Multi-Receiver Real-time Intrusion Detection with GNU Radio and USRP N310

Daren Swasey DAREN.SWASEY@USU.EDU
Department of Electrical and Computer Engineering, 4120 Old Main Hill Logan, UT 84322-4120 USA

Todd K. Moon TODD.MOON@USU.EDU
Department of Electrical and Computer Engineering, 4120 Old Main Hill Logan, UT 84322-4120 USA

Mirelle DeSpain MIRELLE2U@GMAIL.COM
Department of Electrical and Computer Engineering, 4120 Old Main Hill Logan, UT 84322-4120 USA

Abstract

Indoor digital communication, such as Wi-Fi, can be used to provide real-time indoor intrusion detection. In this work, we use signals internal to the radio receiver, such as the PLL or AGC block, to identify when a channel is changing due to a target moving within the physical channel. We describe a test using GNU Radio and Ettus USRP N310 hardware for both a test transmitter and receiver. Test results include a coverage map for a system in a large building.

1. Introduction

Indoor communication such as WiFi typically occurs over a multi-path channel, with transmitted signals reflecting from several surfaces before being received. Modern digital receivers deal with vagaries of such received signals using a variety of internal algorithms, including automatic gain control (AGC), symbol timing tracking, phase-locked loops (PLLs), and equalizers (Rice, 2009). Such receivers also deal with time-varying channels, such as channels having moving reflectors. This is a familiar phenomenon: users of WiFi are accustomed to working online even as objects in their vicinity (people walking, doors opening) are moving around.

Because the internal algorithms of a digital receiver adapt to compensate for time-varying channels, internal variables of these algorithms can provide an indication of when the channel has changed. In this paper, we explore using signals from internal algorithms as a means of detecting channel variations due to the presence of a human in the channel between a transmitter and a receiver. Basically, we want to reach inside a radio receiver and grab signals that can tell us when someone has walked into the channel. We call this problem the intrusion detection problem. More colloquially, this could be referred to as a burglar alarm. Development of this research could provide every WiFi-equipped home or business with a broad way of securing the premises. Similar research has been done with WiFi signals (Wang, et al., 2019; Z. Tian, et al., 2018), but those results remain outside the scope of GNU Radio.

Specifically we address the following questions: What internal algorithm signals can be used to provide an effective intrusion detection system, that provides a high probability of detection and low probability of false alarm? What is the coverage of a transmitter/receiver pair that provides good performance?

These questions are explored by a radio system implemented using GNU Radio. In our prototype system, we implemented a QPSK transmitter/receiver system, preparatory to future investigations using WiFi signals.

2. Feasibility Testing

The testing occurred in the building diagrammed in figure 1 (the Sant Engineering Research Building at Utah State University). Six antennae were placed in the ceiling of the second floor: northwest (NW), north center (NC), etc. The dimensions of the building, assuming largest distances, are approximately 40.84 m (134 ft) from east to west, and about 25.6 m (84 ft) from north to south.

Initial testing was done with the transmitter antenna in the lab, mounted approximately 1.2 m above floor level. The receive antenna used was mounted in the ceiling outside the entrance to the lab (just north of the “transmitter” location in figure 1), about 2.7 m to the north and 0.6 m to the east of the transmitter. By placing these in close proximity, we were able to test whether movement developed any response at all.

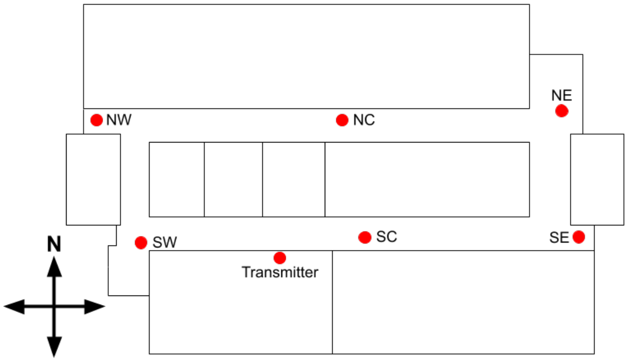


Figure 1: Original testing setup, modified from Bradshaw's figure 2.6

The QPSK transmitter was implemented in GNU Radio. Two Ettus USRP N310s were used, one as a transmitter and the other as a receiver. The transmitted signal ran through two Shireen 90305 amplifiers. Random data was transmitted in the Industrial, Scientific, and Medical (ISM) band at 905 MHz. Figure 2 illustrates the flowgraph for the transmitter and receiver.

Two signal scenarios were considered. The first was "quiescent" data, when it was determined (visually) that there were no people walking around on the floor. The other data was "active" data, when an intruder was present. The active data was obtained when a person was walking laps around the hallways.

