U.S. Department of Transportation

Federal Railroad Administration

Railway Cognitive Radio to Enhance Safety, Security, and Performance of Positive Train Control

Office of Research and Development
Washington, DC 20590

DOT/FRA/ORD-13/09

Final Report
February 2013

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.			
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE February 2013	3. REPORT TYPE AND DATES COVERED Final Report	
4. TITLE AND SUBTITLE Railway Cognitive Radio to Enhance Safety, Security, and Performance of Positive Train Control			5. FUNDING NUMBERS
6. AUTHOR(S) Dr. Ashwin Amanna			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Virginia Tech ICTAS Building I, Suite 410 Blacksburg, VA 24061			8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) U.S. Department of Transportation Federal Railroad Administration Office of Research and Development 1200 New Jersey Avenue, SE Washington, DC 20590			10. SPONSORING/MONITORING AGENCY REPORT NUMBER DOT/FRA/ORD-13/09
11. SUPPLEMENTARY NOTES COTR: Jared Withers			
12a. DISTRIBUTION/AVAILABILITY STATEMENT This document is available to the public through the FRA Web site at http://www.fra.dot.gov.			12b. DISTRIBUTION CODE
13. ABSTRACT (Maximum 200 words)			
14. SUBJECT TERMS Positive Train Control (PTC), Software Defined Radio (SDR), Cognitive Radio (CR), wireless communications			15. NUMBER OF PAGES 71
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT

Robust and interoperable wireless communications are vital to Positive Train Control (PTC). The railway industry has started adopting software-defined radios (SDRs) for packet-data transmission. SDR systems realize previously fixed components as reconfigurable software. This project developed a railway cognitive radio (Rail-CR) which implements Artificial Intelligence (AI) decisionmaking in concert with an SDR to adapt to changing wireless conditions and learn from past experience. Objectives of the project included developing a concept of operations for wireless link adaptation based on use-case scenarios for packet radio systems, designing and implementing a decisionmaking architecture on an SDR, designing strategies for radio environment observations, defining operational objectives and performance metrics, and designing and exercising a test plan to demonstrate performance under varying conditions. The decisionmaking architecture of the Rail-CR begins with observations of the wireless environment and performance metrics. The architecture enables adaptation to new situations and the capability to learn from past decisions. The Rail-CR was tested under a variety of interference conditions designed to simulate real-world experiences. Results show that a radio operating with no-cognition was unable to mitigate interference conditions causing either significantly high errors or a loss of connectivity. The Cognitive Engine (CE) successfully overcame the interference by changing configurable parameters.

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18
298-102

METRIC/ENGLISH CONVERSION FACTORS

ENGLISH TO METRIC

LENGTH (APPROXIMATE)

1 inch (in)	=	2.5 centimeters (cm)
1 foot (ft)	=	30 centimeters (cm)
1 yard (yd)	=	0.9 meter (m)
1 mile (mi)	=	1.6 kilometers (km)

AREA (APPROXIMATE)

1 square inch (sq in, in^2)	=	6.5 square centimeters (cm^2)
1 square foot (sq ft, ft^2)	=	0.09 square meter (m^2)
1 square yard (sq yd, yd^2)	=	0.8 square meter (m^2)
1 square mile (sq mi, mi^2)	=	2.6 square kilometers (km^2)
1 acre = 0.4 hectare (he)	=	4,000 square meters (m^2)

MASS - WEIGHT (APPROXIMATE)

1 ounce (oz)	=	28 grams (gm)
1 pound (lb)	=	0.45 kilogram (kg)
1 short ton = 2,000 pounds (lb)	=	0.9 tonne (t)

VOLUME (APPROXIMATE)

1 teaspoon (tsp)	=	5 milliliters (ml)
1 tablespoon (tbsp)	=	15 milliliters (ml)
1 fluid ounce (fl oz)	=	30 milliliters (ml)
1 cup (c)	=	0.24 liter (l)
1 pint (pt)	=	0.47 liter (l)
1 quart (qt)	=	0.96 liter (l)
1 gallon (gal)	=	3.8 liters (l)
1 cubic foot (cu ft, ft^3)	=	0.03 cubic meter (m^3)
1 cubic yard (cu yd, yd^3)	=	0.76 cubic meter (m^3)

TEMPERATURE (EXACT)

[(x-32)(5/9)] °F = y °C

METRIC TO ENGLISH

LENGTH (APPROXIMATE)

1 millimeter (mm)	=	0.04 inch (in)
1 centimeter (cm)	=	0.4 inch (in)
1 meter (m)	=	3.3 feet (ft)
1 meter (m)	=	1.1 yards (yd)
1 kilometer (km)	=	0.6 mile (mi)

AREA (APPROXIMATE)

1 square centimeter (cm^2)	=	0.16 square inch (sq in, in^2)
1 square meter (m^2)	=	1.2 square yards (sq yd, yd^2)
1 square kilometer (km^2)	=	0.4 square mile (sq mi, mi^2)
10,000 square meters (m^2)	=	1 hectare (ha) = 2.5 acres

MASS - WEIGHT (APPROXIMATE)

1 gram (gm)	=	0.036 ounce (oz)
1 kilogram (kg)	=	2.2 pounds (lb)
1 tonne (t)	=	1,000 kilograms (kg)
	=	1.1 short tons

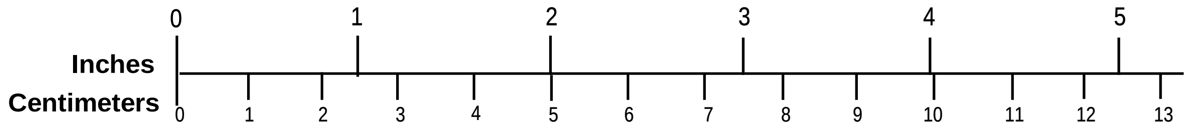
VOLUME (APPROXIMATE)

1 milliliter (ml)	=	0.03 fluid ounce (fl oz)
1 liter (l)	=	2.1 pints (pt)
1 liter (l)	=	1.06 quarts (qt)
1 liter (l)	=	0.26 gallon (gal)
1 cubic meter (m^3)	=	36 cubic feet (cu ft, ft^3)
1 cubic meter (m^3)	=	1.3 cubic yards (cu yd, yd^3)

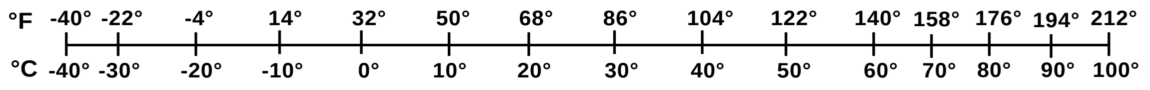
TEMPERATURE (EXACT)

[(9/5) y + 32] °C = x °F

QUICK INCH - CENTIMETER LENGTH CONVERSION



QUICK FAHRENHEIT - CELSIUS TEMPERATURE CONVERSIO



°F -40° -22° -4° 14° 32° 50° 68° 86° 104° 122° 140° 158° 176° 194° 212°
°C -40° -30° -20° -10° 0° 10° 20° 30° 40° 50° 60° 70° 80° 90° 100°

For more exact and or other conversion factors, see NIST Miscellaneous Publication 286, Units of Weights and Measures. Price $2.50 SD Catalog No. C13 10286

Updated 6/17/98

Contents

Figures

Tables

Executive Summary

Robust, reliable, and interoperable wireless communications are vital to the success of Positive Train Control (PTC) systems. The objective of this project is to demonstrate a railway-specific *cognitive radio* (Rail-CR) system that adds to the Federal Railroad Administration's (FRA) ongoing innovation in communications-based PTC. A Cognitive Radio (CR) will meet FRA needs by making railway wireless communications more secure, spectrally efficient, and less costly to deploy and maintain. The proposed application, based on near-ready technologies that use an innovative application of Artificial Intelligence (AI) to transmit information to a software defined radio (SDR) platform, is not currently available in the consumer market. The system achieves situational awareness and optimized performance by applying learning and decisionmaking algorithms to the basic functionality of the SDR, effectively augmenting the system's capability.

The outcomes of this project include a concept of operations (CONOPS) that defines the goals, objectives, and operational scenarios for a Rail-CR. This CONOPS provides the guiding philosophy for development and implementation of a Cognitive Engine (CE). Further contributions include a CE architecture for machine control of an SDR. This architecture combines both experiential and analytic decisionmaking relying on case-based reasoning (CBR) and Genetic Algorithms (GAs). The engine first observes the system performance and current SDR configurations. If performance falls below a predefined performance threshold, the CE proceeds to the next step of identifying a set of SDR configurations that will mitigate the situation. These observations are coded into a case-based representation and compared against past experiences. If the current situation matches a past experience within a defined similarity, then the solution associated with the past experience is implemented. However, if no past experience is similar enough, then the engine relies on a GA optimization routine to identify a potential solution. The GA's solution is then implemented, and the performance improvement is verified. If performance improves, then this solution and the associated situation are appended to the case base. This enables the engine to draw upon this decision at a later time and provides a learning capability.

This architecture was implemented on a reference open platform SDR in a modular fashion where the operations are scalable to new SDR control parameters and observable performance metrics. A tutorial on how to modify the engine is included.

Testing of the engine was performed in a laboratory and outdoor railroad environment. A data file transfer provided a repeatable load. As the data file was transmitted, the engine monitored performance metrics that included Packet Error Rate (PER), throughput, spectral efficiency, and goodput. When packet error increased above a specified threshold, the engine modified one or more control parameters that included transmit power, modulation, coding, or packet size. A third SDR link acted as an optional interference source to create wide-band interference, narrow-band interference, dynamically moving narrow-band interference, and an artificial fading environment.

Results show that a stock wireless link, with no cognition, was unable to maintain a connection under the interference scenarios because of high packet error. The CR was able to manipulate control parameters such that any degradation in throughput or increase in packet error was mitigated enough to maintain a connection.

1. Introduction

Robust, reliable, and interoperable wireless communications are vital to the success of PTC systems. PTC aims to improve on system safety and efficiency and help safeguard against human operator error by utilizing communication, computing, and related technologies. The performance of the underlying wireless communication system directly impacts the operation and effectiveness of PTC.

Railroads operate in a dynamic and complex environment. Given the dynamic, time-varying nature of the wireless communication channel and the large operating territory of railroads, train control systems encounter a challenging operating environment and require mechanisms to ensure reliable, robust, and efficient communication between the locomotive, waysides, and the back office. Maintaining connectivity, sustaining robust communication in the presence of high noise or interference, mitigating intentional jamming incidents, accessing higher bandwidth connection when available (Wi-Fi, WiMax, etc.), undergoing selection of the best base station, and scheduling multiple mobile entities to access the shared wireless channel are some challenges that may be encountered in the daily operations of railroads.

The objective of this project is to develop Rail-CR, a CR specific to railroads. To the best of our knowledge, this is the first effort in applying CR, an emerging technology, to support railroad operations. A CR will meet the needs of FRA by making railway wireless communications more interoperable, robust, secure, spectrally efficient, and less costly to deploy and maintain. The proposed application, based on near-ready technologies that use an innovative application of AI to transmit information to an SDR platform, is not yet available in the consumer market. The system achieves situational awareness and optimized performance by applying learning and decisionmaking algorithms to the basic functionality of the SDR, effectively augmenting the system's capability.

CR technology is very attractive for railroads, and its benefits have been demonstrated in areas that share similar performance requirements, including public safety, military, and commercial applications [1][2]. By leveraging software-radio technology and augmenting it with learning and decisionmaking algorithms, we enable a radio link that can adapt to new situations, thereby improving the performance of the wireless communication system. A CR can help alleviate performance degradation that may arise because of interoperability issues, an overcrowded wireless spectrum, jamming or interference, and high environmental noise. Furthermore, a spectrum has been identified as a scarce commodity, and railroads have been identified as key entities with high-priority access to electromagnetic spectrum. This further highlights the need in railroads and PTC for efficient spectrum usage and robust communications, areas where CR technology has been shown to have tremendous benefits.

Traditionally, the radios being used for voice and data communications have been fixed-functionality radios. Today, more and more industries have adopted SDRs because the functionality and operating parameters can be controlled by software and reconfigured in the field even after deployment. SDRs allow for real-time control of radio operating parameters and also provide access to certain observable parameters or measurements that may help determine state information and communication link performance. Examples of observables include PER, throughput, received signal power, and even Global Positioning System (GPS) location information.

By augmenting the SDR with learning and decisionmaking algorithms, CR technology enables the radio to learn from its past experiences and adapt its behavior to the operating environment.

CR applies principles from the field of human cognition, AI, and optimization theory to wireless communication systems. Over the past few years, CR research has spanned several areas—the development of CEs (decisionmaking and learning processes), environmental parameter extraction (e.g., signal detection, classification, channel characterization, and co-channel interference estimation), cognitive networking concepts, software architecture, CR analysis and design techniques based on game theory, and CR test-bed development. Interoperability, spectral efficiency, optimization of radio resource usage, improving reliability/performance of the communication system, and primary/secondary usage of the electromagnetic spectrum are additional areas that we believe to be of particular interest for railroad operations. CRs have shown significant promise in each of these categories. CR research has been applied successfully to meet the communication needs of the military as well as the public-safety and commercial sectors, which share many of the same needs as railways.

CE is the learning and decisionmaking engine, or the "brains" behind the CR. CE research draws on principles of human cognition combined with AI and optimization techniques, although the capabilities may be considered simplified when compared with the body of human cognition research. CBR, rule-based reasoning (RBR), GAs, and fuzzy logic represent some popular algorithms that find application in CE design.

The development of the Rail-CR has four key aspects: (1) defining the railroad-specific scenarios and test cases; (2) developing the Rail-CR architecture and associated learning and decisionmaking algorithms; (3) integrating the Rail-CR with an SDR; and (4) implementing the test cases and performance evaluation. We have developed a railroad-specific CR and applied principles and methods from the field of human cognition, AI, and optimization to improve the performance of the wireless communication link. Scenarios and test cases encountered in daily railroad operations were used to evaluate and quantify performance benefits using both computer simulation and real-world over-the-air experiments. Computer simulation of real-world channel conditions and radio parameters was used to enable rapid evaluation and design of the CR and to develop decisionmaking based on AI techniques such as GA and CBR. The architecture design enabled us to replace simulation data with live data from over-the-air experiments and testing and apply the optimized parameter values to the SDR in real time.

