GOA COLLEGE OF ENGINEERING
FARMAGUDI, GOA
DEPARTMENT OF ELECTRONICS & TELECOMMUNICATION ENGINEERING

2020 - 2021

DESIGN AND DEVELOPMENT OF RADIO FREQUENCY CLASSIFIER AND JAMMER

by

PRITESH ALIAS VAIBHAV V. NAIK (P.R.No.: 201705004)
KUNAL K.GUDE (P.R.No.: 201704580)
ADITYA D. S. GAITONDE (P.R.No.: 201704577)

A project submitted
in partial fulfilment of the requirements
for the degree of
Bachelor of Engineering
in
Electronics & Telecommunication Engineering
GOA UNIVERSITY

under the guidance of

Prof. NITESH B. GUINDE
Associate Professor,
Electronics & Telecommunication Department
Goa College of Engineering

CERTIFICATE

This is to certify that the project entitled

"DESIGN AND DEVELOPMENT OF RADIO FREQUENCY CLASSIFIER AND JAMMER "

submitted by

PRITESH ALIAS VAIBHAV V. NAIK	**P.R. No.:201705004**
KUNAL K. GUDE	**P.R. No.:201704580**
ADITYA D. S. GAITONDE	**P.R. No.:201704577**

has been successfully completed in the academic year 2020-2021 as a partial fulfilment of the requirement for the degree of BACHELOR OF ENGINEERING in Electronics & Telecommunication Department, at Goa College of Engineering, Farmagudi.

Internal Examiner
(Prof. Nitesh B. Guinde)

External Examiner

Head of Department,
Dr. H. G. Virani,
Professor, ETC Dept.

Place: Farmagudi, Ponda, Goa
Date:

PROJECT APPROVAL SHEET

The project entitled

"DESIGN AND DEVELOPMENT OF RADIO FREQUENCY CLASSIFIER AND JAMMER"

by

PRITESH ALIAS VAIBHAV V. NAIK	**P.R. No.:201705004**
KUNAL K. GUDE	**P.R. No.:201704580**
ADITYA D. S. GAITONDE	**P.R. No.:201704577**

completed in the year 2020-2021 is approved as a partial fulfilment of the requirements for the degree of **BACHELOR OF ENGINEERING in Electronics & Telecommunication Engineering** and is a record of bonafide work carried out successfully under our guidance.

_____

Prof. Nitesh B. Guinde,
Associate Professor,
Electronics and Telecommunication Dept.

_____

Head of Department
Dr. H. G. Virani
Professor, ETC Dept.

_____

Principal
Dr. Rajesh Basant Lohani
Goa College of Engineering

Place: Farmagudi, Ponda, Goa
Date:

Declaration

I/We declare that the project work entitled "DESIGN AND DEVELOPMENT OF RADIO FREQUENCY CLASSIFIER AND JAMMER " submitted to Goa College of Engineering, in partial fulfillment of the requirement for the award of the degree of B.E. in Electronics and Telecommunication Engineering is a record of bonafide project work carried out by me/us under the guidance of Prof. Nitesh B. Guinde. I/We further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Signature of candidate/s
Name of the Candidate

Date

Acknowledgement

We are profoundly graceful to Prof. Nitesh B. Guinde and Prof. Shahjahan Kutty for their expert guidance and continuous encouragement throughout to ensure that this project achieves its target from commencement to completion.

We would also like to thank Dr. Prabhu Chandhar of Chandhar Labs for his help in setting up the necessary equipment and his guidance throughout the project.

We would like to express deepest appreciation towards Dr. H. G. Virani, HOD, Department of Electronics and Telecommunication and , Principal, Goa College of Engineering.

Lastly we must express our sincere heartfelt gratitude to all the staff members of Electronics and Telecommunication Department who helped us directly or indirectly during this course of this work.

Contents

List of Figures

Abstract

In recent times, there have been a lot of advancements in remotely controlled aerial vehicles technology. With the applications of Unmanned Aerial Vehicles or UAVs ranging from aerial photography to last-mile delivery options, they are becoming popular with the public. Lately, they have been used for activities such as illegal surveillance and terrorism. With an ever-increasing threat of unauthorized UAVs being deployed in high-security regions, consistent efforts have been made by the authorities to set up regulations on UAVs. But regulations are rarely enough to keep such activities in check.

The solution would be to create a system that can easily be deployed on a large scale to ward off unauthorised activities. Modern-day UAV detection systems either use RADAR, acoustic or visual identification technology to detect UAVs. But these systems are usually very convoluted, way too expensive or ginormous to be able to deploy in mass.

To counter these issues, we have developed a system that makes use of radio frequency signature detection methods to observe and possibly shut down such illegal activities. This involves the use of SDR for capturing RF signals and machine learning for the classification of the captured signals. The output of such a system can be used to then jam only the UAV signal without interfering with other signals. Subsequently, this system can also be integrated into an automatic anti-UAV system that can either disable or even take control of the UAVs.

Chapter 1

Introduction

1.1 Preamble

An Unmanned Aerial Vehicle (UAV) or Uncrewed Aerial Vehicle, commonly known as a drone is an aircraft without a human pilot on board. UAVs are a component of an unmanned aircraft system (UAS), which include a UAV, a ground-based controller, and a system of communications between the two. The flight of UAVs may operate with various degrees of autonomy: either under remote control by a human operator or autonomously by on-board computers referred to as an autopilot.

Compared to crewed aircraft, UAVs were originally used for missions which were too dull or dangerous for human physical intervention. While drones originated mostly in military applications, their use is rapidly finding many more applications

including aerial photography, product deliveries, agriculture, policing and surveillance, infrastructure inspections, science, and drone racing.

The Major Drone Manufacturers of nowadays are:

- DJI (Dajiang) Innovations: Their products include DJI Mavic and DJI Phantom series, both of which are mainly used for cinematography purposes due to their high reliability and HD video capabilities .
- Parrot SA: Their Products Include ANAFI Series, Parrot AR Series. Both of which can be deployed in military use cases as well as infrastructure inspections due to their higher range and HD video capabilities.

Recently, the usage of Unmanned Aerial Vehicles Drones has increased exponentially. Drones have been popular with the enthusiast community for more than 10 years. Introduction of hardware such as Ardupilot and FlySky controllers has brought down the complexity of making Do-it-Yourself drones down to a manageable standard. Also, the prices of consumer drones have gone down over the years. As such they have gotten cheaper and more ubiquitous.

As such these drones have a high chance of getting in hands of nefarious elements who can use them to invade personal privacy, conduct surveillance on sensitive installations such as military and commercial establishments and prove a hazard near airports. In the past Drones have been used by Naxal forces and terrorists for Surveillance on the Indian Army and CRPF troops. Recently there have also

been cases where drones have been used to deliver fatal payloads on the Indian army.

With the introduction of drone delivery as last mile option by many e-commerce companies, drones are going to be more widespread. Hence the problems caused by drones are going to be more common. Looking at such cases detection and identification of drones has to be our priority.

1.2 Motivation

Right now Drone identification in general is done by 4 methods.

- Drone identification using RADAR: Drone identification using RADAR is one of the most common methods deployed by the military. It uses a Doppler or FMCW RADAR to detect drones. The advantage of this method is that it can track multiple drones simultaneously even during the night and heavy fog conditions. But the disadvantage of this method is that it requires the transmission license for civilian usage and checks to ensure that the spectrum used by the RADAR is empty.

- Visual identification: this is done by using thermal or normal cameras and image processing to detect drones. This method is fairly cheap and passive as it does not require any special equipment but is fairly ineffective when at night or in foggy conditions.

- Acoustic identification: In this method, we use the acoustic signature to detect drones. This method can detect drones at a short distance and when LOS detection is not always possible due to obstructions, etc. But it cannot detect drones unless they are under 300 m, as the acoustic signature of drones is very faint beyond this limit.

- Identification using RF signature: This method uses the RF signature of the drone to identify the drones. Each family of drones has its own unique RF control signals and modulation technique that constitutes its unique RF signature. This method can detect multiple drones and detect which family the drone belongs to, but it cannot detect drones which do not transmit any data back to the controller.

Our project tries to find a fairly cheap solution for finding the presence of UAVs by finding if the RF signature of a particular drone is present or not. This involves us learning about different Software Defined Radios - RTLSDR, ADALM-PLUTO among others and how to operate them via GQRX Software, GNU Radio and their Python APIs. It also involves us learning digital signal processing in Python and machine learning for identifying whether a given RF signature is of drone or something else like Wi-Fi or Bluetooth.

1.3 Outline

Over the past few years, drones have been more common than before. So the issues with them have become more important than before. Right now they are available at an accessible price. Such drones have also been used in surveillance and attacks on individuals and the military. So in this situation, it is highly necessary to detect drones, and to find if a given drone is friendly or not.

According to The SDR Forum and Institute of Electrical and Electronics Engineers (IEEE) P1900.1 group, Software Defined Radio(SDR) is defined as "Radio in which some or all of the physical layer functions are software defined". Using a SDR allows us a high level of freedom as compared to traditional radio equipment. We can quickly change filters, modulation schemes and even do some kind of signal processing that were not possible to implement in traditional radios systems. It also allows us to quickly change target frequency and bandwidth in software as long as it is supported by the hardware.

In our case, the SDR we decided to go with, ADALM-PLUTO, is a SDR evaluation board based on AD9363 and it has one RX and one TX channel. It is able to measure and generate signals from 325 to 3800 MHz with a 20MHz bandwidth. It is able to do all this over a standard USB 3 connection which allows us to plug it in any computer and not worry about using up any ethernet port. It plugs in as a USB ethernet dongle and is easy to pass through to virtual machines to do any work. We also experimented with RTLSDR, a SDR which is able to only receive

signals, but at a lower carrier frequency and has a stable bandwidth of 2.4 Mhz.

Since the 2.4 GHz band is unlicensed, everything from microwave and cordless phones to Bluetooth and Wi-Fi works on this band. Since this band is unlicensed, commercial and hobbyist drones also use this band to transmit and receive control and video signals which makes it very hard to capture isolated drone signals. We captured the signals in isolated fields, where Bluetooth and Wi-Fi signals could be considered absent.