2.1. Close-proximity Testing

The first tests used the output data from the *Carrier Phase Synchronization* block as a baseline. This block may also be referred to as a phase-locked loop (PLL). Figure 3 shows a side-by-side comparison between the raw quiescent and the raw active data from that block. There are obvious differences in the data, making it easier to isolate for intrusion detection purposes. Each prolonged spike seen in the active data is when the tester was closest to the receive antenna.

From the data it is visually clear when an intruder walked past the receive antenna. Of course, the data from the PLL was only one of many options. We also did testing using the AGC.

There are two signal conditions in which the AGC can be placed, bandpass and baseband. For bandpass, Rice recommends placing it after resampling takes place, which in our case is immediately after the USRP Source. The baseband location is after the matched filter, which is the RRC filter in our flowgraph.

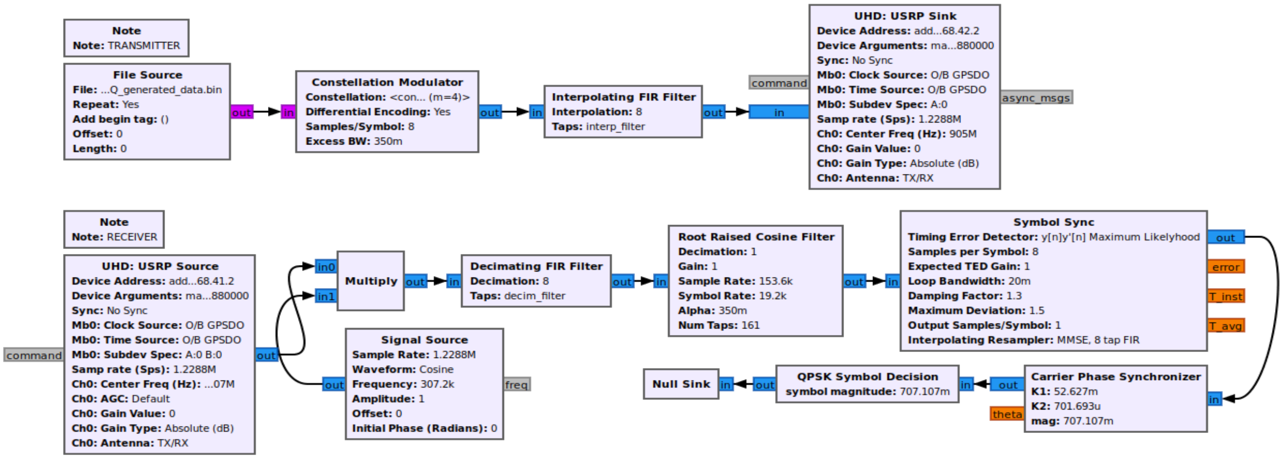
The AGC serves to suppress large variations, however, variables internal to the AGC in GNU Radio must contain that information in order to suppress it. The variable in GNU Radio that provides that information is *_gain*. This gain variable is equivalent to *A(n+1)* in equation 1 (Rice, 2009, p. 591),

$$A(n+1) = A(n) + \alpha\left[R - \left|\left(A(n)A_r(nT)\right)\right|\right] \qquad (1)$$

which is represented by the following in the GNU Radio AGC code within the *scale* function. *R* is equivalent to *_reference*, the input signal amplitude $A_r(nT)$ multiplied by the amplitude from the AGC *A(n)* is equivalent to *output*, and α is equivalent to *_rate*.

$$\_gain \mathrel{+}= \left(\_reference - fabsf\left(output\right)\right) * \_rate \qquad (2)$$

In order to access *_gain*, a custom AGC block was written based on code from GNU Radio's original AGC block, with an additional output port for the gain. Code written to the new files is highlighted in red boxes in figure 4. The original code is from the *agc_xx_impl.cc* and the *agc.h* files. The *work* function comes from the *agc_xx_impl.cc* file and the *scaleN*



Note
Note: TRANSMITTER

File Source
File: ...Q_generated_data.bin
Repeat: Yes
Add begin tag: ()
Offset: 0
Length: 0

Constellation Modulator
Constellation: <con... (m=4)>
Differential Encoding: Yes
Samples/Symbol: 8
Excess BW: 350m

Interpolating FIR Filter
Interpolation: 8
Taps: interp_filter

UHD: USRP Sink
Device Address: add...68.42.2
Device Arguments: ma...880000
Sync: No Sync
Mb0: Clock Source: O/B GPSDO
Mb0: Time Source: O/B GPSDO
Mb0: Subdev Spec: A:0
Samp rate (Sps): 1.2288M
Ch0: Center Freq (Hz): 905M
Ch0: Gain Value: 0
Ch0: Gain Type: Absolute (dB)
Ch0: Antenna: TX/RX