The CE interacts with the hardware or radio using middleware, which is composed of an application programming interface (API) enabling the CE to be modular and platform-agnostic. This arrangement will facilitate integration and testing of the CE with radios or hardware from other manufacturers, including radios being used for railroad signaling. The available radio observables (meters) are specific to the radio manufacturer. They may include system performance metrics such as transmit-to-acknowledgment ratio (i.e., success rate), Received Signal Strength Indicator (RSSI), Bit Error Rate (BER), and PER. Furthermore, some SDR platforms may provide the ability to define additional meters and signal-processing blocks based on the application needs.

The Rail-CR architecture builds upon previous research. The CBR is the first decisionmaking module implemented by the CE and draws upon past history of situations, actions, and results to formulate a course of action to address the current situation. When an event occurs, available metrics from the radio are read by the CBR. An event can be defined as a change in the radio performance or environment that falls outside predefined thresholds. The determination of these events will be discussed later. The CBR uses a predefined function to calculate similarity and choose cases in the history file that both closely match the current situation and also had successful improvements in performance. If an improvement in radio performance was achieved in the chosen case(s), the radio knobs are selected to be adjusted in the same manner. Long-term learning is achieved as new successful cases are added to the library. Note that the success of a CBR depends on the depth of experience within the case-history file. Therefore, training a CBR-based radio becomes an important aspect in the development. We examine more details about our cognition architecture and algorithms in the chapters that follow.

To summarize, we have developed the architecture and algorithms to support learning and decisionmaking and demonstrate the performance benefits of Rail-CR using real-world test scenarios in computer simulation, along with lab experiments using a commercially available SDR platform and field experiments. The CR works by optimizing the radio operating parameters based on the operating environment, communication link performance, and past experiences. Our results show that CR is a promising technology for railroads and can offer significant performance improvement over a traditional radio or SDR. We believe that these results will encourage the wireless communications and train control and signaling research communities to conduct additional research on this technology.

2. CONOPS and Functional Requirements

This chapter presents the CONOPS and functional requirements (FR) for Rail-CR. It defines the requirements for a Rail-CR and specifies test cases and scenarios encountered in railroad operations that will be used for system verification and performance analysis. The format of this section is based on the Institute for Electrical and Electronics Engineers (IEEE) standard for CONOPS as designated by *IEEE Guide for Information Technology—System Definition—Concept of Operations (ConOps) Document IEEE Std 1362-1998 (R2007).*

This chapter is structured as follows: Section 2.1 and Section 2.2 present the scope and identification of the project; Section 2.3 offers a description of this document; Section 2.4 examines the current use of frequencies from the electromagnetic spectrum for packet data radio within the railway context; Section 2.5 addresses policy considerations required by the Federal Communications Commission (FCC); Section 2.6 and 2.7 discuss the current system and the justification for why this design is needed; Section 2.8 and 2.9 explore relevant concepts required to understand the system design, including policy considerations; Section 2.10 surveys the details of the system as well as the requirements and scenarios considered; and Section 2.11 and 2.12 review the interface of the SDR and the hardware used.

2.1 Scope

This project focuses on the development of a Rail-CR. CRs are a cutting-edge research area that combines AI and SDRs with the goal of improving upon existing radio performance. SDRs are radios for which some functionality is realized in software as opposed to a purely hardware platform [3]. This type of architecture is similar to a personal computer that uses different software to realize different functionalities and applications. By utilizing situational awareness from the radio in the form of observable parameters, often known as 'meters,' a CE utilizes software-based decisionmaking algorithms to determine whether a change in the radio parameters—commonly referred to as 'knobs'—is required based on sets of predefined goals. In addition, learning algorithms dovetail with the decisionmaking to enable the system to track and utilize past decisions and observations.

The strategy for this research project follows a basic systems engineering philosophy that begins with the definition of FR. These FRs drive the top level and detailed design of the system and provide the basis for validation. Test case scenarios are presented that map back to the core FR, as shown in Figure 1. This report details the core requirements for a Rail-CR and specifies test case scenarios that will be used in system verification and validation.

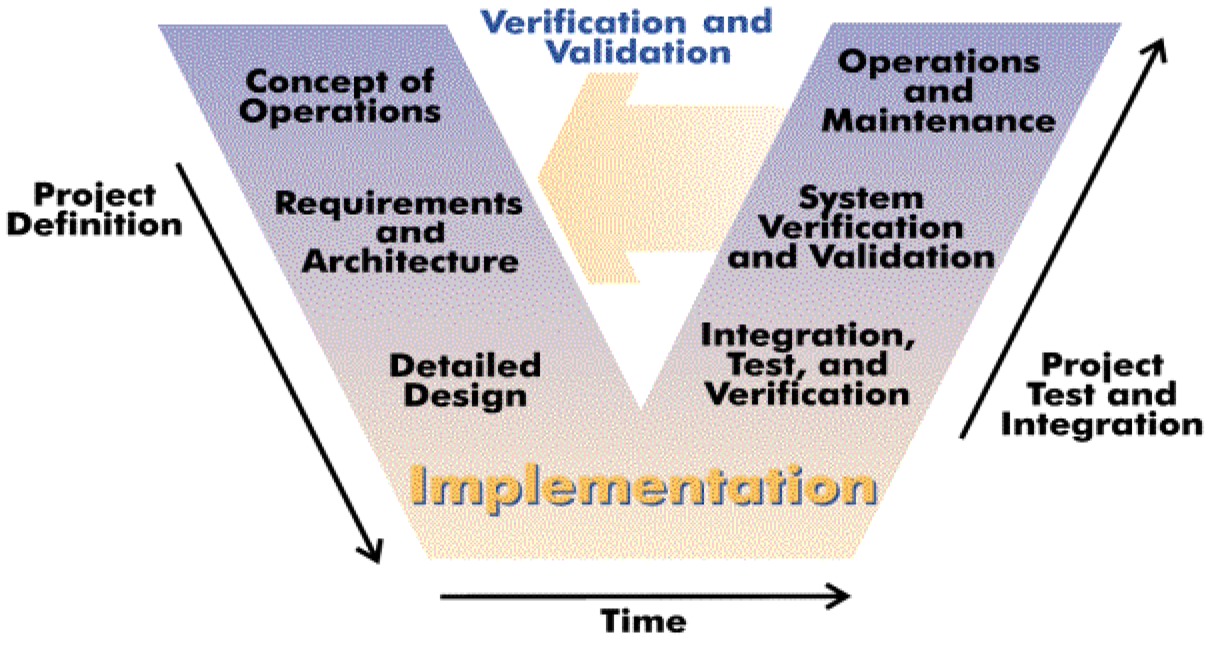

Verification and Validation
Concept of Operations
Requirements and Architecture
Detailed Design
Implementation
Integration, Test, and Verification
System Verification and Validation
Operations and Maintenance
Project Definition
Project Test and Integration
Time

Figure 1. Mapping of FR to Test Case Scenarios

2.2 Identification

This project is sponsored under FRA Research and Development DTFR53-09-H-00021. The title of this CONOPS is Railway Cognitive Radio and the abbreviation is Rail-CR.

2.3 Document Overview

This document is intended for users and developers of SDRs used within a railway context. It describes the functionality and proposed architecture for a Rail-CR that will be composed of software that will control a separate SDR. This software is designed so that it will be able to interface with any SDR regardless of the manufacturer. In order to interface, a ‘middleware’ application will be required to translate commands between the Rail-CR software and the actual SDR. In addition, each SDR will be unique based on which parameters are readable and which parameters are reconfigurable. These elements are the essential inputs and outputs of the Rail-CR. Therefore, the Rail-CR and associated middleware will require customization to fully tie the software to a unique radio.

2.4 Frequency Bands and Their Use for Railroad Communications

Table 1 lists frequencies used within a packet radio context.

Table 1. Frequency Bands and Their Usage for Railroad Communications

Frequency Band	Purpose/Availability
Low band	VHF, 39-50 MHz Data and train control [4] [3]
High band	VHF 160.215- 161.565 MHz for Frequency modulation (FM) equipment [4] Data and voice
220-222 MHz	PTC [3] [5] [6]
450-460 MHz	One-way and two-way end-of-train (EOT) devices [4] Data
UHF	6 channel pairs at 896/936 MHz for ATCS/PTC systems [4] Data
2.4 GHz	Wi-Fi access [3] [7]

2.5 Operational Policies and Constraints

Adherence to spectrum policy is an important issue with regard to SDRs. The ability to change parameters on a radio opens up the potential to violate designated restrictions on spectrum usage. This creates the need for some kind of spectrum policy engine that is capable of verifying configuration commands sent to the radio against defined rules. In a traditional radio, restrictions on frequency operation and power control are often ‘hardwired’ into the device, and it is impossible to operate the radio outside strictly defined parameters. With the SDR, the defined parameters are adjustable and can be set by a configuration command.

Currently, the frequencies utilized within an SDR are regulated under spectrum policies set by the FCC. Specific configuration parameters are verified to meet the FCC requirements and programmed in the SDR prior to the operation. The SDR does not operate outside the preassigned set of configuration parameters and there is no need for policy verification within its current intended operation.

2.6 Description of Current System

The primary purpose of using the SDR in railway application is to provide communication links that allow information exchange among locomotive, wayside, and office networks, as shown in Figure 2. Two scenarios typically are encountered while using the SDR in a railway application. The first scenario is in rail yards and terminal areas where broadband hot spots are available. In this scenario, the SDR can transfer data over the highest throughput link possible—a Wi-Fi link, for example. The second scenario is in the line-of-road operation where data communication is performed on a narrowband wireless network.

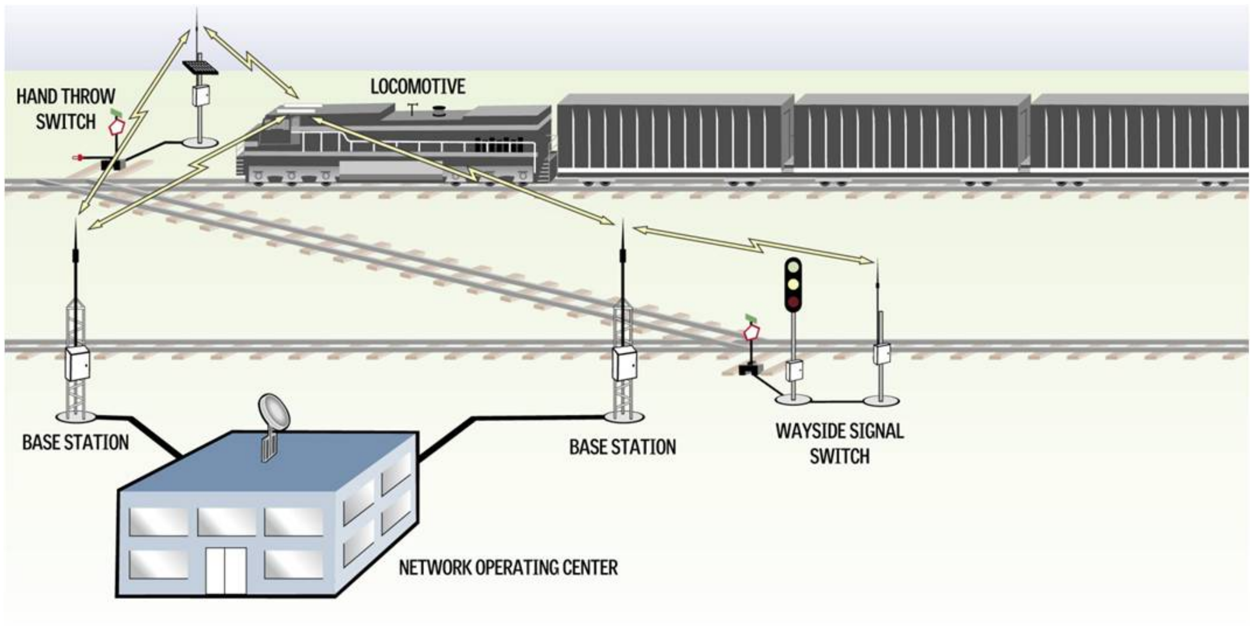


Figure 2. Communications in Railway Application [8]

Currently, SDRs used in a railway environment typically have the option of several frequency bands, including low band, high band very high frequency (VHF), and ultrahigh frequency (UHF), as well as Wi-Fi frequencies. There are usually several choices for modulation, coding, and power. The Meteor Communications Corp. MCC-6100 radio [8], Universal Software Radio Peripheral (USRP) from Ettus research [9], and Lyrtech's Small Form Factor SDR [10] are some examples of SDR platforms. The USRP hardware can be used with open-source software packages such as GNU Radio [11] and has a large user community. For a vast majority of SDR platforms, the tunable knobs would typically include parameters such as frequency band of operation, transmission power, modulation scheme, and forward error correction (FEC). In addition, the ability to program radio functionality in software may allow for the definition of additional knobs.

Although SDRs have the capability to be dynamically reconfigured, they are typically set to a configuration and not changed often. There is limited adaptation within an SDR to changing environments, and SDRs have no long-term learning capabilities. One example of limited adaptation is rate shifting in Wi-Fi systems where the throughput rate decreases as the distance increases between the mobile unit and the base station.

2.7 Justification for the Nature of the Proposed System

Robust, reliable, and interoperable wireless communications is the most vital component to the success of PTC. The objective of this proposal is to demonstrate how a Rail-CR system can enhance the ongoing innovation in communications FRA PTC initiative. A CR will meet the needs of FRA by making railway wireless communications more interoperable, robust, secure, spectrally efficient, and less costly to deploy and maintain. FRA's wireless initiatives include SDR, Communications Management Units (CMUs), and ad-hoc communications between nodes [12]. An SDR is a wireless device that uses software to control the radio's operating parameters and protocols, allowing the radio to adapt through reconfiguration, thus minimizing the need to change existing hardware. SDR is a stepping stone to achieving interoperability between the myriad legacy and new communications technologies in use by railways [3]. Several railways are deploying the MeteorComm SDR, capable of operation in low band VHF, high band VHF, UHF, and 2.4GHz Wi-Fi [8]. CMUs are used to manage voice and data traffic through optimized routing and prioritization, as well as integrate GPS/geographic information system (GIS) data. Ad-hoc peer-to-peer communications provide the wireless connectivity between individual rail cars and wayside equipment.