After signal capturing, we need to process the signals to try and remove noise signals from the captured signals and try to isolate transmitted drone signals. In our case, we did this by first converting the signals from time domain to frequency domain by using Fast Fourier Transform and using signal strength as a parameter and cutting off signals that were below a certain threshold.

After signal processing, we trained a neural network on these signals to make the computer identify these signals. This network will later take a signal occupying a particular spectrum and try and to identify whether it is a drone signal or non-drone signal, which will be a base for jamming that particular spectrum occupied by the signal.

Chapter 2

Literature Survey

2.1 Introduction

An extensive literature survey is done on Drone Frequency arrange and their Power Levels to Construct a detection logic and use these methods of detection based on the requirement along with Drone Regulations in India

This paper [1] gives us a broader understanding of the process of signal capturing and dataset creation Using RF Signature of drone Signals. In the paper they have generated a dataset of 3 drone manufacturers(AR Drone,Phantom Drone and Bebop Drone)

This whitepaper [2] contains the policy roadmap which gives a better idea regarding the rules and regulation imposed by the Indian government under Civil

Aviation Regulations (CAR) for UAS for commercial and non-commercial drone users. It also provided us with a better understanding of the rules and regulations with their respective constraints on the UAS and the Pilot. It further shows the importance of training for the pilot in flying the drone

This website [3] provides the instructions for initial setup and configuration of PlutoSDR in a virtual machine and introduction to digital signal processing in python

[4] gave us a broad idea about the types of ambient signals present in the Wi-fi frequency range. It also provided us with a better understanding of the Wi-Fi signal.

[5] helped us to differentiate between and categorize the drone, Wi-Fi and Bluetooth signals using their RF Fingerprints. It also gave an idea of how to implement machine learning for signal classification using different methods.

Chapter 3

Project Objectives

3.1 Project Objectives

Based on the above Chapters and information, our objectives are as follows:

(a) To design and implement a capture and detection system for manufactured and DIY drones from their frequency signatures using Software Defined Radios like PlutoSDR, NI-SDR etc.

(b) To design and implement a machine-learning algorithm to analyse the captured data and classify the signals as drone and non-drone signals.

(c) To use the output of the machine learning algorithm to find the frequency of operation of the drone and send out a jamming signal to hinder its operations.

3.2 Project Methodology

3.2.1 Data Creation and Preparation

Data creation and preparation involve the capture of the signals and converting them into usable datasets which can be later used by the machine learning algorithm. This entails the following processes :

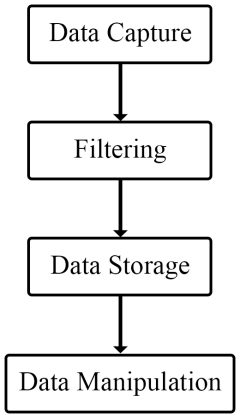


Figure 3.1: Data creation pipeline

1. Data Capture

   - Data capture is the process of collecting data that will be processed and later used to fulfil certain purposes.
   - In our case, we shall make use of a Software Defined Radio to Capture the frequency spectrum for further analysis. This process involves setting up the capture device on the platform used for storing the data.

This is done by installing the required libraries and API's, calibrating the receiver and detecting the signal intended to be captured.

- This requires intensive research about the frequencies that are used for drone communication. Finding an optimum range to be captured to avoid the capture of the unnecessary spectrum and unwanted signals is extremely necessary. This is done to reduce the size of the dataset thus saving processing time.

2. Filtering

- Filtering is the process that, completely or partially, suppresses unwanted components or features from a signal. This means removing some frequencies to suppress the interfering signals and to reduce the Background/ambient noise. In our case, even filtering the natural resonance of the software-defined radio is required to get a clean and usable spectrum scan.
- This can be done by using multiple methods. We plan to use a thresholding function on the captured signals to get rid of the unwanted low power signals which are usually ambient noise and disturbances.
- This also helps to get rid of empty datasets which purely contain noise. Thus helping to create more concise and useful datasets. After filtering, the captured data is sent to the next process that is data storage.

3. Data storage

- Data storage is the process of storing the captured and filtered data in the required format which in our case is in the form of CSV files

(comma separated values) using the Pandas library for ease of access and storage.

- The stored data is then sent to the Next step in the process that is the data manipulation process.

4. Data Manipulation

- Data manipulation refers to the process of adjusting data to make it organised and easier to read. This might involve adding, deleting, and modifying the database. This is one of the main stages to get the data prepared for use by the model.
- In our case, data manipulation is done to make it easier for the machine learning model to process the data and make it useful for model training and testing. This is done by combining multiple CSV files generated during the previous stages and manipulating the data within to reduce the size of the files and organise them as per the requirements of the model.
- Training labels are also assigned during this stage. In machine learning, labelling is the process of identifying raw data and adding one or more meaningful and informative labels. This is done to provide context so that a machine learning model can learn from it. In our case, labels are used to indicate the difference between a drone and non-drone signals such as Wi-Fi and Bluetooth signals.
- The datasets obtained from this process are then split into Training

sets and Evaluation sets which are then used for the machine learning process.

3.2.2 Machine learning model and Training

Once the data is in usable shape and format, the next step would be to form a machine learning model and to train it using the data prepared earlier by applying a range of techniques and algorithms and check its operation and accuracy and get a feasible output.

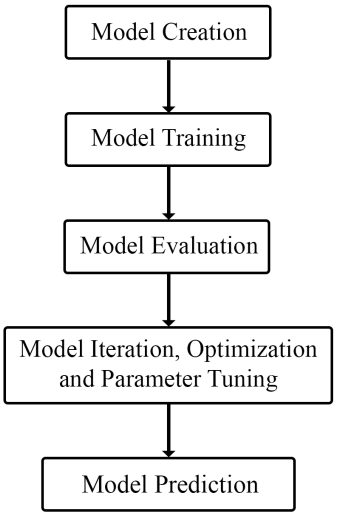


Figure 3.2: Machine learning pipeline

This phase includes model technique selection and application, setting model parameters and adjustments, algorithm selection, Classification, model training, model evaluation, development and testing, model validation, and optimization of the model.

1. Model creation

   - Model creation includes selecting the right algorithm based on the objectives and the data available, Configuring , tuning the hyperparameters for optimal performance and determining a method of iteration. This is done to attain the best hyperparameters and identifying the features that could provide the best results.
   - In our case, we will be using supervised learning to build the model which works on the basis of classification.

2. Model Training

   - Here the goal is to make sure that correct predictions are made as often as possible. Training a model simply means determining appropriate weights and the bias from labelled data using the training sets generated earlier.
   - In supervised learning, a machine learning algorithm builds a model by examining many examples and attempting to find a model that minimizes loss. Loss is the penalty for a bad prediction. That is, the loss is a number indicating how bad the model's prediction was on a single example. If the model's prediction is perfect, the loss is zero, otherwise, the loss is greater. The goal of training a model is to find a set of weights and biases that have a low loss.

3. Model evaluation

- Model evaluation includes the evaluation of the performance of a machine learning model, which is an integral step to the whole process. The model evaluation aims to estimate the general accuracy of the model of future or unseen data. This is done using validation or evaluation data sets which are used to train the model with the goal of finding and optimizing the best model to solve the problem at hand and further tune the hyperparameters for optimal performance.
- Model evaluation can be considered the quality assurance of machine learning. Adequately evaluating the models performance against metrics and requirements determining how the model will work in the real world.

4. Model iteration, optimization and parameter tuning

   - Based on the results of the evaluation process, adjustments are made to the various hyperparameters such as the number of training steps, learning rate, initialization of the weights
   - This is done for multiple iterations or epochs so as to fine-tune the parameters and optimize the values of said parameters for improved performance and accuracy.

5. Prediction using the Model

   - Once the results of the model are satisfactory and the accuracy is acceptable, the model can be implemented in the real world for the sole purpose it was designed for and can start making predictions and

get practicals outputs based on all the training, optimization and work put into the model.

Chapter 4

Design

4.1 Equipment & Software used

4.1.1 Software

1. Oracle Virtualbox

   Oracle Virtualbox is a cross platform type 2 hypervisor. It runs across windows, mac and linux and allows us to install and use guest operating systems inside virtual machines. Using a virtual machine helps in management of dependencies and creates reproducible environments which can help while prototyping. We used Virtualbox for setting up dependencies and capturing data with the help of PlutoSDR.

2. Python

   Python is a programming language with very powerful and easy to use libraries. It has a large set of libraries for machine learning, signal processing, among others. Since it is popular with academic and student communities, it has a large set of APIs for interfacing with various SDR.

3. Conda

   Conda is a open source package and environments management system for Python. It maintains it's own set of libraries and has preconfigured environments for machine learning libraries such as TensorFlow. It also makes it easy to create virtual environments and use multiple versions of Python without dependency issues. We used a minimal version of conda called miniconda to install Python 3.9 and manage dependencies for PlutoSDR and TensorFlow.

4. TensorFlow

   TensorFlow is a open source platform for machine learning. It has a large support for creating and deploying various types of neural networks along with easy to use interface though Keras. We used it for training our neural networks to distinguish between drone and non-drone signal.

5. NumPy

   NumPy is the standard library for array related computation for Python. NumPy makes it easy to handle arrays in Python through it's vast list of modules and functions. It is relatively fast as it relies on C/C++ under

the hood for the computation. It also has some handy functions for Fourier Transforms inside the fft module.

6. Matplotlib

Matplotlib is the standard library for plotting and visualization of data in Python. It makes it easy to plot, add legend and annotations and export plots for various purposes. We used it primarily to plot the sampled data and PSD for the same. We also used it to visualize and interpret many of the results.

4.1.2 Hardware

1. PlutoSDR

ADALM-PLUTO is a Software Defined Radio learning module developed by Analog Devices. It is based on Analog Devices AD9363 and Xilinx® Zynq Z-7010 FPGA. It can scan from 325 MHz to 3.8 GHz which makes it suitable for scanning the ISM band at 2.4 GHz. It can also transmit at the same frequencies with a maximum bandwidth of 20 MHz. It connects over a standard USB3 connection as a USB ethernet controller which makes it easy to pass through to virtual machines. It also has a well documented Python library which makes it easier to use it in projects.