Note
Note: RECEIVER

UHD: USRP Source
Device Address: add...68.41.2
Device Arguments: ma...880000
Sync: No Sync
Mb0: Clock Source: O/B GPSDO
Mb0: Time Source: O/B GPSDO
Mb0: Subdev Spec: A:0 B:0
Samp rate (Sps): 1.2288M
Ch0: Center Freq (Hz): ...07M
Ch0: AGC: Default
Ch0: Gain Value: 0
Ch0: Gain Type: Absolute (dB)
Ch0: Antenna: TX/RX

Multiply

Signal Source
Sample Rate: 1.2288M
Waveform: Cosine
Frequency: 307.2k
Amplitude: 1
Offset: 0
Initial Phase (Radians): 0

Decimating FIR Filter
Decimation: 8
Taps: decim_filter

Root Raised Cosine Filter
Decimation: 1
Gain: 1
Sample Rate: 153.6k
Symbol Rate: 19.2k
Alpha: 350m
Num Taps: 161

Symbol Sync
Timing Error Detector: y[n]y'[n] Maximum Likelyhood
Samples per Symbol: 8
Expected TED Gain: 1
Loop Bandwidth: 20m
Damping Factor: 1.3
Maximum Deviation: 1.5
Output Samples/Symbol: 1
Interpolating Resampler: MMSE, 8 tap FIR

Carrier Phase Synchronizer
K1: 52.627m
K2: 701.693u
mag: 707.107m

QPSK Symbol Decision
symbol magnitude: 707.107m

Null Sink

Figure 2: GNU Radio flowgraph for original transmitter and receiver setup

function can be found in the *agc.h* file.

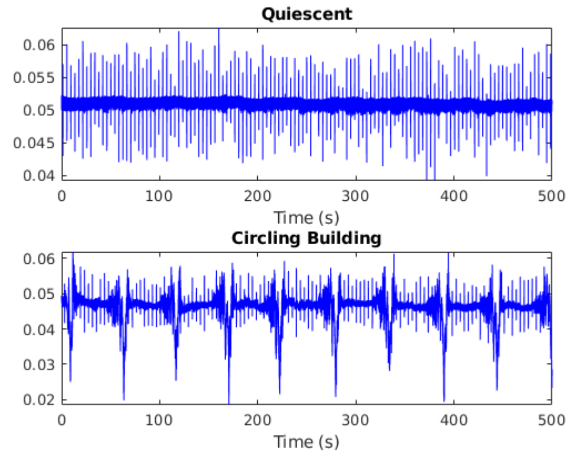
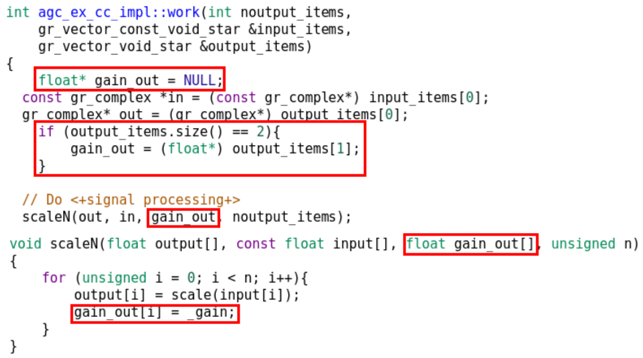


Figure 3: Raw quiescent and active data from Carrier Phase Synchronizer in close-proximity setting.



```
int agc_ex_cc_impl::work(int noutput_items,
    gr_vector_const_void_star &input_items,
    gr_vector_void_star &output_items)
{
    float* gain_out = NULL;
  const gr_complex *in = (const gr_complex*) input_items[0];
  gr_complex* out = (gr_complex*) output_items[0];
    if (output_items.size() == 2){
        gain_out = (float*) output_items[1];
    }

  // Do <+signal processing+>
  scaleN(out, in, gain_out, noutput_items);

 void scaleN(float output[], const float input[], float gain_out[], unsigned n)
 {
     for (unsigned i = 0; i < n; i++){
         output[i] = scale(input[i]);
         gain_out[i] = _gain;
     }
 }
```

Figure 4: Code additions for custom AGC

The custom AGC was placed immediately after the USRP Source, where the incoming signal was still at bandpass frequencies. Bandpass data can be slightly more effective and responsive than baseband AGC data, because the baseband signal amplitude has been somewhat normalized by the matched filter (Rice, 2009, p. 589). The quality of the gain data here functioned equally as well as the data from the carrier phase synchronizer.

Though testing with the initial antenna configuration was promising, it is unlikely that a transmitter and receiver would be so close in a real-world situation. Such a setup would also be ineffective, as it limits the direct path length and amount of possible multi-path reflections, which seem to influence the detectable range of each receiver.