2.8 Concepts of the Proposed Systems

Rail-CR significantly enhances an SDR platform. Rail-CR uses software-based AI to observe the wireless environment, orient to that environment, make a decision about how to change the configurations of an SDR, and then apply those changes to the SDR (Figure 3).

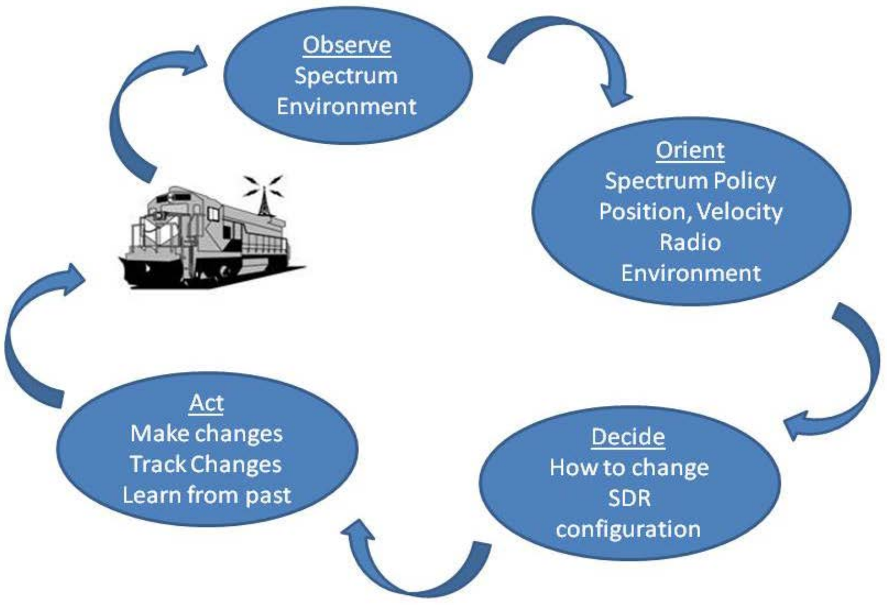


Figure 3. Conceptual View of the CR Process

This process enables the PTC communications backbone to accommodate unanticipated changes in the wireless environment and the changing and evolving application needs of the users. Rail-CR integrates with an SDR foundation by continuously reading the meters of the system and turning the knobs of the SDR. The key benefits of Rail-CR, above and beyond the benefits of SDR, are discussed in detail in Section 3; they include the following: 1) more robust and reliable communications by providing the capability for reacting to interference; 2) improved interoperability between separate railway lines; 3) enhanced information assurance by avoiding interference and identifying and mitigating jamming; 4) improved link performance and higher throughput by avoiding poor channels and increasing data rate on open channels; 5) improved spectrum efficiency by utilizing unused spectrum; 6) lower deployment and operational costs via independent operation, thereby decreasing the need for trained wireless integrators for initial configuration and long term operation; and 7) enhanced interoperability by sensing application needs (voice versus data) and application protocols (ATCS Spec 200, IP (Internet Protocol)) of the user and configuring the SDR to meet the type of traffic.

Without the cognitive capabilities of the CR, a radio link between the wayside and a locomotive is susceptible to outside interference in the form of random wireless interference or deliberate jamming by a security threat. Using the Rail-CR processes, the SDR can sense outside interference, adapt to changes, and restore wireless links (Figure 4). In addition, the system can learn from these adaptations and apply this learned behavior to future situations.

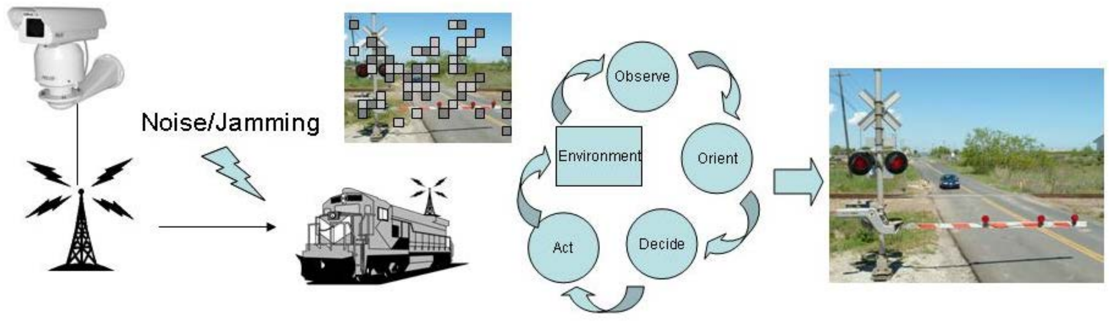


Figure 4. The Rail-CR Will Allow Rail Communications to Adapt to Its Environment

Another example is the use of CR to make more efficient use of licensed spectrum (e.g., in the 160-megahertz band). With spectrum-sensing capability, CRs can detect unused spectrum and exploit it for data transmission in a manner that limits interference to primary users. Moreover, in the absence of narrowband restrictions, CRs can combine multiple narrowband channels to support wideband data applications. Spectrum need not be contiguous, because CRs could adapt to support a variety of configurations.

2.9 Operational Policies and Constraints

The Rail-CR will operate under the same policy constraints as the original system in terms of having to follow designated FCC policies. In the situation of the Rail-CR, a policy reasoner engine will verify that any decisions made by the CE comply with FCC and international regulations based on use and location. As described in Section 3.2, the SDR being used is preprogrammed to only allow frequencies that are FCC compliant. However, the ultimate goal of this engine is to be radio-agnostic and enable integration with other radios, which may not have these preprogrammed safeguards. In addition, sending invalid command requests to the radio will increase time lag in decisionmaking.

2.10 Description of System

The technical approach for the proposed Rail-CR demonstration is based on experience in developing software and CRs for public safety and defense, which have similar needs to the railway industry in terms of interoperability and robustness to interference [13]. The CE consists of simplified case-based learning algorithms that are optimized for efficiency by a GA. The research team’s approach is to develop the CE in software and integrate it with an SDR platform.

This CE is driven by measurable inputs such as spectrum utilization, GPS location, and direction of travel and velocity. These measurable inputs are also known as meters. The CE is integrated with the SDR platform to allow it to manipulate reconfigurable features of the SDR. These features (also known as knobs) could include frequency, transmit power, modulation/coding, channel bandwidth, framing, and other tunable parameters. As diagrammed in Figure 5, the CE of a Rail-CR takes inputs from the meters and turns the knobs of the railway SDR.

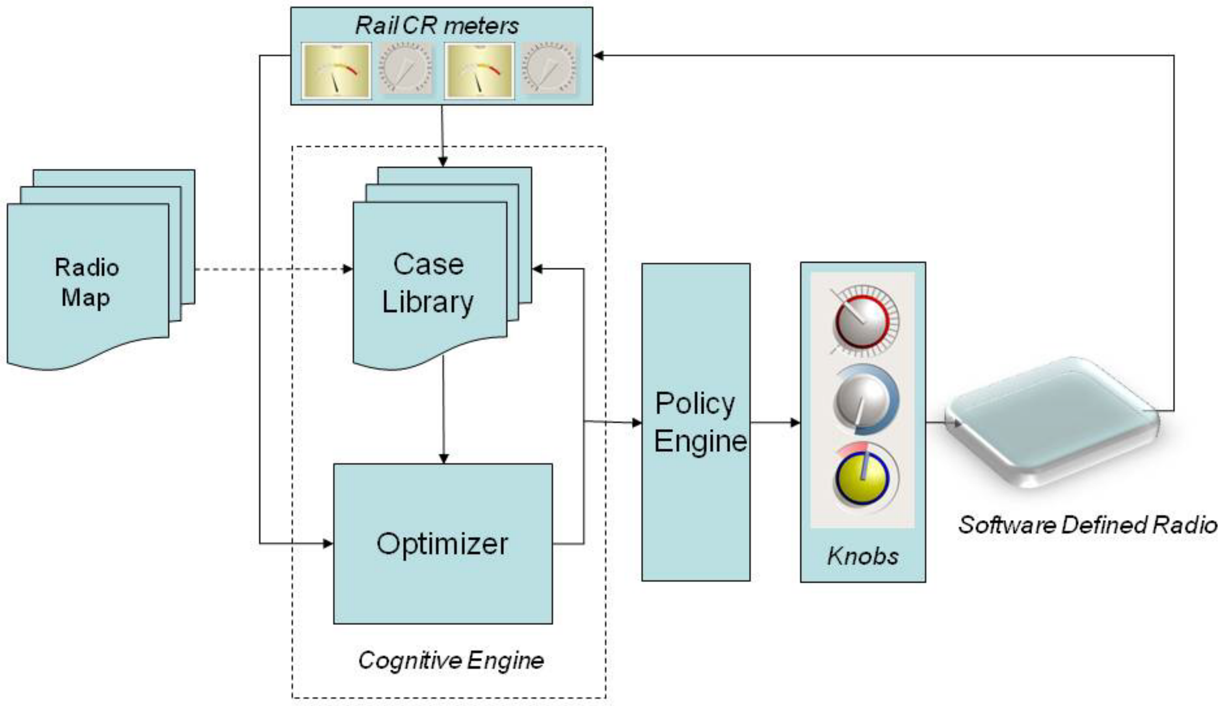


Figure 5. Rail-CR Basic Architecture

The machine learning core of the CE is built on the key design principles of reinforced learning and evolutionary optimization. These design principles are realized in a two-loop cognitive cycle detailed in Figure 6.

The first phase of the cognition cycle involves environment observation that gathers relevant information about spectrum conditions as well as policy and user domain information. The policy domain contains regulatory information such as frequency options, transmission of power, and interference limits. Policy domain knowledge helps guarantee the security and legality of radio operations. User domain information defines and quantifies the service preferences and performance requirements for the end user and includes objectives such as access availability, service type, and Quality of Service (QoS). The CR interprets these objectives upon input into the CE to help inform the adaptations of the SDR to meet changing demands.

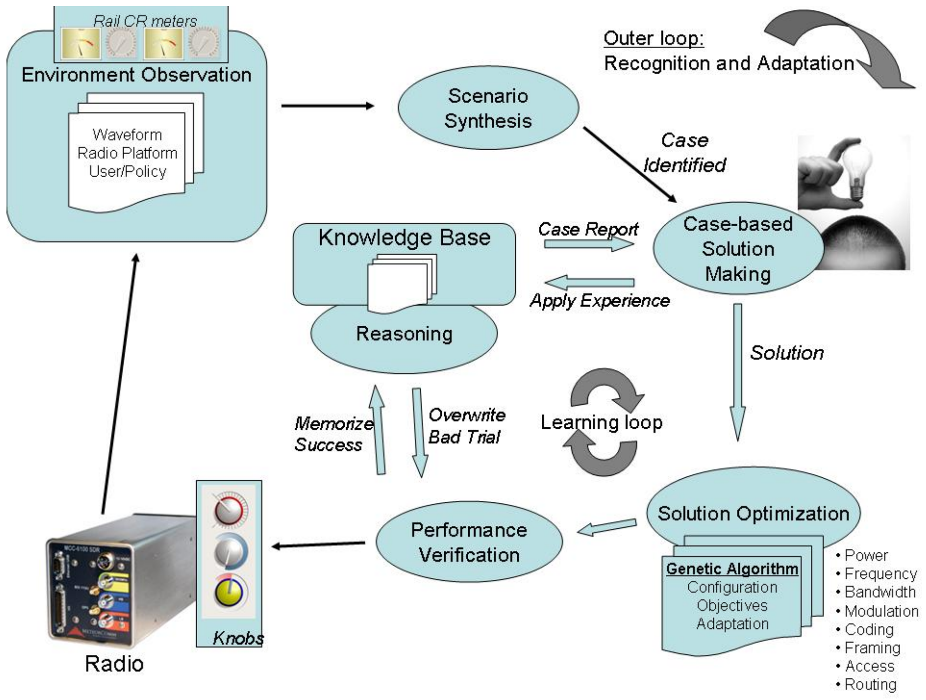


Figure 6. Two-Loop Process: Adaptation and Learning adapted from reference [13]

The research team proposes to use a modular architecture for the components of the CE. This modular architecture will be implemented in a Cognitive Radio Open Source System (CROSS) [14] as shown in Figure 7. The CROSS provides a convenient shell to develop the specific CR modules. Components of CROSS include the Cognitive Radio Shell (CRS), the Policy Engine (PE), and the Service Management Layer (SML). The CRS interfaces with a configuration file that defines what radio elements will be used in operation. The SML acts as a service-oriented architecture and is responsible for making the requests to other modules such as optimization or CBR elements. The SML defines operational goals or ‘missions’ that the radio is tasked with. These missions tie to the test-case scenarios described in Section 5. The mission determines the combination of reasoning and optimization that the system will use to meet the goals. Depending on what the mission is, the pathway chosen can be different. These can be configured from the eXtensible Markup Language (XML) document, which contains the configuration of the engine. The PE will act as the policy checker to verify that configuration commands bound for the radio meet spectrum policy constraints before being applied to the SDR.