2. RTLSDR

RTLSDR is another software Defined Radio based off of DVB-T TV tuner

dongles based on the RTL2832U chipset. It is a very cheap SDR which can be used to scan RF spectrum. It's is capable of only receiving signals and not transmitting them. It can usually scan between 50 MHz to 1000 MHz with some models even able to scan till 2200 MHz. It can also use a upconverter for scanning higher bands. We primarily used it for experimenting with spectrum scanning and plotting.

4.2 Design

4.2.1 Creation of the Dataset

Data Capture

During this stage, we undergo the process of capturing the RF energy signatures of drones or any other RF devices in the vicinity using a RF receiver.

This stage can be divided into three sections:

1. Set up
2. Receiver Calibration
3. Signal Detection

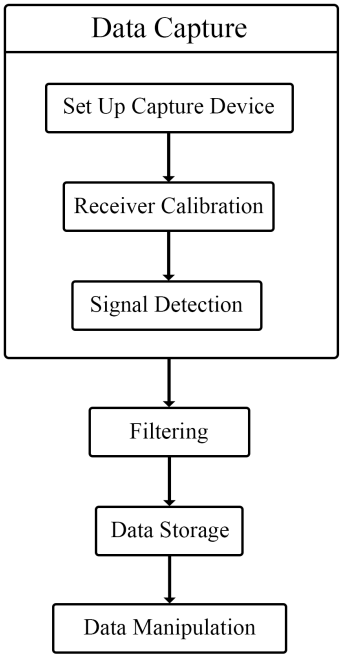


Figure 4.1: Data creation pipeline

1. Set up

   - To begin with the data capture process, we require to install Python 3.9 as given in Appendix-0.1. Then import the appropriate libraries and APIs for PlutoSDR, Matplotlib libraries, and NumPy libraries in Python after following appropriate steps to install TensorFlow via Conda as shown in Appendix-0.1.

2. Receiver calibration

   - We require to import the appropriate Python libraries and APIs for PlutoSDR to initialize its functional parameters such as scan bandwidth,

sampling rate, RF bandwidth, and sample capturing functionalities.

- During this Stage, we ensure to calibrate the Receiver parameters properly to produce the required signal outputs for the training and evaluation stages.

3. Signal Detection

- Using PlutoSDR as a receiver, a time frame of 5 sec is taken for a scan range of 2.405-2.715 GHz with a scan bandwidth of 2.5MHz. The recorded points are the FFT energy readings for corresponding frequency components in the time frame of 5 seconds and are stored in tuple data type. This is achieved using the Python libraries of PlutoSDR, math(filter design).
- To facilitate filtering action ahead, these tuples are then converted to NumPy array to allow data manipulation.

Filtering

- During dataset creation to remove noise and unwanted signals due to various natural reasons (ambient noise & natural resonance of SDR etc), a threshold of 15,000 in energy level is set. This helps in lowering dataset size by removing noise signals and replacing them with 0. ie. any signal greater than 15,000 in energy level are recorded as they are, while others are substituted with '0' in the training dataset while storing them.

- This concept is further utilized to ensure that valid scan time signals are recorded and ideal no-signal scanning values are discarded to ensure proper dataset creation.

- Following this, the recorded signals are manipulated appropriately and passed to the next process block.

Data Storage

- During dataset storage, the recorded signals present in the NumPy array are converted into a Pandas DataFrame using the Pandas library in Python for easy access and storage.

- During the storage process, we have to ensure to include the parameter 'index=False' in the 'store.to_csv' command to remove the storage of the array indices in the resultant CSV files.

- These files are further utilized either for training or evaluation, depending on the process cycle of the project.

Data Manipulation

- This stage is only applicable in the training dataset preparation process to create a training dataset. We have recorded multiple signals and stored them independently resulting in a large set of CSV files. Multiple CSV files are generated by multiple capture runs and fragmentation of the frequency

spectrum due to limited capture range while capturing a wide bandwidth.

- In this stage, multiple CSV files are combined to form a larger file with an additional data row which provides training labels that are utilized in the machine learning training process. In this case, training label '0' is for the non-drone signals while '1' is for the drone signals.

- A singular dataset file is generated containing different ratios of drone and non-drone signals. This is done by appending multiple random drone and non-drone signals with their training labels and shuffling them by using numpy.random.shuffle function.

4.2.2 Training the model

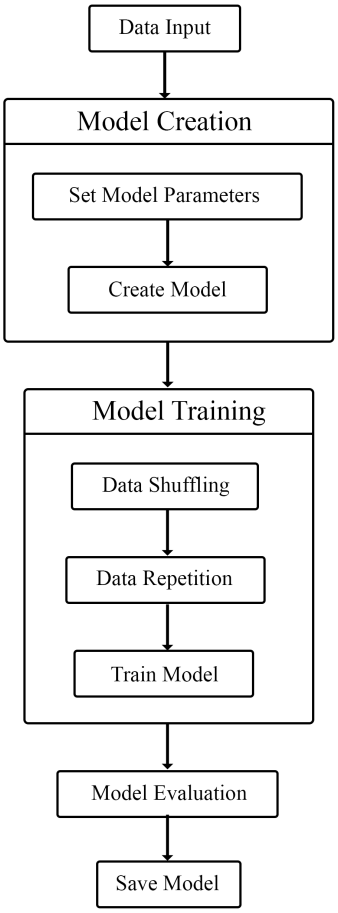


Figure 4.2: Model Creation Pipeline

1.Data Input

- To begin with the training process, we load the stored training dataset stored in .csv format into the system memory in the form of a Pandas DataFrame.
- For training, the DataFrame is divided into two parts namely 'train_labels' and 'train_features'. Where the test_labels are used for training validation

and test_features refer to the input data on which the model is to be trained on.

- train_features undergo a normalization process to ensure proper learning parameters are extracted from the input data.
- Further, these two parts are then joined appropriately forming a tensor dataset named 'train_dataset' which is later given to the training model.

2.Model creation

During this stage, we undergo the processes involved in the creation of a model which is to be trained later on.

This stage can be divided into two sections:

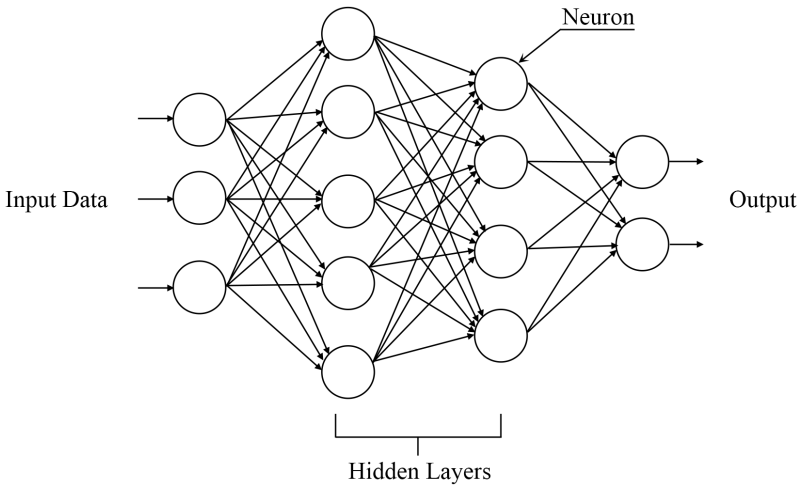
1. Model Parameters
2. Creating the model

- Model Parameters
  - During this stage, the model parameters such as the number of layers, types of layers used, type of activation functions associated with the model layers are declared and initialized.
  - In our application, we have made use of sequential model available under keras.

- Model Creation

  - In model creation, it includes declaration and initialization of the model parameters shown in the prior sections and it also involves the definition and declaration of other model parameters such as its loss function, metrics, and optimizer function

  - The later stated parameters are used during the training process to change the layer weights and strengths associated with hidden layer nodes of the model as and when required for proper training and learning gradient calculation, while metrics are provided for analysis of the model by the user.



3. Model Training

During this stage, we undergo the processes involved in training a model using the concept of **supervised learning algorithms.**

This stage can be divided into three sections:

1. Data Shuffling
2. Data Repetition
3. Training the Model

- Data Shuffling
  - In data shuffling, the prepared dataset is shuffled internally along with their corresponding labels to prevent the model from memorizing the values and give a false sign of higher accuracy, resulting in improper model learning.
  - This is achieved by using batch-wise shuffling as defined by the 'batch_size' and 'SHUFFLE_BUFFER_SIZE' parameters in the shuffle function used on the dataset.
  - This process is repeated multiple times during the training process as the shuffle function is called at the end of each epoch of the training cycle.
- Data Repetition
  - In data repetition, the dataset is made to repeat itself multiple times as defined by its function parameters.
  - This is very useful at times when you wish to test the model with a small dataset size to verify the training process.

- Train Model
  - In the training phase, the prepared dataset is applied to the created model in the previous sections for training.
  - The training process is carried out by the applied optimizer and loss functions to regulate the learning rate of the model, simultaneously metrics parameters are extracted for model analysis by the user.
  - Once the model is trained, using the metric parameters (accuracy) and loss function values retrieved during the training process are used to plot the accuracy v/s epoch and loss v/s epoch plots for graphical analysis of the trained model
  - Model training is achieved by the use of the command 'model.fit()' along with its attributes such as 'epoch=15', target input, and target_label.

4. Model Evaluation - Getting Testing Dataset

- In model evaluation phrase, a small portion of unused training dataset is used to evaluate the response of the model to unknown data input.
- This phrase also provides to give a test pilot run of the model for its actual implementation output response
- Model evaluation is achieved by the use of the command 'model.evaluate()' along with its attributes such as test_input and target_label.

5. Model Saving

- In model saving, the created and trained model is saved in their individual folder name which is given as its parameter.
- This is achieved by the use of the function 'model.save()'. The weights of the trained model, if needed, can also be stored separately in the form of pickle files. The stored weights can later be used to create or re-train another neural network.

