College of Engineering
UtahStateUniversity

Daren Swasey

A01962435

ECE 6250

Rincon Research Corporation

Signed by: Mike Garcia
Chief Engineer, Advanced Systems
Rincon Research Corporation

[illegible]

RINCON RESEARCH CORPORATION INTERNSHIP REPORT

The following pages describe the internship at Rincon Research Corporation held from June to July 2022. The internship was a team-based project. There were six members of the team. The team members brought skills to the table like web development, deep learning and neural network development, information technology, statistical signal processing, digital communications, real-time system development, and system integration. This report provides information about the project, its development, and details of the real-time implementation of the project.

Rincon Research has requested that only high-level details be described outside of Rincon Research as per the non-disclosure agreement signed at the beginning of the internship. System parameters, algorithm implementations, and similar project specifics will not be included in this report. Certain tools used and methods of development are described in place of these details.

ATOMIC Locator Overview

The premise of the project was based on previous work done by a Rincon employee in 2021 in a project called MLGeo. MLGeo used machine learning techniques to develop a system that could locate a radio transmitter in an outdoor environment with a multi-receiver array. That project used a pre-determined input signal to the transmitter and relied on correlation outputs from processing to aid a neural network in the estimation of target location. Many of the techniques used in the MLGeo project were adapted for use in the ATOMIC (AuTOmatic Machine-learning Indoor Correlation Locator) project.

The ATOMIC Locator system performs indoor geolocation of a target transmitting a pre-determined radio signal. The system is shown in figure 1 in the appendix. The flow of the system is explained here.

A pseudo-noise (PN) sequence is transmitted through an Ettus Research B210 radio using the GNU Radio environment, an open-source environment for developing radio simulations, systems, and applications. The PN sequence is modulated using a BPSK constellation at a carrier frequency chosen based on testing. The signal travels through the space on the first floor of the Rincon Research building and in the process, bouncing around inside before arriving at one of two receiver radios, which are Ettus Research B200-minis.

While the signal is being transmitted inside the building, it is reflected off a variety of surfaces, causing attenuation, time-delays, and phase shifts. This new information present in the signal can vary across different locations on the first floor of the building. Some of the variations in the signal can be captured by doing a correlation of a signal at a receiver radio with the originally transmitted PN sequence. Autocorrelations of PN sequences produce a distinct spike at zero-lag. If a PN sequence is sent through a radio channel with multi-path effects, the cross-correlation may have a similar peak at zero-lag, but it also includes other peaks at different amplitudes, spread out behind the initial peak. This is due to the delays induced by the multi-path effects on the signal.

These cross-correlation spreads vary between different locations in the building. The neural network in the ATOMIC Locator system has been trained on over 300 locations around the first floor of Rincon Research. Each location is representative of a square meter on the floor, identifiable using Cartesian coordinates. The trained neural network can be given new data as an input and output a vector of numbers that represent the probabilities in each training location that a radio transmitter is there.

The probability map output from the neural network is then post-processed with a grid-based particle filter. The grid-based particle filter is derived similarly to a Kalman filter, and is a special case designed for filtering on discretized points in a grid. While the neural network performs very well during training, the probability maps that are output from the neural network have very high variance. The grid filter serves to introduce "intuition" into the system by using a filtering kernel that is designed to represent the probability that a human (with a large cart and a transmitter radio) will move into any of the adjacent positions in a 3x3 meter grid. The grid filter significantly reduces the variance of the probability maps, giving a more precise and accurate estimation of a target transmitter's location.

The raw and filtered estimations from the neural network are output as matrices. These matrices are then converted to dictionaries or maps that contain the probability estimation and the Cartesian coordinates for each point that was trained on. These dictionaries can then be stored as files. A web server running on the machine with the data files will then convert the information in the files to a heat-map display, overlaid on top of a map of the first floor of Rincon. All the grid points that data was collected for are accented using color contrast.