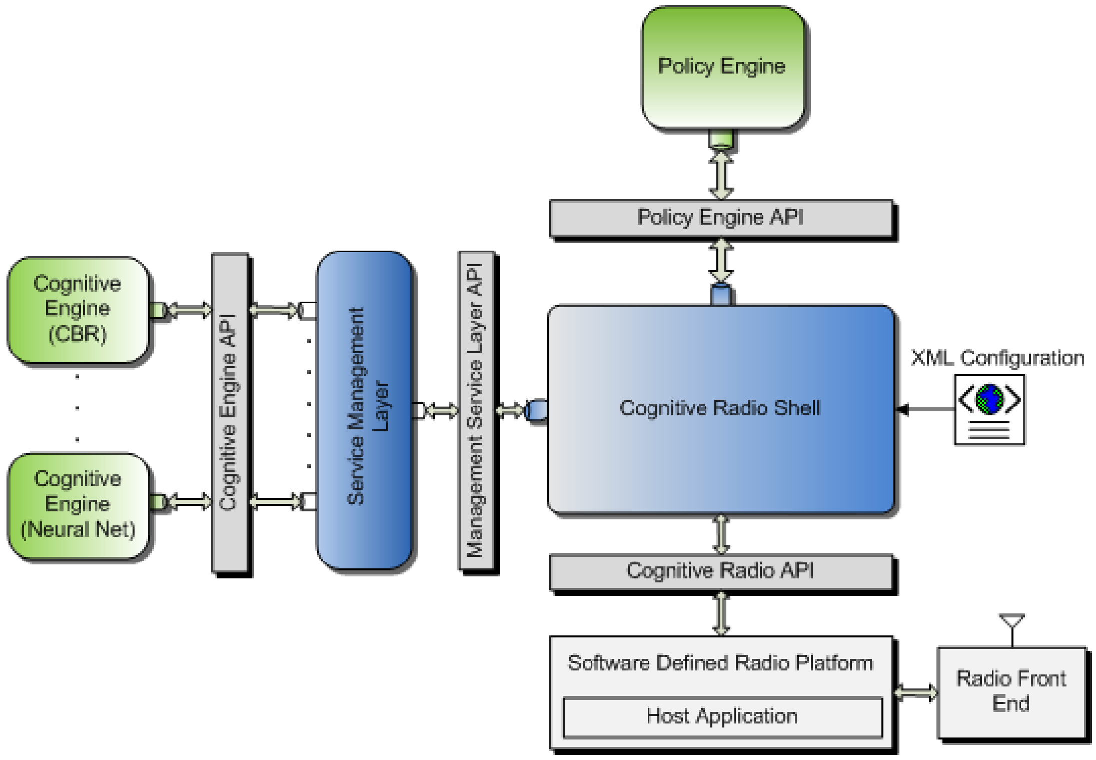


Figure 7. CROSS

The cognition cycle is an evolving loop that includes reasoning, decisionmaking, adaptation, and knowledge accumulation. These functions are realized through two feedback loops that separate the general machine-learning core from the radio-platform-specific operations. The outer loop consists of recognition and behavior adaptation, which are highly dependent on the radio environment map REM database of domain knowledge. The inner loop is where AI algorithms are customized, which then develops solutions and learning from the radio's actions. Just as a child learns from his/her mistakes during the transition from crawling to walking, the Rail-CR can "learn" from its actions throughout its operation. The current architecture adds new cases to the history file when a decision results in an improvement in performance. In this manner, only decisions that lead to positive change are added. When a decision leads to a degradation of performance, this decision is not appended to the case base. In our architecture, the outer loop reports observations such as waveform features, interference and channel propagation characteristics, user preferences, service policies, and spectrum policies to the inner-learning loop. The knowledge base consists of a relational metadatabase that consists of several subdatabases. These databases include the REM, user service knowledge, and a case-based history that keeps track of all decisions made.

In this development, the module acts as a middleware interface between the CE and the SDR's communications software. This middleware provides the translation between the CE and the SDR such that when a command is sent from the CE to change a configuration, the format of the command is placed in the correct syntax that the SDR can understand. In a way, it functions as a dictionary between two languages.

The case-based solution-making determines a course of action based on observed environmental characteristics. To solve a scenario, the problem is matched against a historical library of cases, and then similar situations are retrieved. CBR alone is more effective when dealing with familiar problem scenarios. The Rail-CR architecture allows the system to adapt better to new situations by using an evolutionary searching mechanism to generate more creative solutions. Previous research has found that GAs provide the best features in a CR situation[15][16]. Performance objectives in the form of fitness functions provide a flexible solution search when faced with unfamiliar scenarios. Fitness functions can be related to user needs such as QoS or radio/network performance specifications (e.g., latency or signal strength).

The CE decisionmaking process begins with an event-trigger as shown in Figure 8. As mentioned above, an event can be defined as any significant change in the radio performance or environment. For a CR to operate entirely on its own, it must be able to recognize when an event occurs, triggering the decisionmaking process. Because a CE is simply an add-on module to the SDR with many submodules, a metaCE can be designed to determine which customized CE to use, given a list of possible events to choose from. Each scenario will have a different performance goal (i.e., increased signal strength and increased throughput) associated with it.

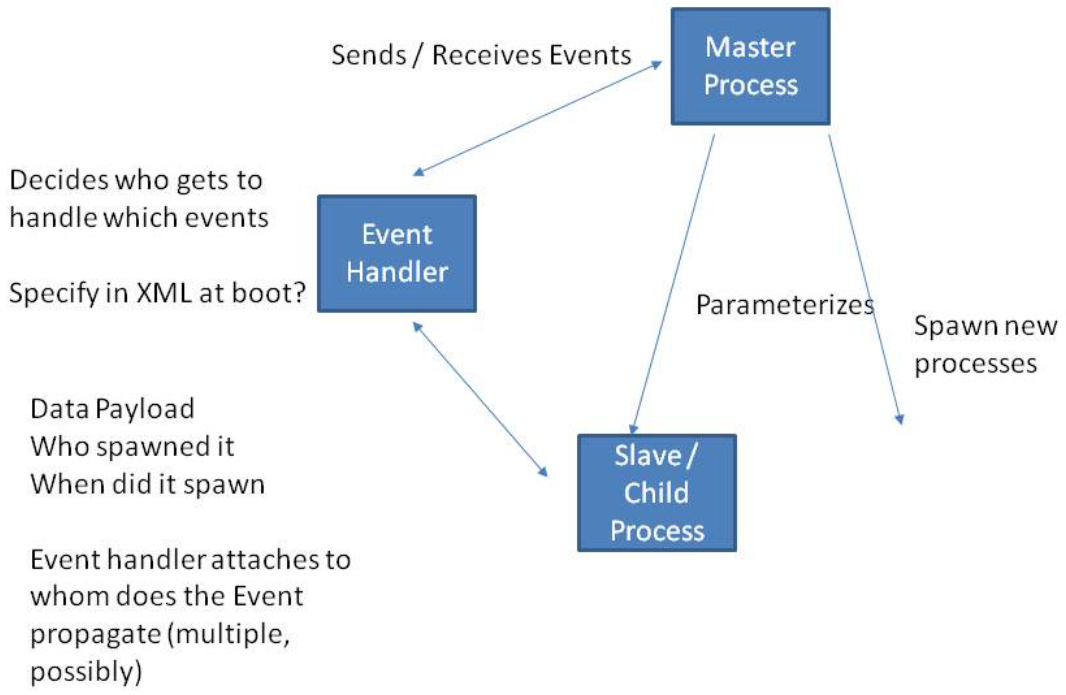


Figure 8. Event-Driven Flow

The CE will most likely be placed in the primary locomotive with the SDR. There is potential to place a sister CE with more cost-effective components at wayside locations. These issues will be addressed during the initial CONOPS and the FR's definition process.

2.10.1 Functional Requirements

2.10.1.1 Ability to Operate in a Noisy Environment

A CR will have the capability to improve operations when compared with the capabilities of a standard radio in noisy environments. Noise may be caused by any number of interference sources or by other radios operating in the same spectrum. An FR of the radio is the capability to use situational awareness garnered from the meters of the radio to mitigate operations within a noisy environment.

2.10.1.2 Ability to Operate in the Presence of a Jamming Signal

This requirement is related to FR 1, although, in this case, the noise and interference is intentional. Similarly, it is desired that the radio be able to recognize a disruption in the service quality and select a new course of action to mitigate the jamming scenario.

2.10.1.3 Ability to Improve Link Performance

The Rail-CR will have the capability to evaluate performance metrics and determine whether they fall within a desired range. If these metrics indicate degradation in performance, the cognitive system can attempt to improve performance by changing certain tunable parameters of the radio such as modulation and power, or finding a better channel.

2.10.1.4 Maintaining Connectivity

Any number of factors can affect the connectivity performance of the radio: from surrounding noise and interference to geographic factors. A CR should have the ability to sense changes in connectivity performance and react accordingly with the goal of maintaining connectivity. The AI algorithms researched within this project incorporate CBR combined with optimization.

2.10.1.5 Conformance to Policy

Spectrum policy as dictated by the FCC or neighboring countries is a very important issue with SDRs. The fact that the configuration of a radio can be affected by simply passing a new parameter file creates the potential for a radio that could be in violation of policy. We propose a PE that works in concert with the CE to verify and enforce policy constraints. Any decision recommended by the CE must first pass through the PE for approval. The PE will be tied to a geographic database of approved policies. The PE safeguards against decisions that the CE makes that may be inconsistent with current policy in the specific geographic area.

Table 2. FRs of a Rail-CR

FR #	CONOPS Requirement	Functional Requirements
1.	Ability to operate in a noisy environment	Sense the wireless environment to identify presence of interference
		Find an open channel/decide best channel
		Reconfigure SDR to new channel
2.	Ability to operate in the presence of jamming signal	Same as 1
3.	Ability to improve link performance	Find an open channel/decide best channel
4.	Maintain connectivity	Reconfigure SDR to new channel
		Maintain database/history of communication devices along with location information
		Search and identify potential (wayside) devices
5.	Radio will operate in line with FCC (and other) regulations	Maintain database of policies
		Maintain database of allowed spectrum usage specific to location
		Ensure CR result implemented only if it conforms to policy

2.10.2 Operational Scenarios

The research team's approach to developing the Rail-CR centers on the mapping of test case scenarios specific to a railway application for each of the FRs. One of the hurdles of this approach is that each test case is 'goal driven,' meaning that a defined performance goal determines how the radio will operate. Conflicts can occur during situations in which goals clash. For example, a goal of FR#5, Conformance to Policy, may dictate that the power level of the radio not exceed a certain threshold within a given geographic region, whereas the goal of FR#4 may drive the radio to increase power level to maintain connectivity. This conflicting issue brings up a research problem known as metaCEs, meaning that an additional CE is needed in order for the original CE to make appropriate decisions regarding which goal needs to be implemented and when. Development in this area will focus on creating individual test case scenarios that map to specific FRs.

The test case scenarios tabulated on the map in Table 3 are directly related to FRs defined in the CONOPS. These cases are specifically defined for railway operations based on current operating procedures and desired functionality.

- ***Policy Verification***

The goal of this scenario is to validate the radio configurations against a defined spectrum policy. The CE will pass any configuration to a PE that must validate the configuration and then pass it on to the radio. The meters used in this case include the operator's country of origin, a database of the operator's authorizations, and a database of spectrum policies. The knobs used include power and frequency.

- ***Identify Presence of Wi-Fi***

In rail yard environments there is often a Wi-Fi higher bandwidth connection available. The Rail-CR must be able to sense whether a Wi-Fi is available to enable the radio to switch to a higher bandwidth connection.

- ***Adapt Parameters to Increase Performance***

Typically, certain configurations default to inactive. In some situations, there may be allowable spectrum to increase coding or channel utilization. The CE will be able to check the current state of these parameters and, if inactive, identify whether the situation will allow it to use increased functionality.

- ***Mitigate Degradation in Performance***

Here, the CE will monitor performance metrics and engage further action if these metrics fall below certain thresholds. Using only available knobs on the SDR, the CE will attempt to change configuration parameters to mitigate interference.

- ***Location-Specific Noise***

Locomotives are often operating close to industrial facilities and the radio frequency (RF) noise created by devices such as welders and power inverters can cause severe signal degradation. If there is a history of noise at a specific location, the radio should be able to see this in the history file. This scenario is similar to the situation discussed in Section 2.10.1.1, but is linked here to location.

Table 3. Test Case Scenarios

No.	Scenario	Current Limitation	Goal of CE Application	Meters Used	Knobs Used	Performance Metric	Mapping to FR
1	Policy verification	No check performed on radio parameters to ensure that they meet policy requirements	Verify that radio parameters meet policy requirements.	Operator's Country of Origin Database of Policies	Power Frequency	Does it meet policy requirements (yes or no); if not, has it been modified to not violate policy?	FR5
2	Identify presence of Wi-Fi	Currently only connects to one Wi-Fi network	Identify if Wi-Fi is available and what SSIDs are.	Create prioritized list of SSIDs to connect to based on identity and location	Wi-Fi SSID Wi-Fi RSSI (Received Signal Strength Indication)	Successful identification of Wi-Fi signal command to switch modes to Wi-Fi	FR4
3	Adapt parameters to increase performance	Some parameters are defaulted to inactive	CEs enable better performance by using past history of channel utilization to anticipate degraded conditions.	GPS location Trans/rec messages that are currently being seen	Error coding	CE at base tells remote to enable certain functionality Should see better PER or increase transmission range	FR1, 3
4a	Interference on current channel	Static settings, performance degraded	React and change settings in order to maintain existing quality.	RSSI T/A (transmit/arrival ratio) packet error	Change tunable parameters on radio	RSSI T/A Connectivity	FR1, 2, 3, 4
4b	Simulate noise in the locomotive At a specific known location range (construction site/welding factory, power substation)	Reconfigure entire network to one band if problem is at locomotive But if it is only at one base location, switch to new band for that location	Historically, it is known that one location always has bad noise. Switch before you even get to it.	GPS location RSSI T/A	Band/frequency	Connectivity RSSI T/A	FR1, 4

2.10.3 Limitations

The Rail-CR’s performance capability will be tied directly to the depth of its case-based history file. The more varied the history file, the more experience the Rail-CR will have to draw upon to make decisions. Therefore, training the CBR and building the history file will be an important aspect of developing the system. The Rail-CR will require a training period during which to grow this history file. In addition, a specific training regimen that includes injecting noise into the system at varied frequencies and locations will assist by providing situations that force the radio to adapt and make decisions.

2.11 Interfacing with SDR

The CE interacts with the hardware or radio using middleware, which is composed of an API that enables the CE to be modular and platform-agnostic. To develop and verify the Rail-CR and demonstrate its performance benefits, we use the USRP SDR, an SDR commercially available from Ettus Research LLC.

The Rail-CR is designed to be platform-agnostic and can be tethered to any SDR using the middleware software provided. However, in this development effort, the Rail-CR must rely on certain aspects of the SDR platform (meters and knobs, for example) for successful operation.

Section 2.11 provides an example, along with the middleware, to enable railroad users to tether the Rail-CR to another SDR platform of their choice.

2.12 Overview of USRP Software-Defined Radio

USRP is a comparatively lower cost hardware solution that runs GNU Radio open-source software. The hardware-software platform has been used widely in developing and demonstrating applications for public safety, dynamic-spectrum access, and rapid prototyping.

GNU Radio is an open-source software package that can be run on USRP. It includes a library of signal processing functions and other radio blocks.