4.2.3 Classifier Implementation

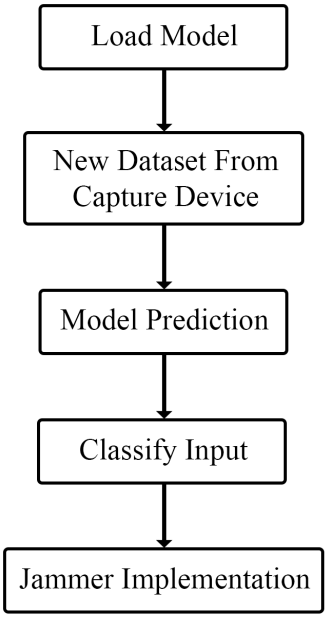


Figure 4.3: Classifier Implementation

Model Loading

- Once the entire model is trained, tested and saved in the file system, it needs to be loaded again by a detection code to use the model. This allows for a faster process as the model need not be trained multiple times as the detection process is carried out.
- Once the model is loaded, it can perform all the operations a model is able to do after training i.e weight calculation, model evaluation, model prediction and model re-training if needed.
- This is achieved by using the function 'models.load_model()' under the Keras or tensorflow.keras library depending on the version of tensorflow used. Given that the parameter passed to the function is the folder name of the intended model that is to be loaded by the code.

Testing new Dataset from PlutoSDR

- DataFrame captured by the SDR, that are stored in '.csv' format, are loaded in the code by the use of function 'read_csv()' under the Pandas Library in Python.
- Once the intended DataFrame is loaded, the necessary input signal values are extracted from the DataFrame and stored in the predict_features variable for the prediction process.
- Following the data extraction process, the prediction process begins with

appropriate data type and input size arrangements in order to ensure proper prediction.

Model Prediction

- During the prediction process, the model with its trained weights and algorithm of prediction, tries to predict what output category the given input comes under.
- Depending on its classification criterion, it predicts the output category and gives the appropriate probability matrix as its output containing the probability of the input falling under a certain category trained by the model to classify into.

Input Classification

- In the classification phrase, the output probability matrix of the model is taken and aggregation is performed on the matrix giving a single value as its output.
- This singular integer value corresponds to the category predicted by the model. Thus providing the make of the target drone to the jammer system implementation.

Jamming

- Once the signal is classified into its corresponding make, the code jumps to the corresponding jammer code for that drone maker.
- Since the input signal is of a fixed length and in a predefined frequency range, depending on the presence of signal in the captured dataset, the required frequency band and channel can be identified
- By using the above two inferences, a suitable and appropriate jamming signal can be transmitted by the jamming transmitter in order to jam/immobilise the target drone communication channels

Chapter 5

Results

- After doing successful set-up and receiver-calibration procedures, 20*100 readings of length 12280 points of drone and non-drone signals respectively were successfully captured and stored into the file system for training of the model.
- some captured sample drone and non-drone signals are plotted below.

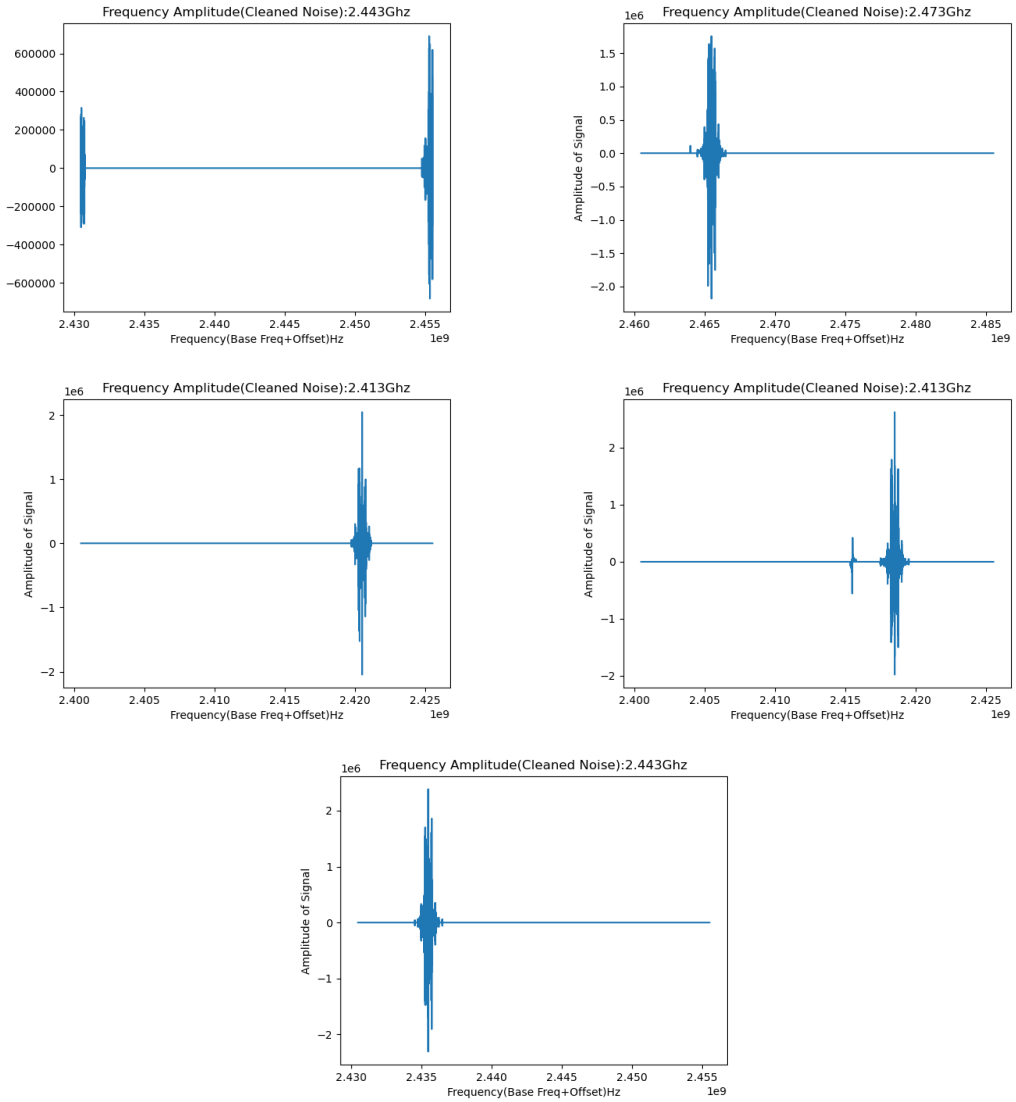


Figure 5.1: Sample drone signals

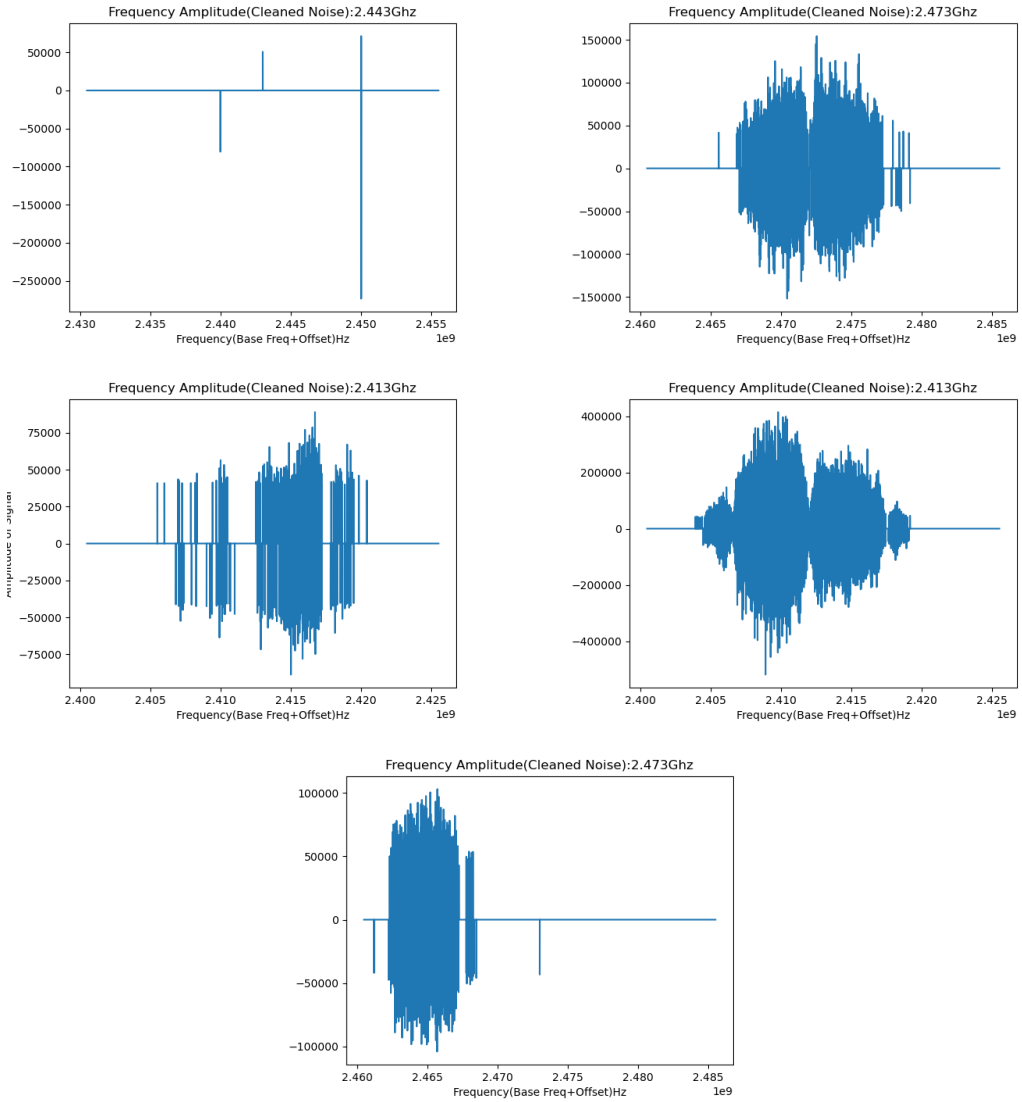


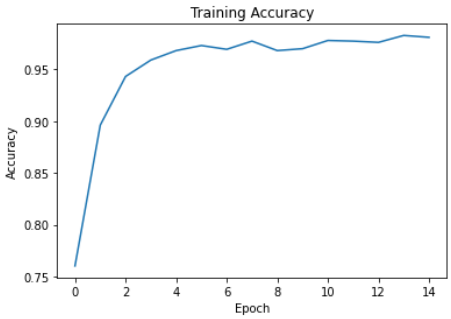
Figure 5.2: Sample Non-Drone signals

- As seen above, drone and non-drone signals have a distinct frequency signature in the same frequency bands with occasional frequency overlaps.