The ATOMIC Locator can operate in a static mode, where each step is performed individually. First, data is collected and stored in a filesystem. From there, all the operations required to get good correlation data output are performed with a Python script, which also formats the data for the neural network. This information is stored in another part of the filesystem. The neural network can then read those files and process all the information, generating another file or set of files that has all the unfiltered probability maps of the transmitter location. Another Python script can then read the file and do grid filtering. After all of this, a final output file is generated, which can be read by the web server and visualized in a web browser, and each probability map can be accessed simultaneously.

This paper discusses some general challenges in developing the system, then describes the real-time operation of the system in greater detail.

Experiment Planning and Setup

One of the most critical components of the project was establishing a well-thought-out plan as to how to achieve the goals established by the project supervisors. This plan needed to include details such as signal type, radio locations, sample rates, data structure, data type, and data flow. Establishing the supporting hardware, software, and algorithms required to enable indoor geolocation took up much of the allotted internship time of eight weeks. Creating a solid foundation from which to collect data, perform system testing, and do data analysis required much of those very things, only on a smaller scale. This portion of the internship required a significant amount of small-scale testing, analysis and evaluation, and design iterations before a capable indoor geolocation machine learning system could be properly developed. Some of those considerations and tests are described below.

Choosing a Signal

The first system parameter that was decided was the type of signal to be used. Due to multi-path reflections in an indoor environment, the transmitted signal needs to be robust enough to be detected even in the presence of multi-path contributions to the received signal.

One way to generate a robust, detectable signal is to use a pseudo-noise sequence (PN sequence). A PN sequence is a random number sequence designed to look as much like statistical white noise as possible. A white noise signal autocorrelation produces a magnitude of zero at all correlation lags except for at a lag of zero (Smith 2011). Because a PN sequence approximates white noise, its autocorrelation produces a similar result.

The PN sequences are generated by using the linear-feedback shift-register (LFSR). When a LFSR is seeded with the correct starting set of bits in each of its register positions, it is possible to generate a sequence of numbers that do not repeat themselves for a length of 2^n, where n is the length of the LFSR. Longer PN sequences look more like statistical white-noise, and thus when correlated with themselves provide a more distinct correlation peak.

Knowing this, the PN sequence can be used as a transmitted signal to provide excellent resistance against interference, but more importantly, to capture more detailed information about the environment. Transmitting a PN sequence is a method of Digital Spread Spectrum (DSS) communication. Because a PN sequence looks like white noise, it has energy distributed across a large band in the frequency spectrum. Transmitting a signal across a large bandwidth implies that there could be frequency responses across that whole bandwidth as well. Depending on properties of a particular indoor environment, varying frequencies will provide varying information that comes in the form of phase and magnitude changes to the signal. Any information present in the return signal from a transmitted signal is a potential data feature for a neural network to learn.

Due to these several properties of a PN sequence, it is a robust way both to transmit information from the signal itself and to collect information from the surroundings as the signal is attenuated and phase-rotated from reflections. This signal is the foundation for the work that was done in this project.

Setting System Parameters

To get the best results from a neural network for indoor geolocation, signals from different locations must be distinguishable from each other. To maximize distinguishability, there are several parameters to consider: sample rate, PN sequence length, signal modulation and upsampling, excess bandwidth, RX radio placement and number of RX radios, and update timing.

Sample Rate and PN Sequence

One of the first requirements in system development was determining the signal type and the sample rate. These two parameters affect one another in crucial ways, requiring an iterative implementation and testing process to narrow down the options and select a final value for either one.

The PN sequence the team settled on was generated using the *GLFSR Source* block in GNU Radio. Using this method of generating a PN sequences was extremely convenient and used throughout the early stages of testing to find the optimal sequence length. The optimal sequence length is a length for which at every location the neural network needs to identify, there are distinct features in the neural network's input signal. The performance of a PN sequence is also strongly linked to sample rate. Higher sample rates mean that the time to transmit a full PN sequence is shorter.

After PN sequences of different lengths were generated and modulated in a simple BPSK constellation, experiments were performed to determine what sequence length would be best to work with. Testing involved many repetitions of the same process. The modulated PN sequence was sent over a chosen carrier frequency then received by a receiver in the same room. The collected data was then stored to a file. The data was sent to a computer that was used to do processing on the initial test data. The correlations were produced and plotted.