The USRP/GNU Radio hardware-software solution is one of the popular and well-cited solutions for SDR and CR. This system has seen application in military and public safety sectors [17] and has been demonstrated for CR algorithm development.

Table 2 and Table 3 provide a more detailed comparison of functional requirements to operational scenarios by outlining the function of the knobs and meters available.

Table 4. USRP GNU Capabilities Summary

Capability	**USRP and GNU Radio Software**
Link layer/Network functionality (i.e., protocols and methods used to enable multiple communication links to share the communication medium)	Will be using a simple protocol to share channel if it is necessary to demonstrate any radio capabilities; there are some simple Media Access Control (MAC) implementations for GNU Radio.
Over-the-air programming (ability to control the parameters of a radio remotely)	A simple protocol for over-the-air programming (e.g., a special packet type with control information) will be used.
Physical layer functionality (i.e., radio-level functions such as waveform type, transmit power, bandwidth, frequency, etc.)	All required functionality is available; however, extra effort is needed for integration.
Accessibility to information and publishing project results	Open-source software implementation making it accessible to anyone interested in replicating results.

Table 5. Meters of GNU Radio

Meter	**Description**	**GNU Radio**
PER: T/A ratio	Transmit to acknowledgment ratio – average number of transmit packets to received acknowledgment packets	X
Number of transmissions		X
Number of received acknowledgments		X
Message transmit/receive time		X
RSSI	Received signal strength in dBm	X
BER	Percentage of data bits received in error	X

Table 6. Knobs on GNU Radio

Knobs		GNU Radio	Notes
Communication frequency	Low VHF	No	USRP hardware choices support a wide range of frequency bands (220MHz, 900MHz, 2.4GHz, etc.)
	Wi-Fi	X	(Using laptop with Wi-Fi card)
Bit rate (raw bit rate)		Programmable. Higher rates supported as well (Mbps)	
Modulation type	GMSK	X	
	FSK	X	
	CPM	X	
Modulation order	Binary	X	GNU Radio software supports other modulations as well
	4-ary	X	
Pulse shaping filter type	Gaussian	X	
	Root raised cosine	X	
Pulse shaping parameters: BT for Gaussian filter, or roll-off factor for root-raised cosine		Programmable	
Channel coding rate	Parity bytes (0-80 bytes)	X	
	Common FEC schemes	X	
Packet size	14-140 segments of 14 bytes	X	GNU Radio provides more flexibility
Link access protocol		Simple MAC available	

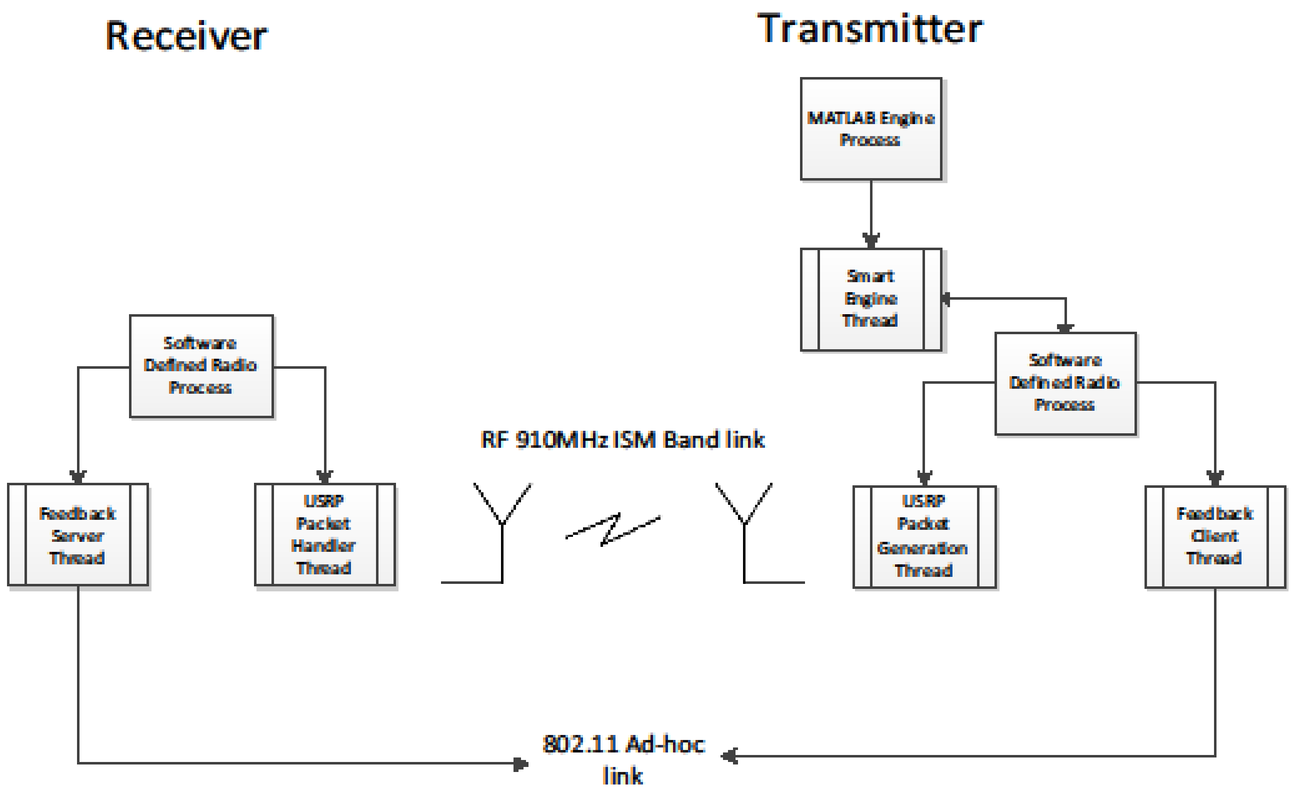


Figure 9. Block Diagram of Transmit and Feedback Links

3. Radio Environment Suite

The radio environment suite empowers the CR to adapt its operation to its environment, thereby making communication more robust and reliable. By providing current-environment information to the CR, the radio environment suite enables the CR to detect and overcome interference and improve performance. This information can be integrated with location information (GPS coordinates, as well as GIS information regarding the location of base stations or wayside units) to form a database that can then drive the location-driven cognition process. The sophistication level of this database can range from simply providing and storing noise levels at various geographic locations to storing noise, interference, and spectrum utilization or open channels at the locations while also being able to differentiate between noise, interference, and jamming. The integration of this radio environment information into a database enables the CR to learn from its past experience, mitigate and work around tough operating environments, and improve performance (throughput, BER, etc.) under good operating conditions.

This section provides a survey of existing methods with a trade-off assessment and suggests a candidate method implementation on USRP/LiquidRadio+GNURadio platform.

The information provided by the radio environment suite can include one or more of the following:

- RSSI, noise-level, presence/absence of interference or jamming, and received signal-to-noise ratio (SNR)
- Spectrum occupancy (i.e., whether a channel/band is occupied or vacant). This will require a survey of techniques for spectrum sensing with assessment of trade-offs in processing cost, accuracy and resolution, and identification of the best candidate method to implement using USRP/GNU Radio.
- Waveform classification. We anticipate that the waveforms most likely to be in use will be Gaussian Minimum Shift Keying (GMSK), Frequency Shift Keying (FSK), and M-ary Phase Shift Keying M-PSK.

The information provided by the radio environment suite will enable the CR to:

- Determine the best operating parameters based on RSSI, noise level.
- Determine and mitigate noise, interference, or jamming.
- Improve link performance and throughput by increasing data rate on open channels based on SNR or other information.

Many techniques, for example, modulation and coding based on the channel conditions and the environment, can be used to achieve the above objectives and are described in literature on adapting link parameters.

In this project, we examined two issues related to the radio environment suite that the CE should address: adapting the modulation and coding of a link to maximize throughput in changing channel conditions—called adaptive modulation and coding (AMC)—and detecting when changing channel conditions are due to interference rather than fading.

Many different techniques exist to perform AMC. Typically, to achieve AMC, it is necessary to make measurements of link quality using either feedback or assumptions of the channel reciprocity to map the measures of channel quality to a modulation and coding scheme (which can be done somewhat independently) and to coordinate with the other side of the particular link (though blind detection and modulation classification can sometimes be assumed). Several different methods for each of these functions are presented herein and cited. Because the optimal design of these parameters depends on the current operating conditions and application, integrating the AMC routines with a CE that can recognize its operating context and appropriately parameterize the AMC block should significantly improve performance.

Interference detection is usually performed by examining and comparing performance metrics where the increased signal power of interference tends to increase some metric while degrading another. After a handful of techniques are presented for detecting interference during communications, this report examines how two common metrics—received signal strength (RSS) and SNR—could be used to detect the presence of an interferer. The report also presents simulation results that illustrate the CE’s operation and behavior.

Integrating either of these techniques will require modifications that must take into account the variances peculiar to the target platform (i.e., supported waveforms, characterizations of statistics for the platform) and environment (i.e., interference sensitivity, assumed channel models).

Given the time duration of our project and some hurdles we encountered while procuring and using an SDR platform, implementation of the above techniques in hardware—and integration with simulation of Rail-CR—was not feasible. However, the project provides design trade-offs and lays a foundation for the radio environment suite that can be used for future work. We present more details of the aforementioned techniques in **Addendum Report: Methods for Detecting Interference and Adaptive Modulation Control**.

4. CE Architecture and Integration with SDR

4.1 Cognitive Architecture

CE vs. Rule-Based Decisionmaking

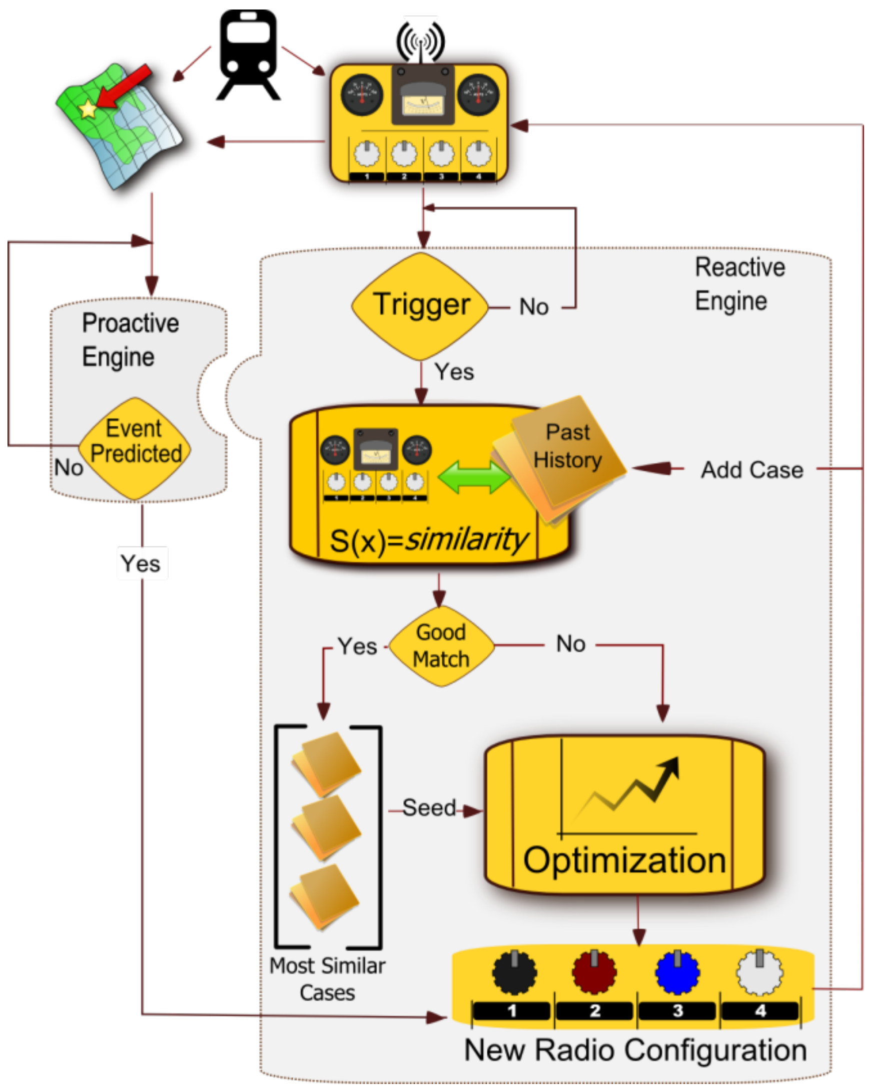


Figure 10. Cognitive Architecture

For the purposes of improving performance, a rule-based decision engine is the simplest to implement. A table of thresholds is set for particular metrics (BER, SNR, etc.), and if one or several of these metrics falls out of the set range (crosses the threshold), then a preset action is taken.

However, the real-world radio environment is too dynamic for such a system to be efficient. As a train crosses the country, it will encounter a large range of radio environments, such as power lines, interfering broadcast signals, RF noise from industrial plants, and, possibly, intentional signal disruption from a rogue radio. Rule-based decision engines have no memory of previous locations or events and are not designed to recognize patterns.

A CE, on the other hand, keeps a memory of past changes in the radio environment, uses these past cases to recognize a similar scenario, and makes a quicker decision in the future. A CE is also able to assign more importance to some metrics over others as directed by the user.

Patterns in the radio environment associated with a particular location or section of track can also be recognized and the CE can adjust its own parameters through metacognition. The goal of the CE is to learn from the past and make the quickest and most appropriate decision that will benefit the performance of the radio, depending on how that is defined by the user.

4.2 The Decision Loop

The main decision loop of the CE is contained within *CELoopSim.m*. First, user-defined meters are read from the radio. The combined meters and knob settings are defined to be the current case. Each case is defined by the original knobs and meters at a sample time. A ‘similarity’ is then quantified and assigned to each case in the radio history and cases are ranked from most similar to least similar. If the most similar case is not close enough to what the radio is currently experiencing, then the radio makes use of the optimization routine. If there are no cases in the history, the current case parameters will automatically be sent to the optimization routine. A new case is added to the case history if the changes made by the optimizer (in this case, a GA) to the knob settings result in improved performance.