2.2. In-Ceiling Testing with Custom AGC

After testing with the initial setup, we began to use the SW in-ceiling antenna as the transmitter and various other in-ceiling antennae were used as receivers. Using this setup, all data transmission and reception would happen in the ceiling of the building, where there is a less direct path from transmitter to receiver. After performing tests with a new transmitter location, it became clear that the system could not function as normal without the AGC. Due to this issue, the data from the PLL became irrelevant, as it could not function well without the AGC. At this point, the AGC gain data became the best option and was used for the remainder of our testing.

Data from the AGC gain indicated two viable detection conditions. One condition shows active movement, and the other is apparent when a non-moving target is present. Each condition can be identified in figure 5. Data for the figure is from the SE antenna in a test where a target stood in several locations around the building, each for a one-minute period. The detection metrics are based on the shown conditions, which were most apparent in the time domain. The frequency domain was less useful, but it also showed signal energy was a valid metric, bringing the number of detection metrics to three.

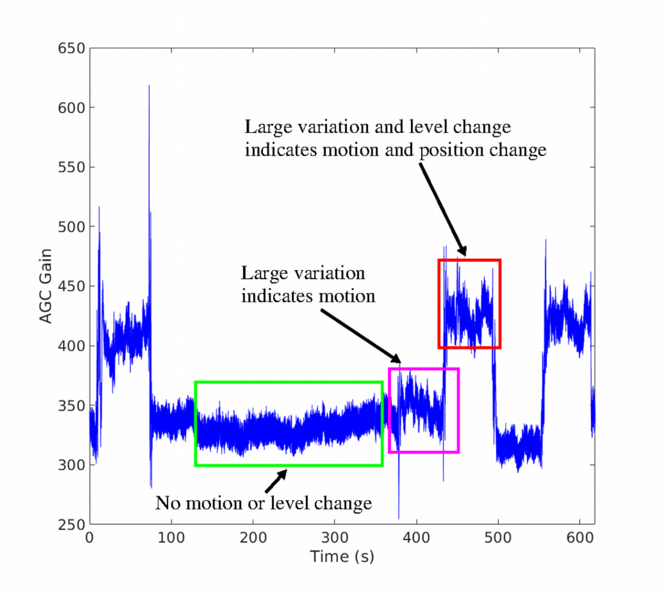


Figure 5: Categorized AGC data from SE antenna

2.3. Intrusion Detection Metrics

Three methods were implemented for threshold setting and intrusion identification. These include the standard deviation of the signal amplitude, standard deviation of the signal energy, and the average signal level. In real-time, each of these are calculated in a moving window. An effective length of the window proved to be anywhere between 1 and 3 seconds. Windows longer than this can be used, but are unreasonable for intrusion detection purposes due to lengthened update times. Processing on shorter windows often could not accurately detect intrusion conditions when using standard deviation.

Standard deviation is used as a metric for comparison because it can isolate localized signal variations. Using a metric like the amplitude of the signal would not capture enough information to make reasonable detection decisions. This is especially true of the AGC gain, as it could vary in range for different receivers from below 500 for one receiver to over 10,000 on others, depending on the signal strength at the receiver.

The level-based detection method is useful for determining when a target may be present but not in motion. Data in figure 5 shows the signal level shifting and remaining at different levels after the target moved to a new location. This information suggests that target localization is possible as well. That concept is not expanded upon here, but can certainly be explored in future research.

2.3.1. STANDARD DEVIATION OF SIGNAL AMPLITUDE

As can been seen in figure 5, amplitude variation is a clear indicator of movement. When there is movement, there are rapid fluctuations in the AGC gain as the AGC adjusts to the incoming signal and stabilizes the output. When there is no movement or presence, the amount of fluctuation decreases.

By calculating the standard deviation of the signal amplitude, we can capture useful information from the fluctuations. For the in-ceiling testing, standard deviation of signal amplitude provided the best overall receiver operating characteristic (ROC) curves. This indicates the highest probability of accurate detection, and the lowest probability of false alarms.

The amplitude standard deviation (ASD) equation used for data post-processing in Matlab is shown in (3). Sample indices are represented by *i*, and *N* is the number of samples per window.

$$\sqrt{\frac{\sum_{i=0}^{N-1} x_i^2}{N} - \mu^2} \qquad (3)$$

2.3.2. STANDARD DEVIATION OF SIGNAL ENERGY

The standard deviation of signal amplitude captures most of the movement information, but not all. Signal spectrograms indicated that signal energy increased when movement happened in testing. In the time domain, signal energy displayed similar characteristics to signal amplitude when movement occurred in testing.