Initial results showed what looked like noise. After manipulating the data, it was found that offset in carrier frequency (CFO) was causing the problem. A correction to the problem was made, and testing continued.

Testing progressed until the correlation peaks were at a consistent magnitude, and the shape of the correlation seemed to indicate enough multi-path information to be valuable for a neural network.

Modulation, Upsampling, and Excess Bandwidth

Modulation, upsampling, and excess bandwidth for a root-raised cosine filter are all connected to bandwidth. Bandwidth is a valuable resource for the purposes of receiving better multi-path information in correlations. An indoor environment has different responses to different frequencies. Increased bandwidth in a transmitted signal means that more of the potential frequency response of the building can be collected in the received data. If this is the case, better multi-path information could be collected from the radios. The parameters listed above all affect the bandwidth.

Using the BPSK signal constellation increases bandwidth because for any transition between constellation points requires a 180° rotation in the IQ plane. The BPSK symbols were sampled at two samples per symbol, decreasing the smoothness of the resulting signal and increasing the bandwidth even further. The excess bandwidth for the root-raised cosine filter was somewhat of limiting factor. The bandwidths achievable with the system parameters were limited both by the available bandwidth around the center frequency used and the sample rate of the radios. Setting an excess bandwidth too high would produce unwanted aliasing and would negatively affect the resulting correlations. The excess bandwidth for the GNU Radio blocks was set to be low to prevent the aliasing problem.

Radio Placement and Number of Radios

Placing the radio receivers in a “good” spot could make a significant difference on the multi-path responses seen in the correlations. Knowing where to put them would change the multi-path information received, as well as the signal amplitude at a given location. The first tests regarding radio placement were associated with distance. After the system was more developed, testing was repeated for the purpose of determining possibly better placements that the ones originally chosen for the final system.

The first tests done were range tests. By putting a transmitter in location *x* and a receiver in location *y*, transmitted PN sequences could be collected and processed for good correlations. If correlation still showed valuable information at long-range, then the receivers could be placed farther apart, in theory.

The second set of experiments run were performed after the system behavior was identified in response to the training data that was collected. The receiver radios were far apart, and the power of the signal at such ranges was not conducive to accurate neural network output in some location in the Rincon building. Many locations throughout the building gave correlations that looked very similar to correlations from other parts of the building. This resulted in noticeable ambiguity in the neural network output.

To resolve the ambiguity, one of the receiver radios was moved closer to the area that had high ambiguity. Testing simply involved collecting more data in the new arrangement, then generating correlations and feeding those to the neural network again, making a new neural network model. The results of processing the collected data indicated that more receivers would have improved overall system performance by giving additional correlations that were distinct from others.

Update Rate

This project aims to specify the location of a radio transmitter attached to a human being. This means the system that identifies position must account for the motion model of a human being. During planning, it was decided that in a real-time situation, the target would be pushing a large cart around with the radio transmitter on the top of it. Heuristically, this cart will never be moving very fast, neither will it be changing directions very quickly. Therefore, a good estimate (possibly even an overestimate) of how many updates per second would be reasonable in the tracking system was determined to be anywhere between 1 and 10 Hz.

Real-Time Operation of the ATOMIC Locator

Each step in the real-time architecture of the ATOMIC Locator project presents unique challenges and introduces constraints on the overall ability of the system. The first process, which involves the GNU Radio ecosystem and interacting with Ettus Research USRP B2XX devices was critical to enable consistent data flow and produce reliable correlations. The second major part of the real-time system is the variety of inter-process communication methods. A third topic of discussion is the metrics by which the performance of the system is measured.

The major goals for the real-time system performance were to achieve an update rate of 5 Hz, with minimal time differences between correlations, and to keep start-to-end latency below 200 ms. The value for latency is based on the value of the update rate, so it is more important to know why the update rate was chosen.

To test real-time operation, a team member moved the transmitter around using a sizable cart. Mobility is very limited while using the cart, and it is not expected that the cart will be pushed very fast. An update rate of 5 Hz is good enough to get updated information about the transmitter location without losing any continuity between measurements. Slower update rates may introduce strange jumps between different locations that seem disjoint, and faster update rates may just be too demanding for the system and/or be unnecessary for the type of target transmitter being located.