4.2.1 Case Base Structure and Use

A large component of CBR is the case base or case history database. This database is filled with cases, each of which describes a scenario with observed meters and associated knob settings, as well as the new knob settings that the radio decided to set after running through its optimization routine.

Case Structure

The case base is defined in MATLAB to be a structure with multiple branches (Table 1). Branches can be added or taken away, depending on the knobs and meters that are available with the particular radio being used.

Table 7. Case Base Structure

CASEBASE (index #):	Scenario id	
	Old Knobs	Old_Knob_1
		Old_Knob_2
		:
		Old_Knob_n
	Old Meters	Old_Meter_1
		Old_Meter_2
		:
		Old_Meter_n
	New Knobs	New_Knob_1
		New_Knob_2
		:
		New_Knob_n
	Utility	Utility_1
		Utility_2
		:
		Utility_n
		Utility_aggregate (fitness)

The structure of the branches used by each case can be defined by the user in *radioDefinition.m*. Each main branch (Old Knobs, Old Meters, etc.) is defined here and the case structure is then defined at the beginning of *CELoopSim.m*. The result is a multilevel structure. The actions associated with each case in the case base are defined by the “new knob” values that were decided upon by the cognition loop. The associated utility values and overall fitness of the resulting configuration are also stored. The ‘utility’ of each knob and meter is calculated and used to determine fitness. All of these values are stored in the case entry to be used for later similarity calculations and for choosing the initial population of the optimization routine.

4.2.2 Utility Functions

The fourth main branch in the case structure is labeled ‘Utility.’ A utility value is a method for quantifying the favorability of a particular value for each metric. To combine radio metrics into a useful fitness measurement, the raw values must first be placed in a similar scale. Each knob and meter is assigned a custom utility function that yields a value between 0 (not desirable) and 1 (extremely desirable). Utility functions are defined in *setUtility.m* and *ga_utility.m*.

4.2.3 Ranking Cases

Distance (Similarity)

The main process of CBR is the determination of similarity between each case in the case base and the current radio configuration. Computation speed can be increased by first filtering out all cases that do not match the current scenario id. However, this will still leave a large list of cases to sort through. A quantifiable method of determining similarity is essential for the CBR to be successful.

One of the simplest ways of determining similarity between two vectors is to calculate the total Euclidean distance between them. However, this method requires that all numbers in all vectors are scaled the same way. For example, if one of the elements in the vector is occupied by an SNR metric and the units of that metric are in decibels, and another element is occupied by BER, the SNR metric will always have more influence in the similarity calculation because the BER values will be miniscule in comparison, having a negligible effect on the total Euclidean distance. Instead of using the raw values, we propose the use of utility values associated with each metric. Using the utility values for each metric as the elements for distance calculation creates a fairer comparison because all values will be in the set between 0 and 1. Similarity or distance, D, is defined as the Euclidean distance between a case in the case base and the new (or current) case. For metrics, m=1,…,n:

$$D = \sqrt{\left(u_1 - u_{1,new}\right)^2 + \left(u_2 - u_{2,new}\right)^2 + \ldots + \left(u_n - u_{n,new}\right)^2}$$

Weighted distance is also an option:

$$D_W = \sqrt{w_1\left(u_1 - u_{1,new}\right)^2 + w_2\left(u_2 - u_{2,new}\right)^2 + \ldots + w_n\left(u_n - u_{n,new}\right)^2}$$

Fitness

The fitness of a case is a user-defined combination of the associated utility values of the knobs and meters. The utility values are also adjusted according to the user-defined weighting values (to give more influence to one metric over others) that are defined in *Loop_OuterLoop.m* along with the desired fitness function. Choices of fitness functions include:

$$f = \frac{1}{N}\sum_{i=1}^{N} w_i \cdot u_i \qquad f = \left(\prod_{i=1}^{N} u_i^{w_i}\right)^{1/N} \qquad f = \frac{1}{N}\sum_{i=1}^{N} u_i^{w_i}$$

Ranking

The ranking method used to sort the cases in the case history is required to solve the following problem: A case with high fitness may not be appropriate for the current scenario. To account for this, fitness and distance have been combined to form an overall similarity or ranking, S(d,f). It is this overall similarity threshold that is set by the user in *Loop_OuterLoop.m*. Ranking methods can also be chosen by the user and include the follow methods for the current scenario and case x:

$S_{x,o}(d,f) = (1-d)\cdot f$

$S_{x,o}(d,f) = f^d$

$S_{x,o}(d,f) = d$

$S_{x,o}(d,f) = f$

$S_{x,o}(d,f) = \|BER_x - BER_o\|$

4.2.4 Case Base Reasoning Walkthrough

The CASEBASE list stores the following values:

1. id: indicates if valid values are stored in that entry. If it's 0, it is invalid; if it's 1, it is valid.
2. knobs_old: stores the values of the knobs before they were changed.
3. meters_old: stores the values of the meters that triggered the CE.
4. knobs_new: stores the values of the knobs after they were changed.
5. utility: stores the new utilities and fitness after the knobs have been changed and the meters measured.

First, we run the simulation four times (with the first run having nothing stored in the CASEBASE), each time with a different environment:

The environment vector indicates the power level of the interference source and when it should turn on. For example, an environment vector of *env(:,1)== [1 1 1 6 6 6 6 1 1 1]'* will turn on the interference at an amplitude of 6 dB starting at time index 4 and turn it off at time index 7. The figures below illustrate the operations of a simple one-meter, one-knob CE. BER acts as the observed meter and transmit power is the only available knob. When BER exceeds a defined threshold, then the engine decides to increase Transmit power in response.

Four separate environments were created and the response from the CE was tracked and placed into the case base.

1. *env(:,1) = [1 1 1 6 6 6 6 1 1 1]';*

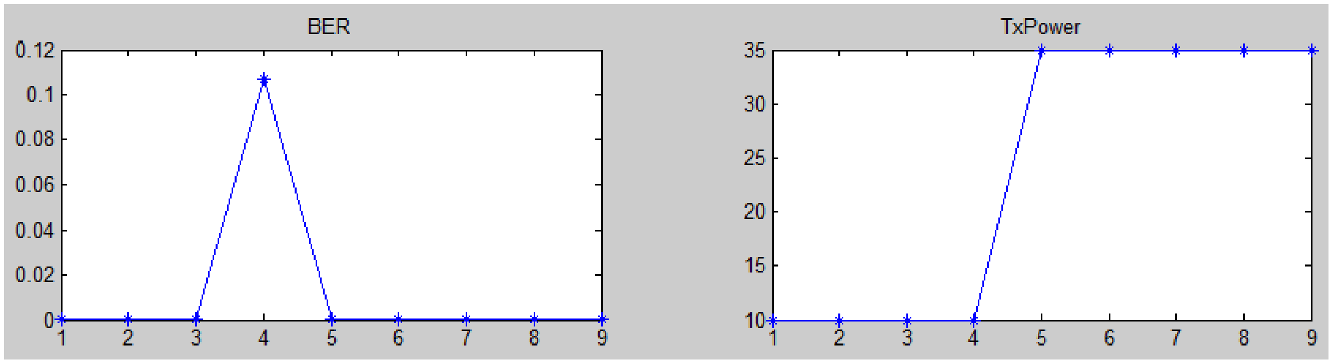


Figure 11. Simulating the Operating Environment in MATLAB Software

2. *env(:,1) = [1 1 1 4 4 4 4 1 1 1]';*

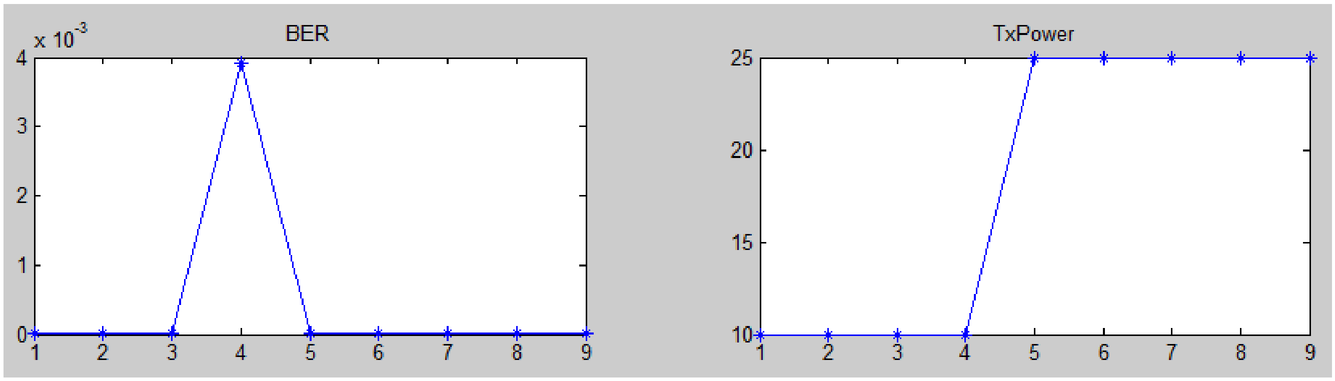


Figure 12. Simulating the Operating Environment in MATLAB Software

3. *env(:,1) = [1 1 1 3.5 3.5 3.5 3.5 1 1 1]';*

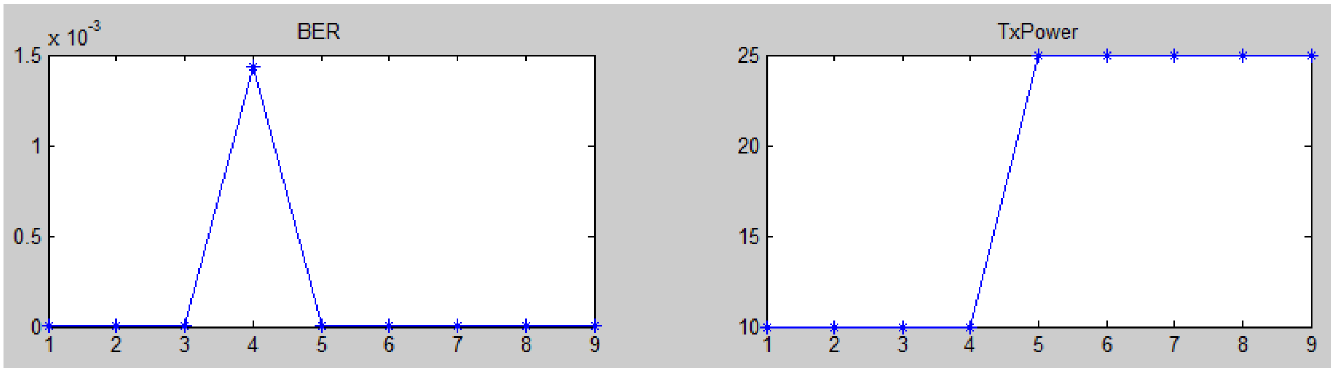


Figure 13. Simulating the Operating Environment in MATLAB Software

4. *env(:,1) = [1 1 1 5 5 5 5 1 1 1]';*

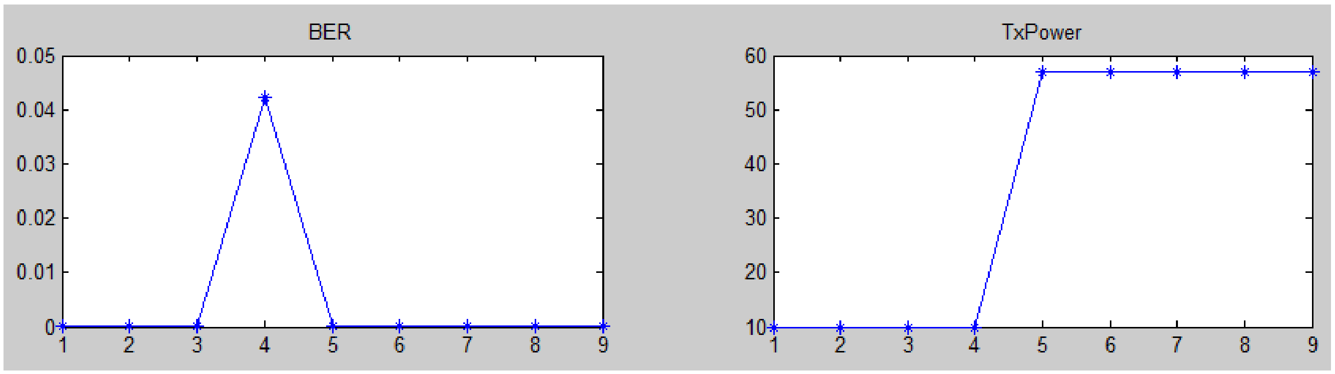


Figure 14. Simulating the Operating Environment in MATLAB Software

The radio measures the meters (BER, etc.) and calculates its utilities and fitness. The current way to calculate fitness is: fitness = ∑(utilities*weights), where the value of weights are currently txPowerWt = 0.8 and berWt = 0.3.

If the meters are higher than a certain threshold (now it is triggered if BER > 0.001), then the CE is triggered. Once the CE is triggered, the system checks if there are more than two cases stored in the CASEBASE list:

1. If there are two cases or less, it will automatically run the GA without seeding it.
2. If there are more than two cases in the CASEBASE, then it ranks them according to the similarity to our measured case.
   - If the highest similarity is higher than a threshold (now RANK_THRESHOLD = 0.243), then the CE uses that case from the CASEBASE as the solution, and it changes the values of the knobs.
   - If the highest similarity is lower than the threshold, then it seeds the GA with the eight most similar cases (or with less if there are not so many cases stored in the CASEBASE). To calculate similarity, first the distance between the utilities from the previous cases and the current cases is calculated (currently using Weighted Eucledian Distance where the weights are the same as for calculating the fitness): Similarity = (1-distance)*fitness

After running these four simulations, the contents of the CASEBASE list will be:

CASEBASE	id		1
	knobs_old	TxPower	10
	meters_old	BER	0.1071
	knobs_new	TxPower	35
	utility	New_Fitness	0.2441
		TxPower	0.9091
		BER	0.0281
	id		1
	knobs_old	TxPower	10
	meters_old	BER	0.0039
	knobs_new	TxPower	25
	utility	New_Fitness	0.2593
		TxPower	0.9091
		BER	0.1782
	id		1
	knobs_old	TxPower	10
	meters_old	BER	0.0014
	knobs_new	TxPower	25
	utility	New_Fitness	0.2593
		TxPower	0.9091
		BER	0.2853
	id		1
	knobs_old	TxPower	10
	meters_old	BER	0.0422
	knobs_new	TxPower	57
	utility	New_Fitness	0.2108
		TxPower	0.9091
		BER	0.0485

Now we run the environment: env(:,1) = [1 1 1 4.2 4.2 4.2 4.2 1 1 1]'; which uses the CBR.