The standard deviation of signal energy and signal amplitude capture slightly different information, so it was deemed beneficial to include both as metrics for intrusion detection. What one metric might not capture, another potentially could. To calculate energy standard deviation (ESD), all the values in the window are simply squared, and the ASD calculation is repeated:

$$\sqrt{\frac{\sum_{i=0}^{N-1} (x_i^2)^2}{N} - \mu^2} \qquad (4)$$

where μ is now defined as the mean of the signal energy.

$$\mu = \frac{\sum_{i=0}^{N-1} x_i^2}{N} \qquad (5)$$

Notice that this is the same summation that occurs in equation 3 for ASD. When running post-processing or real-time processing, the ASD calculation already contains values for the ESD calculation, saving valuable processing time.

2.3.3. AVERAGE LEVEL

A third metric is the average level of the signal. This is a valuable metric for determining environmental changes even when nothing significant appears from the ASD or ESD. The AGC data gathered shows that in different positions, the gain of the AGC will shift to different levels and remain there while the target is still there. Conveniently, the average level μ is already calculated in the ASD calculation.

2.3.4 MATLAB RESULTS

The equations and logic for the detection metrics were first implemented in Matlab in post-processing. For ease of visualization, a highlighter program was created. The program took three input files, the first is used as quiescent data, and other two files are used as comparison tests. The tests performed to get quiescent data required starting the test in the lab and walking out of the building. For this reason, the option of setting bounds was included in the Matlab script to isolate the true quiescent data. This option also served to isolate good data from longer tests.

The script opened the first data file and ran the moving window across the data, getting the maximums for the ASD, ESD, and the average minimums and average maximums for the level. A variable was defined for the number of segments the quiescent data should be broken up into. By using the sample rate of the data and the window length, the number of segments calculated implicitly defined the amount of overlap between each calculation. This process established the foundation for the arming phase of the intrusion detection prototype.

After the arming process takes place, the script looks at the other two data files. For the first set of data, the program finds where the ASD or ESD (only one or the other could be selected) exceeds the respective maximum value calculated in the arming process. For each window location where this is the case, the program plots the same data on top of the original data in a different color, indicating a detection.

Figure 6 shows a combination of ASD/ESD detection and level detection. If the window calculations for the current data indicate that the average level extends beyond the average minimum or maximum level of the quiescent data (indicated by the red lines), the current data is also highlighted a different color than for ASD or ESD detection. In figure 6, ASD/ESD detections are highlighted in purple, and level detections are blue. The original signal is green. Here, the original data gets mostly covered by the detection metric highlighter.

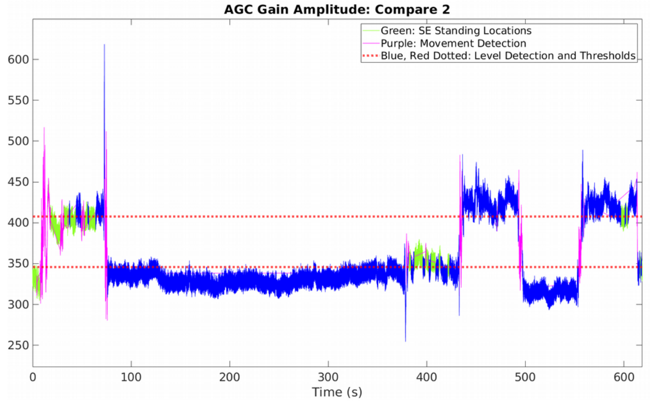


Figure 6: Detection highlighter example for the SE antenna

3. gr-RID

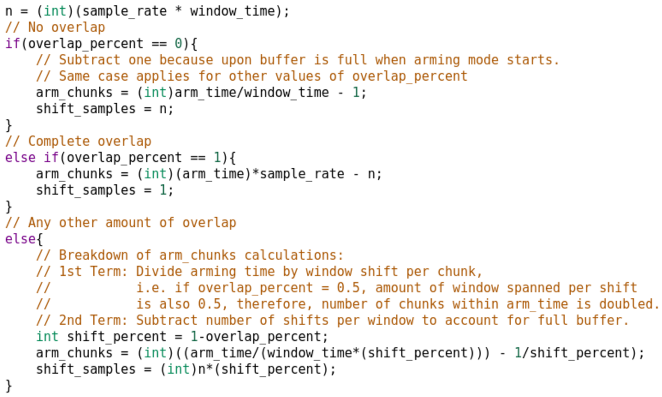
Initial testing with the custom AGC and data processing in Matlab for each detection metric provided an algorithmic basis for a full real-time detection system. All these functions were developed into a new GNU Radio sync block, written in C++. This block is known as RID (Real-time Intrusion Detection).

There are three components or phases the RID block utilizes. The first is initialization, in which an internal buffer is created to store data and run calculations. Variables related to detection thresholds are also calculated. The second phase is to take input data in a quiescent state and “arm” the block using the quiescent data. The third phase is detection mode. In this state, the block periodically evaluates data for the detection metrics to determine if there has been an intrusion.