The value chosen for the latency naturally follows in the footsteps of the update rate requirement. For an update rate of 5 Hz, all the real-time processing must be able to be completed within 200 ms, or else the system cannot truly operate in real-time. The methods for and obstacles to achieving these goals are explained in future sections.

GNU Radio Ecosystem and Radios

The sample rate was the first thing that was tested. Because of the close-quarters nature of an indoor environment, the multi-path signals only happen for a short duration. Higher sample rates provide a greater number of samples in a correlation. The more information present in a correlation, the better the neural network may be able to distinguish the correlations that appear from different locations. The radios used to achieve a sufficient sample rate were the B200mini and the B210.

The hardware in the radios can easily achieve sample rates upwards of 50 MHz. While this is possible even in a radio as small as the B200-mini, the real limitation in data bandwidth was on the machine hosting the radio. These host devices simply were not powerful enough to do I/O operations and maintain the sample rate that was chosen. This lack of computing power is indicated during runtime by the radios. If there is a problem in the data stream (regardless of which end it is on) and the radio cannot push data through to its host device fast enough, the radio may drop samples.

If this is the case, a meta-data message is passed into the data stream. The type of data stream issue is indicated by one of a few letters. "O"s indicate overflows, when there is too much data flowing into a system component for it to handle at once. "D"s indicate dropped data packets, which is essentially the same problem as overflows. Lastly, "U"s indicate underflow, when there is not enough data coming into the radio for the radio to output data at the selected sample rate.

Early on in testing, it was noted that several "O"s and "D"s appear when using the selected sample rate. The effects in the data stream of overflow and dropouts can be serious, but the issue was not seriously considered until later in the development process. Fortunately, the effects of the overflows and dropouts were negligible. Several methods of testing the severity of this issue were devised.

Simulation of the results of overflow and dropped data indicated that correlations would be of very poor quality, usually looking like nothing more than noise, even if only a few samples are dropped. Because a PN sequence looks like statistical white noise, if a part of it is missing and the rest of the sequence is shifted over, all the good signal properties obtained by using a PN sequence are nullified. If the overflows in the system were common enough, it would destroy some of the utility of the transmitted signal.

Because the sample rate requirement was too high for the RX host machines to process all incoming data, having overflows and dropped data is guaranteed during real-time system operation. If the rate of overflow and dropped samples is low compared to the number of samples being received, most of the correlations that might be attempted in each data set will all be valid, leaving only a few that may not be valuable for use.

Testing Overflow

Because overflow was a real cause for concern, more rigorous testing methods were required. Attempts at using benchmark testing scripts provided by Ettus Research proved futile because the hardware the benchmark was performed on was not under the same processing load as it was during real-time testing. Online searches for other methods of how to detect and record the effects of overflow led to the discovery and use of the *Tag Debug* block in GNU Radio.

The *Tag Debug* block is a simple block that reads metadata from the GNU Radio data stream and prints the metadata to the console during runtime. The information contained in the *Tag Debug* messages was simple, but informative. For testing, the *Tag Debug* block was attached directly to the output of one of the B200-mini radios. The B200-mini radio outputs stream metadata every time one of the *'O'*s, *'D'*s, or *'U'*s appears in the console. From these messages, there are two key pieces of information necessary to determine how many dropped samples happened in a given amount of time. The printed information looks like the line below.

Offset: 15784235 Source: rx2_host   Key: rx_time   Value: 10.0731

Other values are also printed, but the most important piece of information is the *rx_time* tag. This tag specifies the time that has passed (in seconds) since the radio started its internal clock. If the sample rate is also known, then the number of expected samples can be calculated. The second important piece of information is the *offset* value. The offset indicates the number of items or samples that have been pushed through the radio until this point.

Given the selected sample rate for the system, testing showed that over the span of time required to transmit about one-billion samples, the drop rate for the real-time system was about 13%. Based on the operation of the real-time system that is explained later, this means that up to 13% of the correlations could be erroneous and result in poor location estimates from the neural network. Given the parameters used in the final system, a 13% drop rate still gives plenty of room to work with without corrupting too much information.