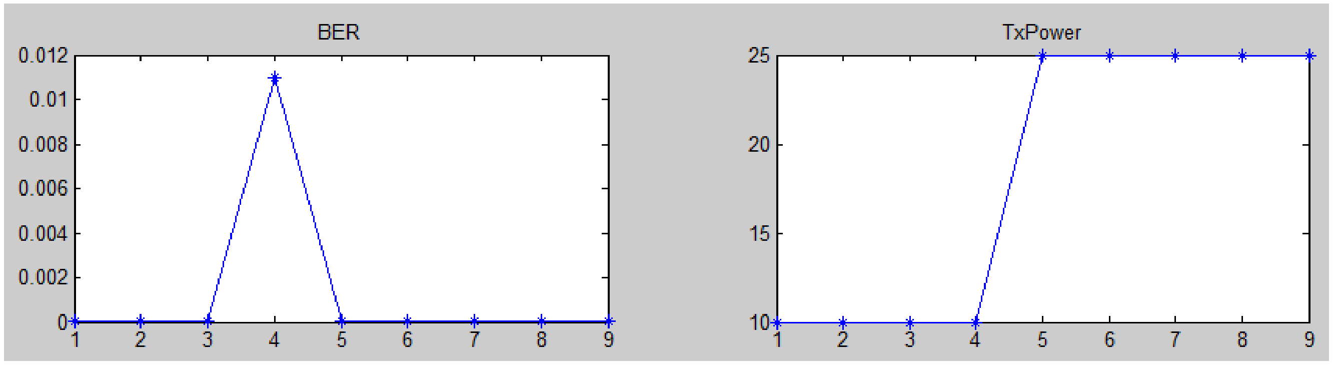


Figure 15. Simulating the Operating Environment in MATLAB Software

For this case, its utilities and fitness were:

Utilities		Fitness
TxPower	BER	
0.9091	0.1036	0.1519

Comparing with the cases stored in the CASEBASE:

Case	Fitness	Distance	Similarity
1	0.2441	0.0413	0.234
2	0.2593	0.0408	0.2487
3	0.2593	0.0995	0.2335
4	0.2108	0.0301	0.2044

The case that has the highest similarity is case number 2 (similarity = 0.2487), which is higher than RANK_THRESHOLD = 0.243, so the CE uses this previous decision for the current situation.

After making a decision, the CE sends the new knob values to the radio and then receives the new measured meters back. It then calculates the utilities and fitness for these meters. Then, this new fitness is compared to the old one (the fitness that we had before changing the knobs) and if it is higher, a new case is added to the CASEBASE. After this, the process is restarted.

4.3 Genetic Algorithm

4.3.1 GA Walkthrough

When the GA is triggered, it does the following:

1. First, it seeds the top third of the population with the parent cases, which are:
    - If in the CASEBASE list there are more than two cases, the parents are knobs_new of previous cases from the CASEBASE.
    - If there are two cases or less in the CASEBASE, the population is seeded with the current measured knobs.
2. It sets the lower two-thirds of the population with random bits.
3. It starts running generations. In each generation, it does the following:
    - Calls ga_fitness.m: For each chromosome of the population:
        a) Converts the genes from binary to decimal.
        b) Quantizes code rate.
        c) Calls est_radio.m:
            - Each chromosome is used as an estimate of what the knobs should be changed to.
            - The SNR for these possible new knobs is estimated.
            - BER and PER are estimated assuming an Additive White Gaussian Noise (AWGN) channel.
        d) For the estimated knobs and meters, utilities and fitness are calculated.
        e) It gives back an array with the fitness for each chromosome.
    - It adds up all the fitnesses into one single value.

    - The whole population is seeded with parent chromosomes. These parent chromosomes are selected the following way:
        a) For the first generation and for the first number of original parents with which the GA was seeded within the population, the parents are what the GA was seeded with.
        b) For the rest, the parents are randomly selected using roulette.m (this function chooses with a higher probability parents that have a higher fitness).
    - Calls crossover.m to cross the parents. This function will cross with a certain probability two parent chromosomes. If it does cross them, at a certain random point within the chromosome, the bits that come after that point will be exchanged.
    - Calls mutate.m, which randomly mutates some of the bits of the whole population.
    - It does this entire process for each generation.
4. When it finishes running generations, it chooses the chromosome with the highest fitness value.
5. It makes sure the knobs are within acceptable range. If it is not the case, then they are changed to the closest possible value.
6. It quantizes code rate.
7. It gives back the new knobs.

4.3.2 The GA and Adjusting Parameters

A GA was chosen as the optimization routine for this CE. The number of generations was limited for the purpose of curtailing latency. The GA default configuration is run with the following parameters:

- Population Size: 100
- Maximum Generations: 10
- Crossover Rate: 0.7
- Mutation Rate: 0.01

If latency is less of a concern, population and/or generation size can be increased inside the file *GA*.m. An essential component of the GA is the estimation of the fitness of each chromosome in the population of each generation.

4.3.3 Estimating Radio Performance

Because BER has an effect on fitness, the BER utility must be estimated for each gene of each chromosome of each generation of the GA. Therefore, the MATLAB ‘bercoding’ function is used to estimate BER quickly, returning an upper bound of the BER of the convolutional code.

4.4 CE Loop Decision Architecture

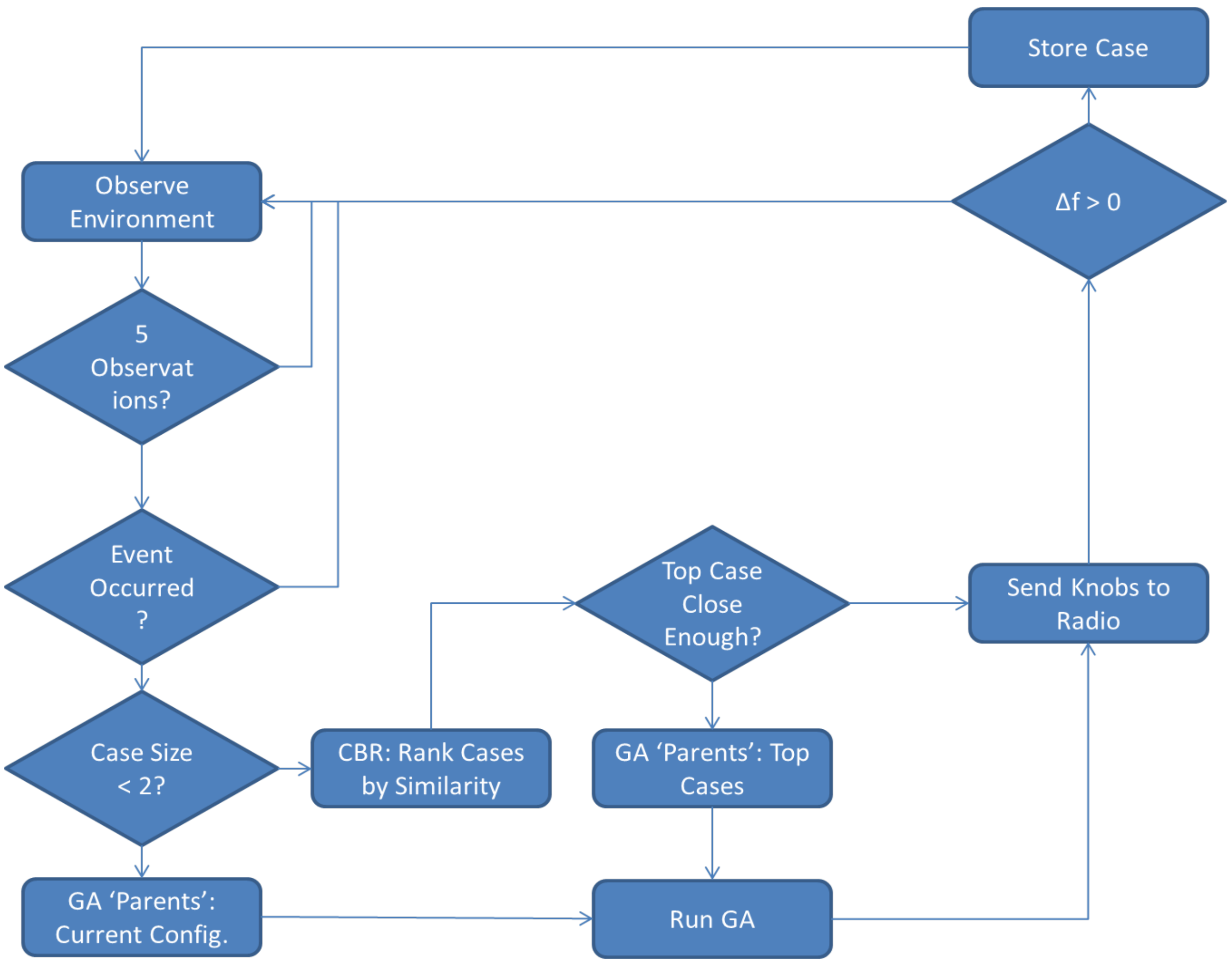


Figure 16. CE Decision Architecture

4.5 Defining and Changing Knobs/Meters

To add additional knobs or meters to the radio, modifications must be made in the following files:

radioDefinition.m:	Add new knob/meter to knobsFormat or metersFormat.
getUtility:	Adjust knobs/meters input to ga_utility.m.
setUtility:	Add utility function to correspond to new knob/meter.
ga_utility:	Add utility function to correspond to new knob/meter.
Cbrmatchid:	Add additional CASEBASE(jj).utility entry as the last row in 'utilVec'.
CELoopSim:	If adding new knob, add knob range to 'k_ranges'.
	Add new knob to both occurrences of 'parent_knobs'.
	Add new knob to k_now BEFORE CURRKNOBS.PacketSize.

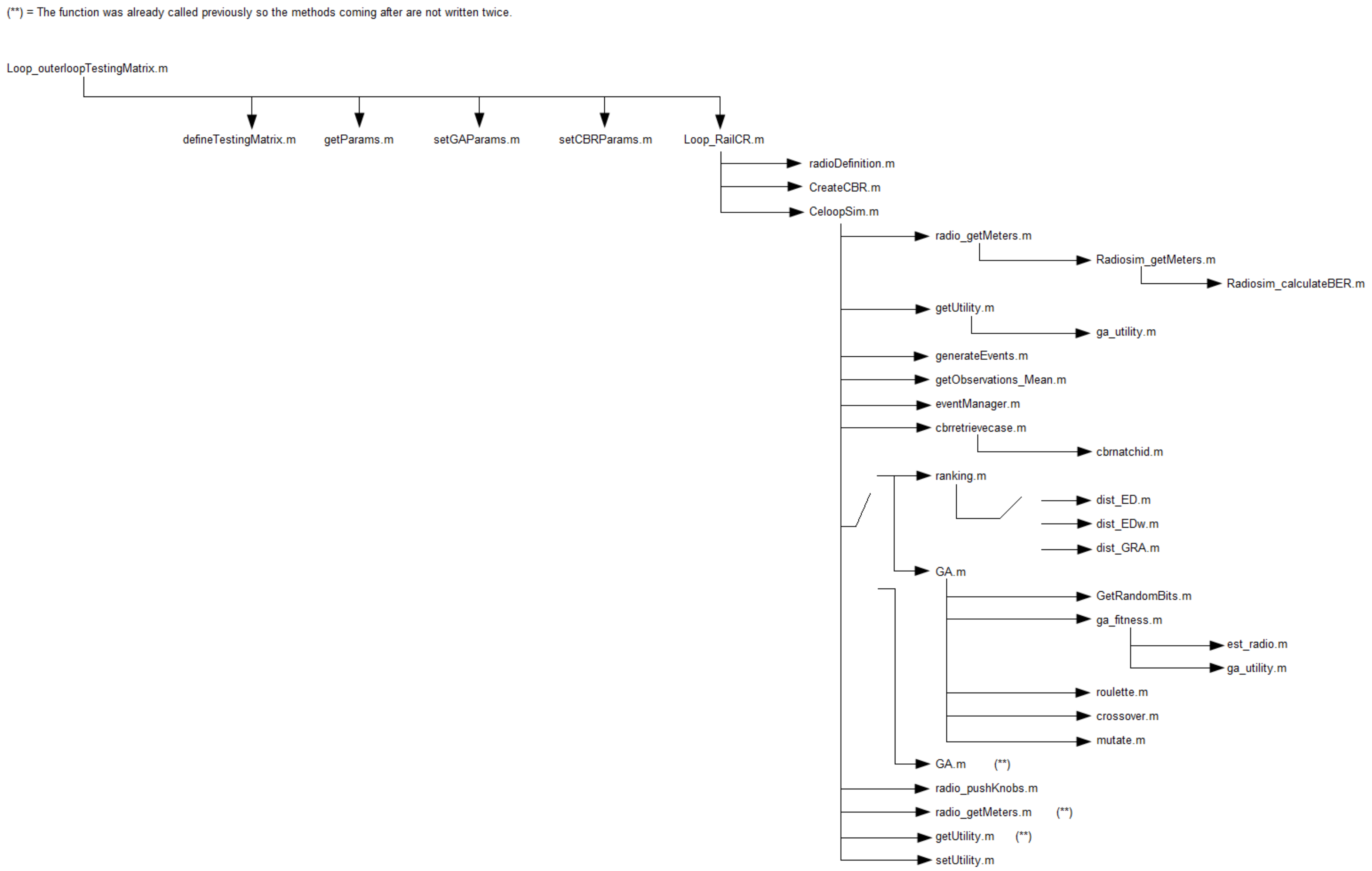


Figure 17. Flowgraph of CE MATLAB Modules

5. Test Plan and Results

Test Plan:

The CE is tested at two different sites:

1. The Mobile and Portable Radio Research Group (MPRG) lab in Durham Hall at Virginia Tech.
2. The rail yard at the Museum of Transportation in Roanoke, VA. At this site, the radios were tested at three different locations:
    a. Short distance: LOS with a distance between the radios of approximately 30 ft.
    b. Long distance: LOS with a distance between the radios of approximately 60 ft.
    c. Short distance between trains: No-LOS with a distance between the radios of approximately 30 ft.
    d. Long distance between trains: No-LOS with a distance between the radios of approximately 60 ft.