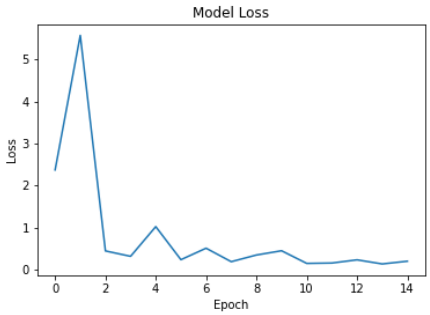
- This is used as one of the primary criteria during classification, which is distinct from their energy signal levels.

- After successful creation and dataset manipulation, the model is trained for a batch size of 1665 readings per epoch. After a successful run of 15 epochs,

the model prediction accuracy and loss function are shown as graphically plots as follows:



(a) Model Accuracy



(b) Model Loss

- As seen above, the model tends to nearly maintain a relatively low loss while maintaining relatively high accuracy values over the range of the epoch cycles.

- Due to frequent data shuffling and randomization, there are some sudden changes in loss and accuracy values which signify that the model is prevented from overtraining and memorizing the correct labels over the epoch cycles while training.

- After successful data and model loading, the model can predict successfully under which category the inputs came.

- The model can give output class values in integer form that are later mapped to an array containing the category names as its index.

  eg: '0' corresponds to non-drone and '1' corresponds to FlySky drone.

- The jammer section later uses this classification to target the corresponding spectrum in which the drone is operating.

Chapter 6

Conclusion

6.1 General Conclusion

From the overall implementation of the detection system we can conclude that:

- Drone and Wi-Fi signals have a very distinct frequency signature even though they cover relatively the same frequency bands.
- Bluetooth and Wi-Fi both use similar center frequencies, which makes it hard for us to distinguish them on the basis of center frequencies alone.
- Amount of bandwidth used by both Drone and Bluetooth signals is similar, which also makes it hard for us to distinguish between them based on bandwidth alone.

Hence for our application, both bandwidth and center frequencies are part of the frequency signature that we use to distinguish between drone and non-drone signals. We have categorized both Wi-Fi and Bluetooth as non drone and assigned them to a single category.

6.2 Challenges

During the Course of the project journey we encountered numerous challenges as follows

- PlutoSDR has a limited SNR sensitivity response:
  - During the course of the project,we encounter the issue of high internal noise generated in the PlutoSDR during data capture even in the absence of any signals in the vicinity.
  - It was however not completely rectified, but was lowered to a situation wherein it was not a major influence, which resulted in lowered sensitivity in the captured data.
- CSV resulting in larger loading times:
  - During the training phrase, a majority of the time was consumed by the system in simply loading of the stored dataframe in the csv format
  - This can be overcome by the used of .npy binary files in Python,which is however not suitable for our use-cases for generating the training

dataset for the model.

- Training requires longer time for Larger Dataset Length
  - It was observed that for larger dataset lengths in our application,the system used to take a relatively long time in completing the epoch training cycle
  - It was however reducible to a lower extent by using batch training methodology which as a trade-off leading to lower model category accuracy at times
- Better Equipment required for better clarity of Data
  - At times it was felt that higher clarity of data is required to perform advanced data extraction and manipulation of the input data
  - This was superseded by our desire to rapidly prototype as other SDRs which could provide better data had insufficient documentation and support for Python.

6.3 Future Work

- As seen the in prior section,there is an issue in differentiation between Bluetooth and Wi-Fi signals,which can overcome by quantitative analysis of the Bluetooth and Wi-Fi signal signatures along with their signal power regulations permitted by their specifications.

- Further training under multiple drone makers can be conducted in order:
  - Enable the system to identify more drones.
  - To identify drone make and model based on their respective RF Signature.
- The recorded signals can be studied in time and frequency domain to find more correlation and differences between drone and non-drone signals.
- Further studies can be done to distinguish between Bluetooth and drone signals based on their symbols and modulation schemes.

Bibliography

[1] D. Mototolea and C. Stolk, "Software Defined Radio for Analyzing Drone Communication Protocols," 2018 International Conference on Communications (COMM), 2018, pp. 485-490, doi: 10.1109/ICComm.2018.8484821.

[2] Drone ecosystem policy Roadmap - Digital Sky Platform ,Ministry of Civil Aviation Government of India https://aviation.report/whitepapers/drone-ecosystem-policy-roadmap-book

[3] Lichtman, M., 2021. PySDR: A Guide to SDR and DSP using Python — PySDR: A Guide to SDR and DSP using Python 0.1 documentation. [online] Pysdr.org. Available at: ¡https://pysdr.org/¿ [Accessed 6 March 2021].

[4] Aniket Mahanti, Niklas Carlsson, Carey Williamson, Martin Arlitt. Ambient Interference Effects in Wi-Fi Networks. 9th International IFIP TC 6 Networking Conference (NETWORKING), May 2010, Chennai, India. pp.160-173, ⟨10.1007/978-3-642-12963-6_13⟩. ⟨hal-01056315⟩

[5] M. Ezuma, F. Erden, C. Kumar Anjinappa, O. Ozdemir and I. Guvenc, "Detection and Classification of UAVs Using RF Fingerprints in the Presence of Wi-Fi and Bluetooth Interference," in IEEE Open Journal of the Communications Society, vol. 1, pp. 60-76, 2020, doi: 10.1109/OJCOMS.2019.2955889.

Appendices

Appendix A

Appendix

A.0.1 Installation instructions for libraries and environment

The following instructions were carried inside a virtual machine running ubuntu 20.04

Installation instructions for Conda

1. Download the miniconda installer from the conda website.
2. Navigate to the folder where you have downloaded the installer and open it inside a terminal.
3. run the following command to make the installer executable

```
chmox +x <miniconda installer>
```

4. run the following command to run the installer

```
./<miniconda installer>
```

Installation instructions for Tensorflow with Conda

Open a terminal and run following command to create a virtual environment for Tensorflow

```
conda create -n tf tensorflow
```

The above command creates a virtual environment for Tensorflow and installs all the necessary packages required. To activate the above virtual environment run

```
conda activate tf
```

Installation instructions for drivers and libraries of PlutoSDR

PlutoSDR requires libiio, libad9361-iio and pyadi-iio for interfacing. Before installing any of these libraries, we will need some build tools. To install the build tools run the following command

```
sudo apt-get install git libxml2 libxml2-dev bison flex libcdk5-dev \
cmake python3-pip
```

The Installation instructions for the libraries are as follows.

- libiio

  Run following commands inside terminal to install libiio

  ```
  cd ~
  git clone --branch v0.19 https://github.com/analogdevicesinc/libiio.git
  cd libiio
  cmake ./
  make all -j4
  sudo make install
  sudo ldconfig
  cd bindings/python/
  sudo python3 setup.py.cmakein install
  ```

- libad9361-iio

  Run following commands inside terminal to install libad9361-iio

```
cd ~

git clone https://github.com/analogdevicesinc/libad9361-iio.git

cd libad9361-iio

cmake ./

make -j3

sudo make install
```

- pyadi-iio

  Run following command inside terminal to install pyadi-iio

```
cd ~

git clone https://github.com/analogdevicesinc/pyadi-iio.git

cd pyadi-iio

sudo python3 setup.py install
```

A.0.2 Code

Code for Capturing the Data

```
#@Author:Pritesh Naik

import numpy as np

import adi    #(adi-Library for PlutoSDR)

import math

from scipy import signal

import pandas as pd


sample_rate = 250e4 # Hz

b,a =signal.butter(3,.2)

final = np.array([])

flist=np.array([])


##Function Calls

def filter(b,a,freq):

   z1=signal.lfilter_zi(b,a)

   pl=signal.lfilter(b,a,freq)

   return(pl)


def setup(center_freq,sample_rate,final):
```

```
#PSD_Format

    def power(val):

     p=abs(val**2)

     pdb=math.log10(p)

     return(pdb)


##SDR Setup Commands

    power=np.vectorize(power)

    fname = center_freq/1e9

    name=str('{:.3f}Ghz'.format(fname))

    sdr = adi.Pluto("ip:192.168.2.1")

    sdr.sample_rate = int(sample_rate)

    # filter cutoff for Frame Bandwidth

    sdr.rx_rf_bandwidth = int(sample_rate)

    sdr.rx_lo = int(center_freq)

    sdr.rx_buffer_size = 1024  # buffer size used by PlutoSDR


    ##Sampling Signal at Carrier Freq

    samples = sdr.rx()

    freqx=np.linspace(int(-sample_rate//2+center_freq)

       ,int(sample_rate//2+center_freq),int(1024))

    frq=np.fft.fft(samples)

    freq=np.fft.fftshift(frq)
```

```
  #Detecting Required Amplitude Peaks
    sig_avg = signal.find_peaks(filter(b,a,freq),height=10000)
    print('Size:',sig_avg[0].size)
    temp=np.array(sig_avg[0])
    if(sig_avg[0].size>0):
        print("Active Band:{}".format(name))
        for i in range(0,temp.size):
            temp[i]=temp[i]*(sample_rate/1000)+center_freq
        print("Signal Peaks:",temp)
        final=np.concatenate((final, temp))
    return final


#Rx Function Codes
final = np.array([])
print("Sample rate:",sample_rate,'Hz')
for center_freq in range(2405,2715,3):
   center_freq=int(center_freq*1e6)
   print("Center_freq:",center_freq,'Hz')
   final1=setup(center_freq,sample_rate,final)
   final=final1
print("Final Peaks_Array:",final)
store=pd.DataFrame(final)
```

```
store.to_csv("result.csv")
```