Real-Time Processing Block

The real-time processing block is a custom GNU Radio block written in Python. This block is used as a wrapper for all the processes necessary for creating the correlations that will be used in the rest of the system. This is the first section of the real-time system that requires the tracking and use of timestamps. These timestamps were critical in ensuring that the form of the real-time data would be as similar as possible to the data format used when training data was being collected and processed. The limits associated with this time alignment are also discussed.

<u>NTP and Timestamping</u>

Depending on the methods of signal processing, timing can make or break a system. The use NTP (Network Time Protocol) was critical in the development and use of the real-time system. NTP synchronization was available and active via the unclassified network provided during the internship at Rincon Research. This capability was used to provide training data to the neural network that was synchronized down to the millisecond. This means that every small motion of the cart during the process of collecting training data would also have been captured simultaneously by both receivers. This time synchronization of data is useful in the data collection situation, but it is far more important during real-time operation when there is significantly more movement from the transmitter.

If the real-time system is not synchronized, then data sent to the neural network could correspond to two different physical locations. The neural network does not expect to receive data with that sort of disassociation, so it will give bad output of the location of the transmitter. To overcome this problem in the real-time system, timestamps are used.

An algorithm was created to synchronized timestamps between independently operating receiver host machines. Other methods were considered but would have required the two receivers used to be connected over a network that was sometimes unreliable. Using tools available in Python, timestamps could be generated within milliseconds of each other and then accurately aligned after correlation data was produced in the GNU Radio ecosystem.

<u>Timing Tolerance</u>

In the GNU Radio ecosystem, every block in a flowgraph has a *work* function. This *work* function is called every time the block has enough input data to produce the type of output data specific to that block. Sometimes this is a $1:1$ input to output ratio, but often it can be a $N:1$, or even an $N:M$ ratio. Depending on the processing requirements of the *work* function, the *work* function may not be called as frequency as expected or may be called more often.

For this real-time system, correlations are not generated until there is a number of items equal to or greater than the length of the modulated PN sequence that is sent from the transmitter. Due to the nature of GNU Radio and the requirements to keep data flowing through the system, it takes several calls to the *work* function before a correlation can be produced. Because of this, the time at which a timestamp is generated is non-deterministic. A rather simple algorithm was implemented that aided the system in generating timestamps at approximately the same time, given a tolerance.

Initially the tolerance specified seemed to be working well but reducing the tolerance too much caused correlations to be skipped because the *work* function in the custom GNU Radio block was not called often enough. An experiment was to find a reasonable tolerance to use. The experiment was performed in the code for the data synchronization service.

Because of some pre-determined system parameters, the number of correlations to be calculated during a period of time was known. By testing for a specified amount of time, the amount of data that should come through is known. After testing with different tolerances, a number of correlations would come through that could be compared with the number expected. The ratio of correlation received to correlations expected varied with different tolerances for the correlation calculations. These ratios plateaued at a certain tolerance with high receive rates, so the tightest tolerance at that plateau was selected to be used in the final version of the real-time system.

Inter-Process Communication

The main method of communicating the data from one device to another was through a web-socket framework called *ZMQ*. As a package available in Python, *ZMQ* offers an easily accessible, high-level API, valuable in implementing a variety of communication paradigms between processes. The paradigms used are briefly described here. *ZMQ* allowed the data processing in the system to be split between processes and even across multiple devices, enabling multi-processing and reducing latency in the system.

Between the GNU Radio ecosystem and the data synchronization service that can be seen in figure 1 in the appendix, a PUBlisher/SUBscriber paradigm was used. With the GNU Radio flowgraph as the publisher, it could simply send correlation outputs directly to the synchronization service without the concern of whether the messages arrived or not. This allowed the GNU Radio environment to dedicate all its attention to the production of correlations.

Between the data synchronization service and the service that contained the neural network, grid filter, and file system, a *ZMQ* REQuest/REPly paradigm was used. This way, whenever the neural network and grid filter processing was complete, the newest set of time-aligned correlations could be requested for use. While processing in the neural network and grid filter take place, the data synchronization service can still receive and align incoming correlations.

