 10BPM to 30BPM. The predicted breathing rate was then compared to the breathing rate set by the doll to give an accuracy. Below is a graph depicting the predicted vs correct BPM.



Conclusion

This report explains the design of a breath detection algorithm using a radar system. The system consists of three major steps, first the Infineon Radar board sends the raw ADC data to the Raspberry Pi. Second, the Raspberry Pi streams the data over TCP to a server. Finally, the server does all the processing of the signal including range FFT and autocorrelation. The server also outputs the detected range and breathing rate with graphical representation.

The autocorrelation algorithm is shown to be effective for finding the periodic nature of breathing. Taking the autocorrelation with some filtering and extra processing the correlation of the signal can be determined and the range with the highest correlation can be used to detect breathing rate. The breathing rate can also be extracted using the correlation from the lag coefficient. The peak detection can be improved to make the system more reliable.

The system was tested with a doll and a person breathing. In both cases, the autocorrelation algorithm was able to determine the correct range and can determine the correct breathing rate. The maximum amount of delay from the start of breathing to the first detection can be adjusted to give a trade-off between accuracy and speed.

Bibliography

- [1] Infineon, "Infineon Radar Systems," [Online]. Available: <https://www.infineon.com/cms/en/product/sensor/radar-image-sensors/radar-sensors/>.
- [2] Y.-J. Park, "A method of detection of respiration rate on Android using UWB Impulse Radar," *ScienceDirect*, 2016.
- [3] "Radar Wikipedia," [Online]. Available: <https://en.wikipedia.org/wiki/Radar>.
- [4] S. E. Anderson, "Bit Twiddling Hacks," [Online]. Available: <http://graphics.stanford.edu/~seander/bithacks.html#RoundUpPowerOf2>.
- [5] Infineon, in *IFX_AN600_BGT60TR13C Shield - V2.1*.
- [6] "Phase-comparison monopulse," [Online]. Available: https://en.wikipedia.org/wiki/Phase-comparison_monopulse.
- [7] Infineon, in *IFX_AN599 - Radar Baseboard MCU7 - V2.2*.
- [8] Infineon, in *IFX_BGT60TR13C Preliminary Datasheet V2.3.14*.