Code for labeling the Dataset

```
import pandas as pd

import numpy as np


#Code for labeling and merging the sampled data to generate a testing set

y = 0

while y < 21:

    c=np.array([])

    e = 0

    n = 0  #insert starting file

    while n < 20:  #select more than last file

        z ="Pure-Wifi_{}.00".format(n)  #modify file name as per required

        initial=pd.read_csv(r'F:\Home-wifi\{}\{}.csv'

          .format(y,z),usecols=[1])

        comp= initial.to_numpy()


         a=np.array([])

     # Add machine learning labels.

     # 1 if drone data set and 0 if non-drone dataset

        a=np.append(a,0)
```

```
        for b in comp:
            a =np.append(a,b[0])
        print(y,n)

        d =0
        d = np.sum(a)
        print(d)
        if d != 0:
            if e == 0:
                c = np.copy(a)
                e = e + 1
            else:
                c = np.column_stack((c,a))
        n = n + 1
    storeR=pd.DataFrame(c)
    storeR.to_csv(r'F:\Home-wifi\{}\testing-set.csv'.format(y),
      header=None, index=False)
    y = y + 1
print("end")


#Code to Merge multiple training sets into one set
#with different ratios of drone and non drone datasets
e = 0
```

```
c=np.array([])

y = 0

while y < 6:

    initial=pd.read_csv(r'F:\Drone-Set-new\VA+VB\{}\testing-set.csv'

      .format(y),header=None)

    comp= initial.to_numpy()

    print(comp.shape)

    if e == 0:

        c = np.copy(initial)

        print(c.shape)

        e = e + 1

    else:

        print(c.shape)

        c = np.concatenate((c,comp),axis = 1)

    y = y + 1


z = 16

while z < 21:

    initial=pd.read_csv(r'F:\Home-wifi\{}\testing-set.csv'

      .format(z),header=None)

    comp= initial.to_numpy()

    print(comp.shape)

    print(c.shape)
```

```
    c = np.concatenate((c,comp),axis = 1)

    z = z + 1

print(c.shape)



np.random.shuffle(np.transpose(c))

print(c.shape)



np.transpose(c)

print(c.shape)



storeR=pd.DataFrame(c)

storeR.to_csv(r'F:\Drone-Set\VA+VB\mixedColms.csv',

  header=None, index=False)

print("end")
```

Code for Training the Neural Network

```
import tensorflow as tf

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from tensorflow.keras import layers

from tensorflow.keras.layers.experimental import preprocessing

normalize = preprocessing.Normalization()

np.set_printoptions(precision=4)


#Reading csv File

print("CSV File Reading....")

df = pd.read_csv("Training-set-Trans2.csv")

train_features = df.copy()


#Classification Feature Extraction(1/0)

train_labels = train_features.pop('0')

train_features = np.array(train_features)

batch_size=train_labels.shape[0]

normalize.adapt(train_features)


#Converting to Dataset
```

```
train_dataset = tf.data.Dataset.from_tensor_slices(

  (train_features, train_labels))



#Dataset-Randomizer

BATCH_SIZE = 1

SHUFFLE_BUFFER_SIZE = int(batch_size)

train_dataset = train_dataset

  .shuffle(SHUFFLE_BUFFER_SIZE).batch(BATCH_SIZE)



#Model Creation

model = tf.keras.Sequential([

   normalize,

   tf.keras.layers.Dense(256,activation='relu'),

   tf.keras.layers.Dense(128),

   tf.keras.layers.Dense(64, activation='relu'),

   tf.keras.layers.Dense(32),

   tf.keras.layers.Dense(16, activation='relu'),

   tf.keras.layers.Dense(2)

 ])

model.compile(optimizer=tf.keras.optimizers.RMSprop(),

              loss=tf.keras.losses

              .SparseCategoricalCrossentropy(from_logits=True),

              metrics=['accuracy',])
```

```
print("Model Training")

history=model.fit(train_dataset,epochs=12)


# Training Parameter Plotting

plt.title("Training Accuracy")

plt.plot(history.history['accuracy'])

plt.ylabel("Accuracy")

plt.xlabel("Epoch")

plt.legend(["Accuracy"],loc="upper left")

plt.savefig("Model-Accuracy.png")

plt.plot(history.history['loss'])

plt.title("Model Loss")

plt.ylabel("Loss")

plt.xlabel("Epoch")

plt.legend(["Model Loss"],loc="upper left")

plt.savefig("Model-Loss.png")

print("Model Evaluating")


#Creating Testing-Dataset

df = pd.read_csv("Training-set-Trans2.csv")

test_features = df.copy()

test_labels = test_features.pop('0')

test_features = np.array(test_features)
```

```
test_dataset = tf.data.Dataset.from_tensor_slices(

  (test_features, test_labels))

test_loss, test_acc = model.evaluate(test_features,

  test_labels,verbose=2)



#Outputted Results

print("At Loss: {:5.2f}%".format(100 * test_loss))

print("Accuracy: {:5.2f}%".format(100 * test_acc))

print("Tested Against:",test_labels.shape[0]," Values")

model.save("MyModel1")
```

Code for Classifying the output

```
import tensorflow as tf

import pandas as pd

import numpy as np


model = tf.keras.models.load_model('MyModel1')

df = pd.read_csv("Testing-set.csv")

predict_features = df.copy()


predict_features = np.array(predict_features)

predictions = model.predict(predict_features)

predictions = tf.nn.sigmoid(predictions)

print('Predictions:')

for i in predictions:

  print(np.argmax(i),end =" ")
```

Appendix B

Data Sheets

In our project we primarily used two Software defined Radios

1. RTLSDR

2. PlutoSDR

ANALOG DEVICES
AHEAD OF WHAT'S POSSIBLE™

ADALM-PLUTO

SDR Active Learning Module

Product Overview

The easy to use ADALM-PLUTO active learning module (PlutoSDR) helps introduce electrical engineering students to the fundamentals of software-defined radio (SDR), radio frequency (RF), and wireless communications. Designed for students at all levels and from all backgrounds, the module can be used for both instructor-led and self-directed learning to help students develop a foundation in real-world RF and communications that they can build on as they pursue science, technology, or engineering degrees.

Connecting RF Theory with RF Practice

The PlutoSDR works as a portable lab that, when used with a host, can augment classroom learning. MATLAB® and Simulink® are two of the many software packages supported by PlutoSDR, and it provides an intuitive graphical user interface (GUI) so students can learn faster, work smarter, and explore more.

Made for Teachers, Students, and Self-Learners

The PlutoSDR features independent receive and transmit channels that can be operated in full duplex. The active learning module can generate or acquire RF analog signals from 325 MHz to 3800 MHz at up to 61.44 megasamples per second (MSPS). Small enough to fit in a shirt pocket, the PlutoSDR is completely self-contained and entirely USB powered with the default firmware. Because PlutoSDR is enabled by libiio drivers, it supports OS X®, Windows®, and Linux®, which allows students to learn and explore on a variety of devices.

With dozens of available online tutorials for SDR-based projects, PlutoSDR boasts labs and teaching material covering topics such as ADS-B aircraft position, receiving NOAA and Meteor-M2 weather satellite imagery, GSM analysis, listening to TETRA signals, pager decoding, and many more!

Features

- Portable self-contained RF learning module
- Cost-effective experimentation platform
- RF coverage from 325 MHz to 3.8 GHz
- Flexible rate, 12-bit ADC and DAC
- One transmitter and one receiver (female SMA, 50 Ω)
- Half or full duplex
- MATLAB, Simulink support
- GNU Radio sink and source blocks
- Libiio, a C, C++, C#, and Python API
- USB 2.0 interface
- Plastic enclosure
- USB powered
- Up to 20 MHz of instantaneous bandwidth (complex I/Q)

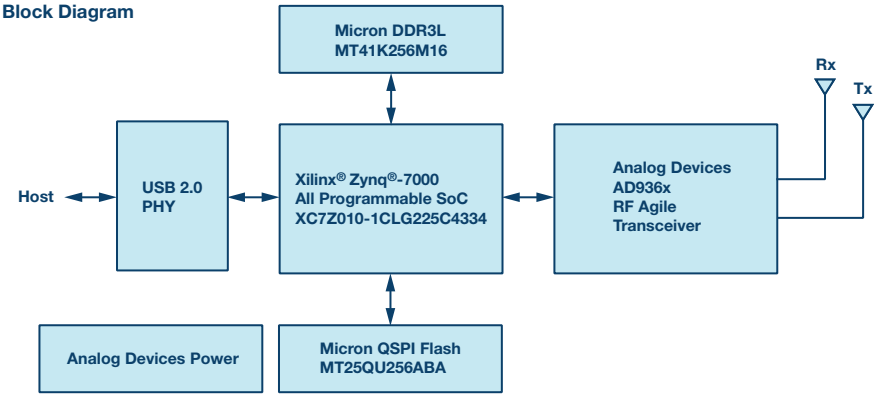
Kit Includes

- Analog Devices PlutoSDR active learning module
- Two antennas (824 MHz to ~894 MHz/ 1710 MHz to ~2170 MHz)
- One 15 cm SMA cable
- One USB cable

To purchase this active learning module, visit *analog.com/plutosdr*.

Visit *analog.com*

Simplified Block Diagram



PlutoSDR Host Interface

The PlutoSDR supports the following USB device classes:

- Mass storage (for easy firmware updates)
- Serial (for interacting with the Linux kernel/userspace on PlutoSDR)
- Networking/RNDIS (for loading and controlling custom ARM® applications)
- Libiio (bulk USB for SDR data transfer and control)
- Device firmware upgrade (for backup firmware upgrades)

Open Source

The PlutoSDR open source firmware is built from Das U-Boot, the Linux kernel, and Buildroot. As part of the class materials, the firmware is able to be run, copied, distributed, studied, changed, and improved with Vivado® HL WebPACK™ Edition (license free). Supporting USB 2.0 On-the-Go, the PlutoSDR can attach to a variety of USB peripherals (wired networking, Wi-Fi dongles, audio, etc.), which extends functionality. All documentation is open and available at *wiki.analog.com/plutosdr*.

University and Active Learning Program

ADI's many learning activities provide faculty and students with the opportunity to further develop the theory, understanding, and practice to interpret the world around them by intelligently bridging the physical and digital realms with unmatched ADI technologies that sense, measure, and connect. From select sponsored faculty research to samples program, ADI provides faculty and student access to ADI products, information, teaching materials, and mentoring for use in research, thesis, and undergraduate projects. ADI offers co-ops and internships, and hires many new college graduates worldwide. More information can be found at *analog.com/university*.