3.1. Block Setup

When the RID block is initialized, several variables are used to set the internal buffer size, calculate how many iterations are required for arming the block, and set how many samples should be collected before processing data again. These variables are the sample rate, window time, arming time, and the overlap percent. Of these variables, only the overlap is run-time adjustable.

The internal buffer is created by simply multiplying the sample rate by the window time. After the internal buffer size is determined, calculations are done to determine how many times the block needs to call its arming sequence before the block is considered to be armed. This is based on the arming time and the overlap percentage. The amount of overlap can range from 0 to 100 percent. Zero overlap means the block will process data for every *n* samples, and 100 percent overlap means the block will process the window with every new sample obtained. Based on conditional statements shown in figure 7, the number of samples between window processing is calculated and stored in the *shift_samples* variable. Here, the number of iterations of the arming phase is also calculated.



```
n = (int)(sample_rate * window_time);
// No overlap
if(overlap_percent == 0){
    // Subtract one because upon buffer is full when arming mode starts.
    // Same case applies for other values of overlap_percent
    arm_chunks = (int)arm_time/window_time - 1;
    shift_samples = n;
}
// Complete overlap
else if(overlap_percent == 1){
    arm_chunks = (int)(arm_time)*sample_rate - n;
    shift_samples = 1;
}
// Any other amount of overlap
else{
    // Breakdown of arm_chunks calculations:
    // 1st Term: Divide arming time by window shift per chunk,
    //           i.e. if overlap_percent = 0.5, amount of window spanned per shift
    //           is also 0.5, therefore, number of chunks within arm_time is doubled.
    // 2nd Term: Subtract number of shifts per window to account for full buffer.
    int shift_percent = 1-overlap_percent;
    arm_chunks = (int)((arm_time/(window_time*(shift_percent))) - 1/shift_percent);
    shift_samples = (int)n*(shift_percent);
}
```

Figure 7: RID constructor calculations for arming and detection phases

The calculations done in the constructor are set up such that when the full system is run, the *arm_time* and *window_time* variables would be in units of seconds. Later testing revealed that the precision of these variables relative to real-time is poor, likely due to lack of clock cycles per second for processing a whole iteration during the arming phase.

After initialization, the block waits to activate until a *start_test* variable becomes true. The *start_test* variable may be implemented by using a Qt GUI Checkbox in GNU Radio to give the user control of when the block begins the arming phase. All outputs of the RID block are set to -1 until *start_test* is true. This allows for easy visual confirmation that the block has begun to function when running tests.

3.2. Arming Phase

In the arming phase, the *arm_chunks* variable shown in figure 7 defines how many times the arming section of the code will be accessed. The arming code gets accessed every time *shift_samples* samples have been obtained from the custom AGC in the GNU Radio flowgraph. In this section of code, the block implements the formulas shown in section 2.3 to calculate the thresholds for the detection metrics.

During arming, the maximum and minimum values are accumulated for level detection, and the maximum ASD and ESD are stored. After the calculations are performed *arm_chunks* times, the accumulated level detection variables are averaged and stored. Tolerances are introduced here, which are defined with tolerance percentages set by the user.

The tolerance for level detection is set by multiplying the level’s tolerance percentage by the difference between the average maximum and average minimum level. The ASD and ESD tolerances are determined by multiplying the respective tolerance percentage by the maximum ASD and ESD values. All tolerances are added to the original value from which they were calculated and results are used as the final thresholds.

Finally, the block prints out a message to the console indicating that it is armed. At this point, if the *start_test* variable become false, all the quiescent values and tolerances that were stored are cleared, and the block will again wait to do any processing until *start_test* becomes true.

3.2. Detection Mode

When the arming phase has finished, the block continues to run calculations and the results are compared to the quiescent thresholds. If the current measurements exceed the thresholds of any metric, the output value for the respective metric will be changed to one, indicating a detection. Otherwise each output will be set to zero. Each metric has a run-time adjustable tolerance percentage that can be used to tweak the detection sensitivity if necessary.