Conclusion

The final version of the project was successful at performing the task of indoor geolocation in real-time. The limitations on the accuracy and performance come from a variety of factors that could be explored in much greater detail than time allowed for this project. The real-time system goals were met after many design iterations and optimizations to the processing methods, timing synchronization services, and data transfer mechanisms.

The selected parameters and code optimizations were done in such a way that minimized overflow and dropped samples from the receiver radios. Results showed that even with the dedicated correlation processing done in the real-time system, the percentage of dropped samples was only five percent higher than that of the static system, in which no processing was done on the raw data after collection.

The timing synchronization service allowed the receiver radios and respective processing blocks in GNU Radio to operate independently. The main factor in this capability was NTP synchronization.

The data transfer mechanisms were selected in such a way that the multi-processing capabilities of the processing hardware was utilized. Much of the rest of the system was distributed across different computers as well, spreading out the computational load of real-time operation.

After making the most of the hardware available in the limited time of the project, and eliminating bottlenecks to data flow, the goals for real-time performance were met, and in some cases exceeded expectations.

Bibliography

Nõmm, M., Kozel, D., &amp; Braun, M. (2016, December 6). Re: [Discuss-gnuradio] rx_time tag after drop. discuss-gnuradio. Retrieved August 3, 2022, from https://lists.gnu.org/archive/html/discuss-gnuradio/2016-12/msg00067.html

Overflow rectification for recorded GNU radio samples. Akademisk Radioklubb. (2018, September 5). Retrieved August 3, 2022, from https://www.la1k.no/2018/09/05/overflow-rectification-for-recorded-gnu-radio-samples/

Smith, Julius O., Spectral Audio Signal Processing, W3K Publishing, http://books.w3k.org/, ISBN 978-0-9745607-3-1.

APPENDIX

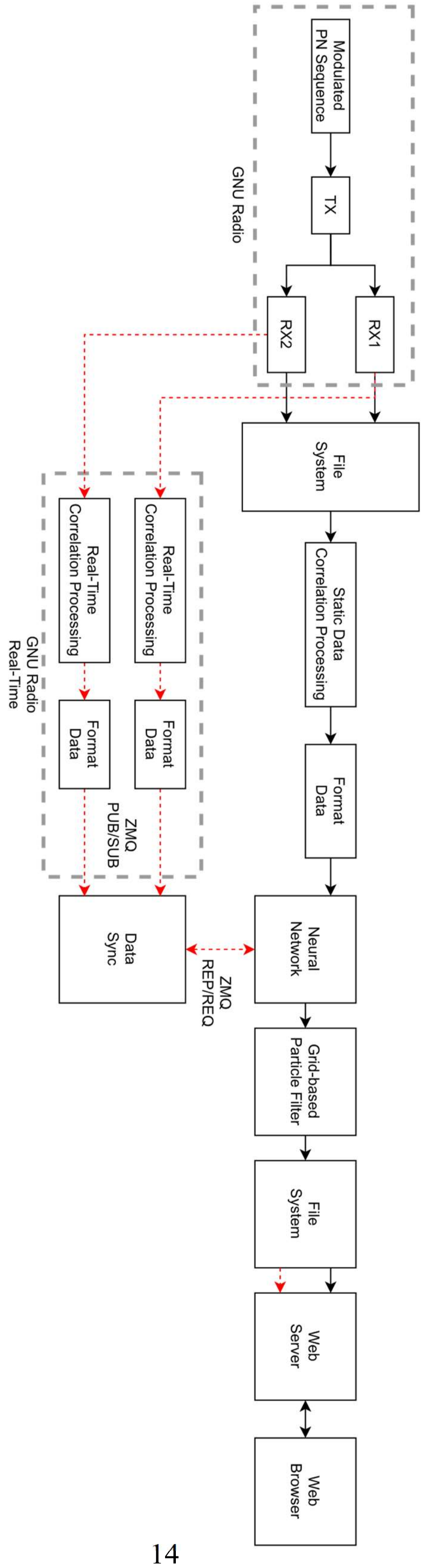

Figure 1. ATOMIC Locator System Diagram