Specifications	Typical
Power	
DC Input (USB)	4.5 V to 5.5 V
Conversion Performance and Clocks	
ADC and DAC Sample Rate	65.2 kSPS to 61.44 MSPS
ADC and DAC Resolution	12 bits
Frequency Accuracy	±25 ppm
RF Performance	
Tuning Range	325 MHz to 3800 MHz
Tx Power Output	7 dBm
Rx Noise Figure	<3.5 dB
Rx and Tx Modulation Accuracy (EVM)	–34 dB (2%)
RF Shielding	None
Digital	
USB	2.0 On-the-Go
Core	Single ARM Cortex®-A9 @ 667 MHz
FPGA Logic Cells	28k
DSP Slices	80
DDR3L	4 Gb (512 MB)
QSPI Flash	256 Mb (32 MB)
Physical	
Dimensions	117 mm × 79 mm × 24 mm 4.62" × 3.11" × 0.95"
Weight	114 g
Temperature	10°C to 40°C

EngineerZone® Online Support Community

Engage with the PlutoSDR developers in the virtual classroom, as well as ADI's technology experts in our online support community.

Visit *ez.analog.com/community/university-program*

EngineerZone™ SUPPORT COMMUNITY

Analog Devices, Inc.
Worldwide Headquarters

Analog Devices, Inc.
One Technology Way
P.O. Box 9106
Norwood, MA 02062-9106
U.S.A.
Tel: 781.329.4700
(800.262.5643, U.S.A. only)
Fax: 781.461.3113

Analog Devices, Inc.
Europe Headquarters

Analog Devices GmbH
Otl-Aicher-Str. 60-64
80807 München
Germany
Tel: 49.89.76903.0
Fax: 49.89.76903.157

Analog Devices, Inc.
Japan Headquarters

Analog Devices, KK
New Pier Takeshiba
South Tower Building
1-16-1 Kaigan, Minato-ku,
Tokyo, 105-6891
Japan
Tel: 813.5402.8200
Fax: 813.5402.1064

Analog Devices, Inc.
Asia Pacific Headquarters

Analog Devices
5F, Sandhill Plaza
2290 Zuchongzhi Road
Zhangjiang Hi-Tech Park
Pudong New District
Shanghai, China 201203
Tel: 86.21.2320.8000
Fax: 86.21.2320.8222

analog.com

ANALOG DEVICES
AHEAD OF WHAT'S POSSIBLE™

RTL-SDR Blog V3 Datasheet

The RTL-SDR Blog V3 is an improved RTL-SDR dongle. RTL-SDR dongles were originally designed for DVB-T HDTV reception, but they were found by hardware hackers to be useful as a general purpose SDR. The standard dongles are okay for DVB-T reception, but are just barely suitable for SDR users/experimenters. The RTL-SDR Blog V3 was redesigned with SDR user needs in mind, instead of DVB-T HDTV users who typically have more relaxed requirements.

Purchase at: www.rtl-sdr.com/store

Quickstart setup guide available at: www.rtl-sdr.com/qsg

Basic Information

- **Bandwidth:** Up to 2.4 MHz stable.
- **ADC:** RTL2832U 8-bits
- **Frequency Range:** 500 kHz – 1766 MHz (500 kHz – 24 MHz in direct sampling mode)
- **Typical Input Impedance:** 50 Ohms
- **Typical Current Draw:** 270 – 280 mA

Required Computing Hardware

Same requirements as a regular RTL-SDR. Compatible with Windows XP and above (SDR# requires Win 7 or newer), Linux, MacOS and Android. A dual core machine is recommended.

Single board PCs like the Raspberry Pi, Odroid, C.H.I.P are also supported with most command line apps.

RTL-SDR V3 Improvements over generic models

TCXO

The V3 uses a 1PPM TCXO for excellent frequency stability. The temperature drift is around 0.5 – 1 PPM, and the initial offset is 0 – 2 PPM. This means that the signal will not drift on the spectrum as the dongle or ambient temperature changes. Also, the frequency offset will be close to zero. Standard dongles have a PPM offset of up to 100PPM, and tend to drift a lot. Using a TCXO solves these problems.

SMA Connector

Typical RTL-SDR dongles use a relatively obscure MCX RF connector. The V3 uses commonly used SMA connectors, so it is easy to obtain adapters, connectors and antennas for the unit. SMA connectors also last longer.

Aluminium Enclosure

Unlike standard RTL-SDR's, the V3 comes standard with an aluminium enclosure. The enclosure has two purposes. The first is to help block any RF interference from entering through the PCB. The second is to act as a heatsink to the PCB.

Improved Heat Dissipation

Typical R820T/2 RTL-SDR dongles tend to lose PLL lock in L-band at around 1.5 GHz and above, causing a loss of reception to those frequencies. The reason is due to the high heat generated by the R820T2 chip. The V3 uses a thin thermal pad to thermally bond the PCB and metal enclosure together. This allows the metal case to work as a heat sink, which solves the PLL lock problem. Ideally the thermal pad should be as thin as possible to enhance maximum heat transfer, and we have designed the enclosure so that the thermal pad only needs to be 3mm thick.

The V3 also uses a larger ground plane on the middle layers of the PCB which also helps with heat dissipation.

R820T2 Chip

Older RTL-SDR units used the R820T chip. There is a newer R820T2 which has slightly better manufacturing tolerances. The R820T2 is produced in a factory with higher quality silicon which allows for more reliable chips. A side effect of the better silicon is overall slightly better and more stable sensitivity across manufacturing runs compared to the R820T, and less PLL lock problems at L-band frequencies.

Improved ESD protection on the RF front end

The BAV99 diode which is used on most RTL-SDR dongles is not a true ESD rated diode. We have added a real ESD rated diode for better protection. The BAV99 remains in the circuit as it works a strong signal clipper, which prevents damage to the R820T2 from overly strong signals. Please remember that not even this will save your radio from a lightning strike or huge ESD impulse, and any permanently outdoor mounted antenna system must have its own lightning and ESD protection. To help avoid lightning damage unplug your antenna during a storm and when the dongle is not in use.

Improved front end circuit

The standard matching circuit on the RTL-SDR was designed for DVB-T use, and tends to attenuate signals above ~1 GHz. The new matching circuit has less attenuation above 1 GHz and similar performance below. We have used high quality, high SRF, high Q inductors in this circuit.

Software switchable 4.5v bias tee.

The V3 makes use of a low noise LDO and one of the GPIO pins on the RTL2832U to provide a 4.5V bias tee that can be activated in software. The bias tee can pull about 180 mA continuously so is suitable for the majority of 3-5V powered LNAs that are popular with RTL-SDR devices. The bias tee is protected against accidental short circuits at the LDO level, and with a thermal auto-resetting PTC fuse. See 'Activating the Bias Tee' for more information on software for activating the bias tee.

This bias tee is great for powering a remote LNA (like Adams PSA5043+ based LNA4ALL) or something like the SpyVerter upconverter.

Bias Tee Warning: The bias tee thermal fuse or LDO could be damaged if you short circuit the bias tee for long periods of time. Before turning on the bias tee, ensure the circuit to be powered is not shorted, or that the RTL-SDR is not connected to a DC shorted antenna!

Lower Voltage Operation

The V3 uses an LDO that has a much lower 'dropout' voltage compared to the typical AMS1117 LDO used on most dongles. Hence the V3 should run better on long USB extension cables.

Long USB cables tend to drop the 5V USB voltage down to lower levels. Below about 4V the AMS1117 stops working. The LDO used in the V3 works almost down to 3.3V.

Of course, with low voltages from long USB cable, the bias tee will be unable to put out 4.5V. At low voltages the bias tee LDO will revert to a non-filtered voltage slightly under the supply.

Reduced noise with a modified PCB design

Typical RTL-SDR dongles use 2-layer PCB designs and route signal lines improperly. The V3 uses a modified 4-layer PCB design which helps to significantly reduces clock spurs and noise pickup.

The V3 also adds a USB common mode choke on the USB data lines to reduce USB noise, adds SMD ferrite chokes on the PCB power lines, and uses a lower noise LDO.

HF direct sampling circuit, diplexed out from the SMA connector

The idea behind direct sampling mode is that an antenna can be connected directly to the ADC pins of the RTL2832U, and this can enable HF reception. This is useful because the R820T/2 tuner can only tune down to about 24 MHz at the lowest. On typical R820T RTL-SDR dongles one can enable direct sampling mode by soldering a wire to the Q-branch pins of the RTL2832U. The RTL2832U samples at 28.8 MHz, so 0 – 14.4 MHz, and 14.4 MHz – 28.8 MHz can be listened to.

The V3 has direct sampling mode implemented in hardware already, so no hardware mods are required to listen to HF via direct sampling.

To split the HF signal out at the SMA connector, a diplexer tuned to 25 MHz is used. A 10dB buffer preamp sits after the diplexer which helps to boost the signal and overcome losses in the subsequent filter and impedance transformer. After the preamp is a 24 MHz low pass filter and then an impedance matching and single to double ended transformer. The addition of the preamp, filter and transformer ensures good direct sampling performance.

The result is that 500 kHz to about 24 MHz can be received in direct sampling mode.

Direct sampling could be more sensitive than using an upconverter, but dynamic won't be as good as with an upconverter. It can overload easily if you have strong signals since there is no gain control. And you will see aliasing of signals mirrored around 14.4 MHz due to the Nyquist theorm. But direct sampling mode should at least give the majority of users a decent taste of what's on HF. If you then find HF interesting, then you can consider upgrading to an upconverter like the SpyVerter (the SpyVerter is the only upconverter we know of that is compatible with our bias tee for easy operation, other upconverters require external power).

If you search on YouTube for "RTL-SDR V3", you will find several videos showing what you can get in direct sampling mode. Most people are surprised at how good it can be, but also many users will need a broadcast AM filter to reduce overloading. We sell a suitable broadcast AM filter on our store www.rtl-sdr.com/store.

Expansion pads on the PCB

Access pads for the unused GPIO pins, CLK in/out, 3.3V, GND and I2C pins have been added. The CLK input/output is disconnected by default. Access pads for the I branch have also been added as some users and industrial customers are using these in special projects. These pads are only for advanced users who need them for special projects. Take care as these pins are not ESD protected.

Clock selector jumper

By soldering in a 4 pin 1.27mm pitch jumper header and removing the default 0 Ohm resistor, one can now easily select between the onboard clock, an external clock, or having the on board clock be the output for another dongle. This is for advanced users only who want to experiment with things like passive radar, and coherent receivers.