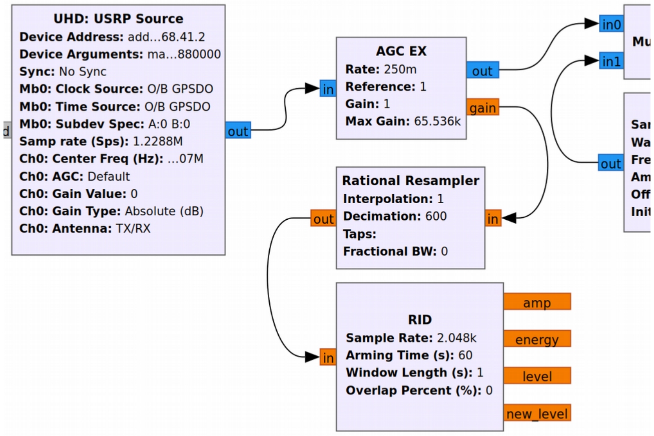
Comparison logic for the standard deviation metrics for signal amplitude and energy is simple. If the standard deviation is greater than the standard deviation threshold, indicate a detection. For level detection, if the average level of the current data is greater than the average maximum level threshold, a detection is triggered. Similarly, a detection is triggered if the current average level drops below the average minimum level threshold.

A final feature of the RID block is sound output for when detection happens. There are three conditions for an alarm. One condition is for when both ASD and ESD detections are triggered, one is for when either ASD or ESD is triggered, and the last is for when only the level metric is triggered. In the first case for an alarm, a detection is far more likely to be correct because two metrics are indicating a presence. This aids in reducing the chance of a false alarm. The sound output is run-time adjustable, which is useful for when multiple receivers are being used at the same time. All sound playback was implemented using the *system()* C++ command to call the *canberra-gtk-play* terminal command in Linux.

4. Intrusion Detection System Architecture

The intrusion detection system prototype consists of a transmitter and up to four receivers. The multi-receiver system is wrapped up into a hierarchical block written in Python, composed of duplicates of a base flowgraph. The full intrusion detection system uses a hierarchical block that combines up to four copies of a receiver section similar to the one from figure 2. The transmitter portion remains the same. For intrusion detection, additions were made to the receiver side of the flowgraph, including the custom AGC with gain output and the RID block, as depicted in figure 8. For the purposes of testing RID on non-AGC data ports in the system, the post-AGC blocks remain available.

As an aside, the RID block in figure 8 has an additional output port called *new_level*, but that is a feature that is still being developed. It could potentially be used as a method of intruder localization, but it is not necessary for intrusion detection as described in this paper, and thus is not explained here.



UHD: USRP Source
Device Address: add...68.41.2
Device Arguments: ma...880000
Sync: No Sync
Mb0: Clock Source: O/B GPSDO
Mb0: Time Source: O/B GPSDO
Mb0: Subdev Spec: A:0 B:0
Samp rate (Sps): 1.2288M
Ch0: Center Freq (Hz): ...07M
Ch0: AGC: Default
Ch0: Gain Value: 0
Ch0: Gain Type: Absolute (dB)
Ch0: Antenna: TX/RX

AGC EX
Rate: 250m
Reference: 1
Gain: 1
Max Gain: 65.536k

Rational Resampler
Interpolation: 1
Decimation: 600
Taps:
Fractional BW: 0

RID
Sample Rate: 2.048k
Arming Time (s): 60
Window Length (s): 1
Overlap Percent (%): 0

Figure 8: Receiver section of flowgraph with intrusion detection blocks added.

5. Testing and Results

The simple tests conducted with the intrusion detection capability have shown very encouraging results. The test described here was performed per-receiver for the NC, SC, NE, and SE antennae. The SW antenna was used as the transmitter in all cases. The aim of the test was to determine what areas are covered by each receiver’s detection capabilities.

The test for intrusion detection was performed at a medium-speed walking pace by an adult. Audio feedback was enabled for one receiver at a time to determine per-receiver detection ranges. The tester walked around the hallways of the building and through the rooms in the center as shown in figure 9. As much space was covered as possible. Bluetooth headphones were connected to the test computer to allow the tester to receive auditory detection feedback while performing the tests. The test for each receiver was performed when the tester was the only person on the 2nd floor of the building so as to prevent erroneous data during the arming and detection phases of the RID block. During the arming phase, the tester could not leave the building, so instead remained still at the testing computer for arming duration.

Variables such as transmitter and receiver gain were adjusted such that the receiver blocks could achieve stable lock in both PLLs and symbol timing synchronizers. This was done as a visual confirmation of consistency between tests. The transmitter gain was set to 65 dB (absolute), and the gain values for the receivers were set to 65, 65, 65, and 51 dB (absolute) for the NE, SE, NC, and SC receivers respectively. The AGC rate was set to 0.25. Other AGC values were left at default values established in the original GNU Radio AGC. The tolerance percentages for ASD, ESD, and level detection are listed in table 1. The random data was transmitted at a frequency of 905 MHz.

Table 1: Detection metric tolerances

Receiver	ASD Tolerance (%)	ESD Tolerance (%)	Level Tolerance (%)
NE	75	75	30
SE	10	10	25
NC	15	5	5
SC	15	15	5

While testing, a printed, blank version of figure 9 was used to record where an alarm was activated. Detectable areas were marked for any of the three detection metrics. Figure 9 shows the results for the SC antenna. The NE, SE, and NC antennae covered a lot of similar area between each other and thus are shown here. Figure 10 is a heat map of the combined cover for the floor of the lab building. Testing showed that all locations have at least two antennae that will react to walking motion.