Corner mounting holes for those who want to stack PCBs.

Some customers have been building devices that require multiple RTL-SDR dongles, and these standoff holes should aid in stacking.

Feature Information

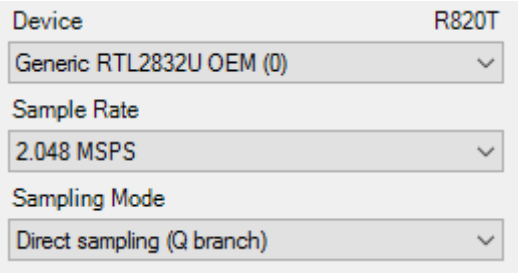
Feature 1: Direct Sampling HF Mode

This feature allows you to listen to HF signals between about 500 kHz to 28.8 MHz.

To use direct sampling mode first connect an appropriate HF antenna to the SMA antenna port (this is the same port where you connect your VHF/UHF antenna).

In SDR# select the Q-branch in the configure menu (the cog icon next to the play button). (If it is greyed out make sure you stop the SDR first, by clicking the stop button in SDR#)

Press Play and tune to 500 kHz – 28.8 MHz.



VHF antennas like small discones or short whip antennas will probably not pick up HF signals very well, if at all. If you have no such antenna you *might* get something with the large telescopic antenna extended to its maximum length of 1.5m, but really this is still not long enough for HF. You can instead use the screw nut provided with the antenna base to clamp on a long wire antenna that is 5 meters or more in length. Ideally you should use a 9:1 unun with the long wire antenna for optimal reception but it is not totally necessary. Even more ideally you'd use an antenna tuner, though this is expensive.

Other software like HDSDR and GQRX can also support direct sampling. It may entail setting a device string, and for the Q-branch, the value should be 2. In GQRX the device string would be "rtl=0,direct_samp=2" (without the quotes). Make sure that there is no space after the comma.

To go back to listening to frequencies above 28.8 MHz remember to change the sampling mode back to "Quadrature Sampling".

Note that this feature makes use of *direct sampling* and so aliasing will occur. The RTL-SDR samples at 28.8 MHz, thus you may see mirrors of strong signals from 0 – 14.4 MHz while tuning to 14.4 – 28.8 MHz and the other way around as well. If these images cause problems, then to remove them you will need to use a low pass filter for 0 – 14.4 MHz, and a high pass filter for 14.4 – 28.8 MHz. Either that or you can simply filter your exact band of interest.

Feature 2: Software Selectable Bias Tee

The V3 RTL-SDR introduces a bias tee which can be enabled easily in software.

WARNING: Before using the bias tee please ensure that you understand that you should not use this option when the dongle is connected *directly* to a DC short circuited antenna. Although the bias tee circuit is dual protected against accidental shorts with a PTC automatically resetting fuse and overcurrent protection on the LDO, short circuiting the bias tee for an extended period (hours) could damage the LDO or fuse permanently. Only use it while connected to an actual powered device, like an LNA, active antenna or the SpyVerter.

To make things clearer: DC Short Antenna -> LNA -> Coax -> V3(bias tee on) is fine. What's not good and makes no sense anyway is DC Short Antenna -> Coax -> V3(bias tee on). DC Short Antenna -> Coax -> V3(bias tee off) is fine.

To enable the bias tee in Windows:

1. Download and extract all the files in the zip file downloadable at https://github.com/rtlsdrblog/rtl-sdr/releases/tag/v1.1 into a folder on your PC. It contains two batch files that can be run.

2. Next make sure that all SDR software like SDR#/HDSDR/SDR-Console etc is fully closed. If there is another program accessing the RTL-SDR the bias tee software will not run.
3. Run the biastee_on.bat file to turn the bias tee on. It will run and open a CMD prompt that will briefly say "Found Rafael Micro R820T Tuner". The CMD prompt will close soon after upon success.

The bias tee is now on. To turn it off repeat steps 2 & 3, but instead run the biastee_off.bat batch file. Alternatively, simply disconnect and then reconnect the SDR to turn the bias tee off.

If you have multiple dongles connected you'll need to edit the batch file to specify what dongle's bias tee you want to activate. Open the bat file with any text editor, like Notepad, and add the dongle selector "-d" flag. For example, to activate the bias tee on the dongle that was plugged in second you'd need to change it to "rtl_biast -b 1 -d 1".

If you get a Smart Screen message, click on More Info, and then on Run Anyway. Also note that some versions of Windows may fail to run batch files due to misconfiguration or aggressive antivirus software. If you cannot fix these problems with Windows or your antivirus, run the command manually on the CMD line.

To run it manually on the CMD line first browse to the directory where the bias tee software is stored using "cd" (e.g. cd C:\SDR\bias_tee_folder), and then run:

ON: rtl_biast -b 1

OFF: rtl_biast -b 0

If needed select a particular RTL-SDR device with the -d flag.

In Linux or MacOS download the source from git, compile it the same way you do the regular RTL-SDR drivers, and then run ./rtl_biast -b 1 to turn the bias tee on and ./rtl_biast -b 0 to turn the bias tee off. The procedure is:

```
git clone https://github.com/rtlsdrblog/rtl_biast
cd rtl_biast
mkdir build
cd build
cmake ..
make
cd src
./rtl_biast -b 1
```

If you want to be able to run the bias tee program from anywhere on the command line you can also run "sudo make install".

If you have trouble running the bias tee use a multimeter to check if there is 4.5V at the SMA port, and that your powered device is actually capable of receiving power. Remember that not all LNA's can accept bias tee power. We recommend Adam 9A4QV's LNA4ALL, as you can order this from his store with the bias tee power option enabled.

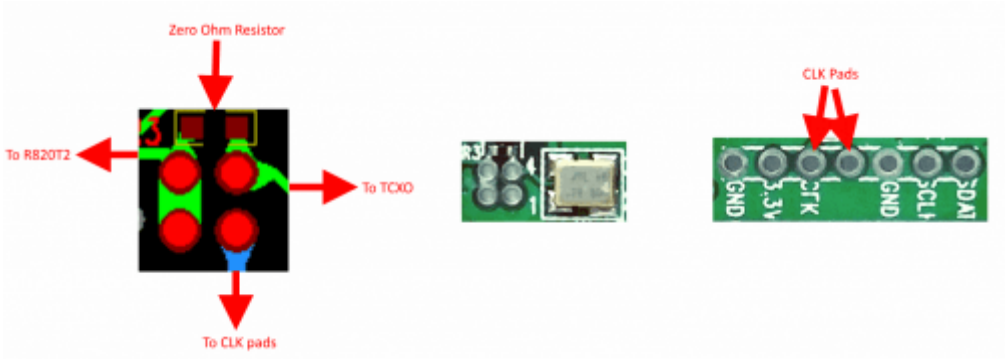
Feature 3: Selectable Clock & Expansion Headers

This is for advanced users who need to daisy chain clocks together for coherent experiments, or need to access other ports. You can either bridge the clock selector the directly with a solder bridge, or solder on a 1.27mm 2×2 header pin jumper.

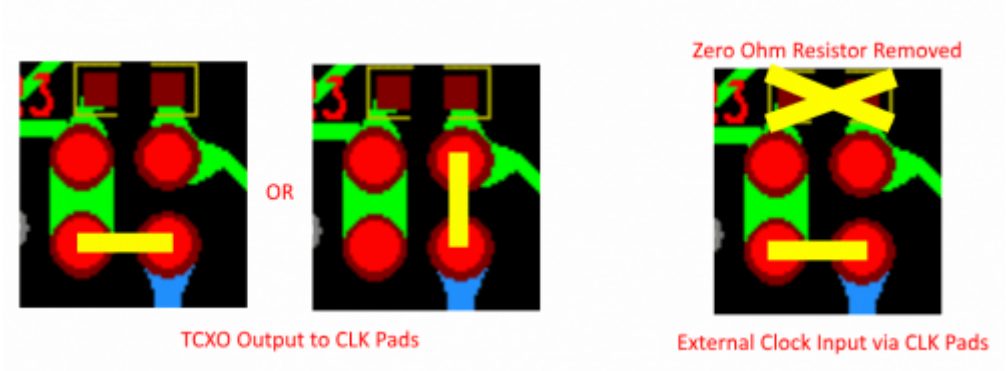
1. To add a jumper to the CLK selector header.
2. Carefully remove the 0 Ohm resistor.
3. Very carefully solder a 1.27mm 2×2 header onto the clock selector pads.

You can now select your clock input.

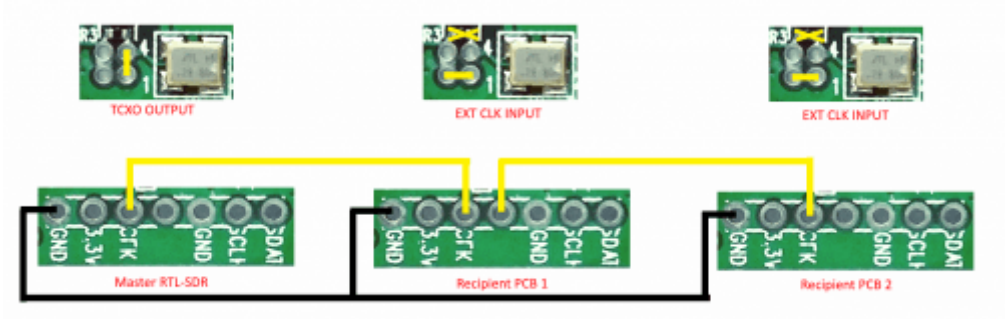
How to connect the CLK jumpers:



The first position allows you to output the dongles clock to the CLK pads. The second position allows you to input an external clock.



An example of CLK daisy chaining is shown below. One dongles TCXO is connected to two other dongles who have disconnected clocks.



LF Improvement / Bias Tee Disable Mod

If you want to improve the performance at LF/MW and do not require the bias tee, then you can remove the bias tee inductor at L13. Of course, remember that if you are really interested in VLF/LF, then it might be a better idea to use a VLF/LF compatible upconverter like the SpyVerter, which can be powered by the bias tee on the dongle. Obviously if you remove the bias tee inductor, the bias tee will no longer function, and so you'd have to power the SpyVerter externally via a USB cable.