Some of the receivers were more sensitive to different metrics. The sensitivity was such that the tolerances were adjusted to remove any erroneous detection indications. This was possible to do given that the tester was the only person on the test floor. After tuning the tolerance percentages for each receiver, detections were accurate and reliable as long as there was only one person in the testing environment. Tolerance tuning also served to reduce random hardware-induced detection from any hardware noise, which seemed to be prevalent in an overnight test run before the full test.

A pattern emerged when post-processing data with Matlab that was again valid in these real-time tests. For our specific setup, it appears that the sensitivity of a given receiver is largely determined by two parameters: distance from transmitter location, and how indirect the signal path is. The NE antenna is the farthest away from the transmitter and was able to detect motion around the entire testing area. The SC antenna is significantly closer to the transmitter and could detect just under half of the motion in the testing area.

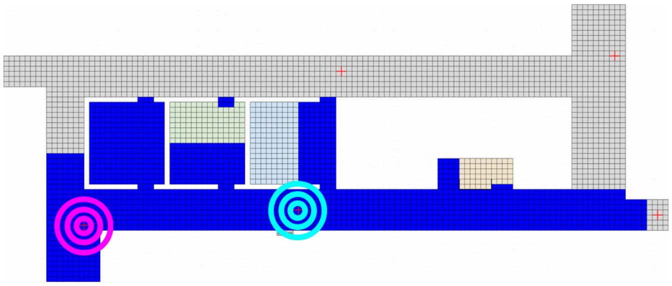


Figure 9: Detection Range for SE antenna during walking test. Detection area is highlighted in dark blue. The transmitter is centered in the purple rings and the receiver is centered in cyan rings. Each square represents a square foot.

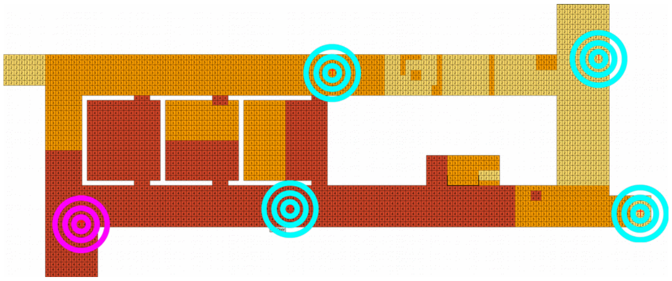


Figure 10: Heat map of the lab building for walking test of all four receivers. Darker colors indicate areas covered by more receivers.

6. Conclusion and Further Research

The intrusion detection system was designed to be a viable and sensitive method of detecting intruders in a real-world test environment using GNU Radio. Early testing showed some metrics that could be processed to identify the presence of an intruder based on detection theory concepts. These metrics were successfully integrated into a real-time GNU Radio block as a component of a QPSK communication system. The system is able to provide real-time feedback about intrusion using real-time hardware.

The intrusion detection system is highly tunable, allowing for generalization to other situations and building layouts than were used in our testing. As long as the state of the test environment is known and detection metric tolerances, transmit/receive gain, and AGC rates are tuned correctly, the intrusion detection system can provide very good results under basic testing conditions.

In further research, we aim to learn more about the capability of the system in other testing conditions. We also seek to implement the same process of detection by using Wi-Fi signals within GNU Radio. Another possible research route is target localization based on the level detection metric. This may be combined with other work from this lab, which involves identifying and processing Doppler-shifted radio signals for motion tracking. Motion detection and the potential for localization with this project may enhance the tracking capability of the motion tracking concept.

References

Bradshaw, Thomas L., "Alternative Doppler Extraction for Indoor Communication Signals" (2021). *All Graduate Theses and Dissertations*. 8111. https://digitalcommons.usu.edu/etd/8111 https://doi.org/10.26076/82c8-42ac

Rice, Michael. (2009). *Digital Communications : A Discrete-Time Approach* . Upper Saddle River: Pearson.

Wang, T., Yang, D., Zhang, S., Wu, Y., & Xu, S. (2019). Wi-Alarm: Low-Cost Passive Intrusion Detection Using WiFi. Sensors (Basel, Switzerland), 19(10), 2335. https://doi.org/10.3390/s19102335

Z. Tian, Y. Li, M. Zhou and Z. Li, "WiFi-Based Adaptive Indoor Passive Intrusion Detection," 2018 IEEE 23rd International Conference on Digital Signal Processing (DSP), 2018, pp. 1-5, doi: 10.1109/ICDSP.2018.8631613.