At each site five different wireless environments were produced for the radios to overcome:

1. Interferer off: The noise floor is simply that of the site in which the test is taking place.
2. Wide-band noise floor increase: The overall noise floor, centered at the transmitted signal carrier frequency and with a wider bandwidth than the transmitted signal, is raised.
3. Static narrow-band noise spike: A high power noise spike with a much narrower bandwidth is inserted into the transmitted signal bandwidth at a fixed frequency.
4. Hopping narrow-band noise spike: A high power noise spike with a much narrower bandwidth is inserted into the transmitted signal bandwidth with a shifting center frequency.
5. Fade: The amplitude of the transmitted signal is randomly varied to simulate an environment with a strong fading component.

To compare the CE system versus the no-CE system, it is assumed that the no-CE case is designed for the noise environment "Interferer off" at the lab site.

For the CE system, the CE parameters that need to be tuned are:

- GA Crossover Rate: It is the probability that two parents, after being selected from the population in the GA, will cross over.
- GA Mutation Rate: It is the probability that each bit of the GA population will get modified (changed from one to zero or from zero to one).
- GA Population Size: It is the amount of chromosomes that the GA has in each generation.
- GA Max Generations: It is the maximum amount of iterations that the GA is allowed to make before coming up with a solution.
- Case Base Size: It is the maximum amount of cases that will be stored.
- Similarity Threshold: It is a metric for the CE to decide whether to use the GA or the CBR. If the current case's similarity to any of the ones stored in the CBR is higher than this threshold, the system will use the CBR, and if it is lower, it will use the GA.

- Current PER Threshold: If the measured PER is over this threshold, then the CE will get called.
- Knobs and Meters Weights: Knobs and Meters Weights are used for the calculation of fitness. The relative value of each weight with respect to the others will give each parameter more or less significance in the calculation of the value of fitness. This allows the GA to know how to compromise knob and meter values to try to find an optimum solution based on the goals.

The values of these parameters can be seen in the following table:

GA Xover Rate	**5.1.1 GA Mutation Rate**	**5.1.2 GA Population Size**	**5.1.3 GA Max Generations**	**5.1.4 Case Base Size**	**5.1.5 Similarity Threshold**	**5.1.6 Current PER Threshold**
0.7	0.01	100	25	100	0.3	0.1

Tx Power Weight	**Packet Size Weight**	**Modulation Type Weight**	**Coding Type Weight**	**SNR Weight**	**PER Weight**	**Spectral Efficiency Weight**	**Throughput Weight**
1	0.3	0.2	0.5	0.5	5	0.3	0.3

The initial values of the transmitter knobs (transmit power, packet size, modulation, and coding) need to be determined. For the case of the CE system, they will be set to values that are close to the most ideal case, that is, close to minimum power, close to maximum packet size, highest modulation order, and no coding. For the case of the no-CE system, different tests were run with different combinations of knobs to find a good set of them for the case of "Interferer off". This set of knobs will also be used for the other types of environments to be tested. The initial knob values for both systems can be seen in the following table:

	Software Gain (dBm)	**Packet Size**	**Modulation Type**	**Coding Type**
CE	-20	300	64-QAM (Quadrature Amplitude Modulation)	No Coding
No-CE	-65	300	32-QAM	No Coding

For the tests, a 1-MB data file is transmitted. However, because the Packet Size changes for the CE system, the exact amount of transmitted bits changes for each run.

For the CE system, the knob ranges are:

- Software Gain: [-65 dBm, -15 dBm]
- Packet Size: [20, 300]
- Modulation Type: [BPSK (Binary Phase Shift Keying, QPSK (Quadrature Phase Shift Keying), 8-PSK, 16-QAM, 32-QAM, 64-QAM]
- Coding Type: [Hamming (7, 4), No Coding]

Results:

The first figure shows how the knobs and meters change during a run over time (in seconds). Within this plot, there are eight graphs:

- The top four show what the different values of our different knobs are and what they are changed to when the CE gets called. For the case of no-CE, these values will remain constant over the whole run.
- The bottom four show the values of the different meters at the time instances when they are measured. For this reason they are represented by dots. The blue dots are the meters the radio measured, whereas the green ones are what the GA estimated these values should be. On the PER graph, there is also a red line that represents the threshold for when the CE should get called. Therefore, if there is a blue dot in the PER graph that is over the red line, then the CE will get called (for the CE system).

For the case of the CE system, there is also a second and third plot. The second one shows at what instance the CE was used, where “2” means the GA was used, “1” means that the CBR was used, and “0” means the CE was not used. The third plot has two subplots: the first one shows how the bias changed while the system was searching for the optimal bias, whereas the second one shows the difference between the observed PER and the estimated PER. As an example, the plots for the Long Distance test at the rail yard are shown below:

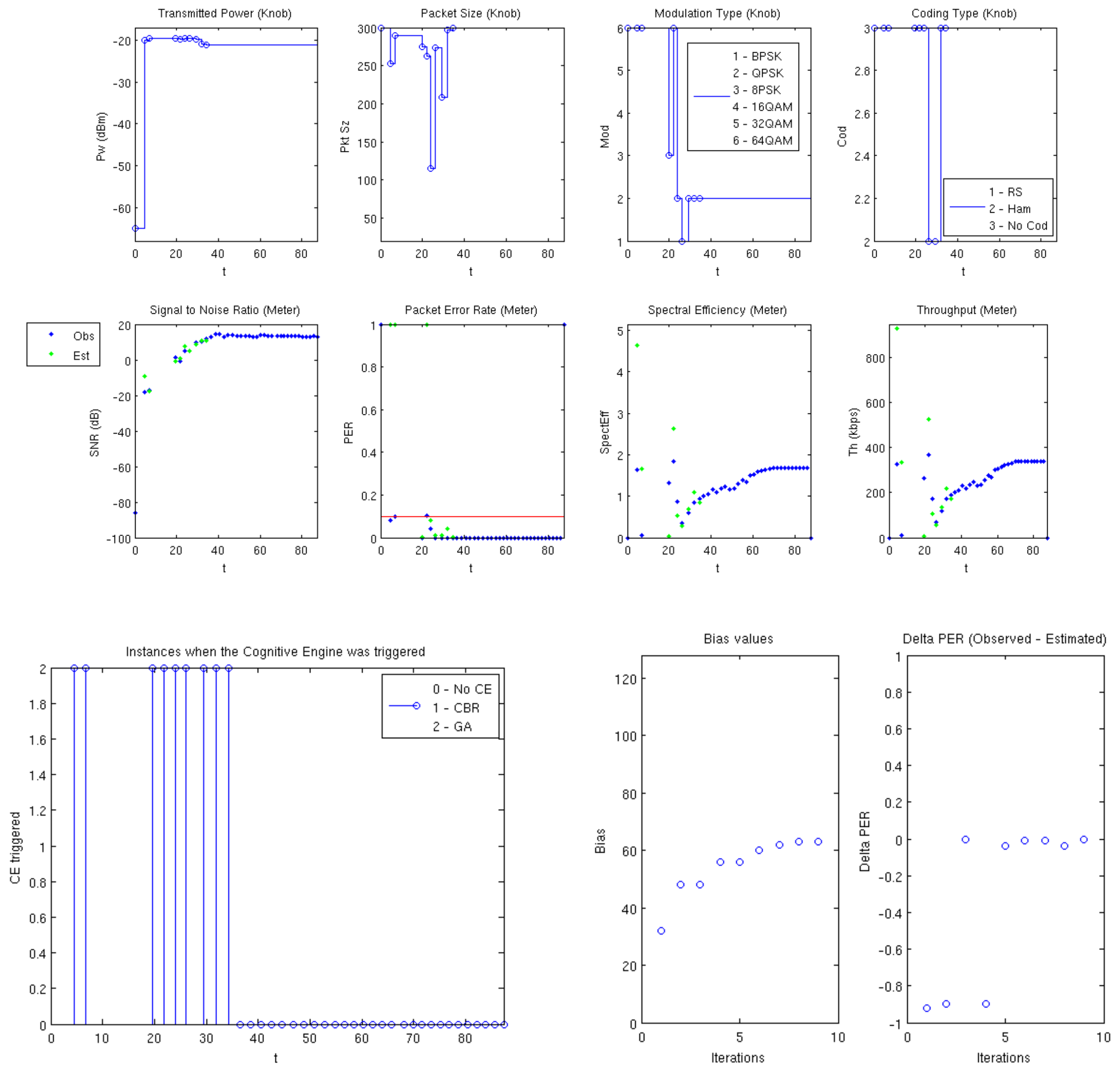


Figure 18. Radio Performance Results

In the following table, the mean value of the meters for the CE system after the CE was called for the first time versus the mean value of the meters for the no-CE system are compared (on the following page):

Table 8. CE System versus Non-CE System Meters

				METERS			KNOBS			
Location	Wireless Environment		PER	Spectral Efficiency	Throughput (bps)	Software Gain (dB)	Packet Size (Bytes)	Modulation	Coding	Fitness
		No-CE	0.302	3.0272	6.05E+05	-30	300	5	3	0.027961082
	Ambient	CE	0.0256	2.6057	5.21E+05	-29.81	281	4	3	0.607887135
		No-CE	0.9996	0.6057	1.21E+05	-30	300	5	3	5.39742E-05
	Noise	CE	0.0738	2.4148	4.83E+05	-25.01	225	4	3	0.551582171
		No-CE	1	0	0.00E+00	-30	300	5	3	0
	Jam	CE	0.116	1.9967	3.99E+05	-20.48	221	3	3	0.42803398
		No-CE	1	0	0.00E+00	-30	300	5	3	0
	Hop	CE	0.2014	1.8162	3.63E+05	-22.7	287	2	3	0.137967666
		No-CE	0.8999	1.4989	3.00E+05	-30	300	5	3	0.0001045
Lab	Fade	CE	0.102	1.3297	2.66E+05	-19.59	269	2	2	0.449376565
		No-CE	0.1842	1.5957	3.19E+05	-30	300	5	3	0.198072627
	Ambient	CE	0.0437	1.3897	2.78E+05	-28.39	201	5	3	0.569562136
		No-CE	1	0	0	-30	300	5	3	0
Rail yard	Noise	CE	0.2232	0.8015	1.60E+05	-20.3	299	3	3	0.084206051
short	Jam	No-CE	1	0	0.00E+00	-30	300	5	3	0

		CE	0.1872	0.966	1.93E+05	-21.01	269	2	3	0.161747078
		No-CE	1	0	0	-30	300	5	3	0
	Hop	CE	0.1733	0.7353	1.47E+05	-20.12	272	2	3	0.19297652
		No-CE	0.8789	0.9298	1.86E+05	-30	300	5	3	0.00011198
	Fade	CE	0.0602	1.3085	2.62E+05	-20.83	274	2	3	0.515006926
		No-CE	0.0352	1.8708	3.74E+05	-30	300	5	3	0.596781704
	Ambient	CE	0.0312	2.2717	4.54E+05	-22.34	234	4	3	0.554830805
		No-CE	1	0	0	-30	300	5	3	0
	Noise	CE	0.0357	0.7987	1.60E+05	-19.77	292	2	2	0.486598423
		No-CE	1	0	0	-30	300	5	3	0
	Jam	CE	0.0892	0.7843	1.57E+05	-19.86	294	2	2	0.447646836
		No-CE	1	0	0	-30	300	5	3	0
	Hop	CE	0.137	0.6261	1.25E+05	-19.59	290	2	3	0.305246056
		No-CE	0.9941	0.0404	8.09E+03	-30	300	5	3	3.01419E-05
Rail yard long	Fade	CE	0.1512	0.4153	8.31E+04	-19.59	72	1	2	0.239191103
Rail yard short no-LOS		No-CE	0.9976	0.7882	1.58E+05	-30	300	5	3	5.68186E-05
	Ambient	CE	0.0743	0.8692	1.74E+05	-20.12	198	2	3	0.475744323

		No-CE	1	0	0	-30	300	5	3	0
	Noise	CE	0.1457	0.5242	1.05E+05	-14.6	95	2	1	0.255251879
		No-CE	1	0	0	-30	300	5	3	0
	Jam	CE	0.1728	0.441	8.82E+04	-14.6	281	1	2	0.169165106
		No-CE	1	0	0	-30	300	5	3	0
	Hop	CE	0.4958	1.0059	2.01E+05	-14.6	294	1	2	0.001842397
		No-CE	0.9522	0.0957	1.91E+04	-30	300	5	3	4.67271E-05
	Fade	CE	0.2083	0.4542	9.08E+04	-19.68	161	1	2	0.098854315
		No-CE	0.8042	2.1693	4.34E+05	-30	300	5	3	0.00019215
	Ambient	CE	0.0358	1.3325	2.67E+05	-21.19	300	2	3	0.525611182
		No-CE	1	0	0	-30	300	5	3	0
	Noise	CE	0.8325	0.3762	7.52E+05	-14.7	146	2	2	0.000132304
		No-CE	1	0	0	-30	300	5	3	0
	Jam	CE	0.757	0.317	6.34E+04	-14.6	28	1	2	0.00017143
		No-CE	1	0	0	-30	300	5	3	0
	Hop	CE	0.458	0.4102	8.20E+04	-14.6	264	1	2	0.002392698
		No-CE	0.959	0.0654	1.31E+05	-30	300	5	3	5.62649E-05
Rail yard long no-LOS	Fade	CE	0.1438	0.4433	8.87E+04	-19.59	270	1	2	0.26570727

As can be seen from the results, in general, the performance of the CE system and the no-CE system for the case with the interferer off is similar. However, the CE system was able to find its solution on its own, whereas the no-CE system had to be designed and tested first.

For the scenario with the interferer off, the no-CE system has a higher throughput and Spectral Efficiency at the cost of also having a higher Software Gain than the CE system.

For the other two scenarios (wide-band noise floor increase and narrow-band noise spike), the CE system was able to find a solution that allowed it to get data across the link, whereas the no-CE system could not operate under the new channel conditions, and therefore, the PER was one. Note that the throughput for the no-CE system under the unfavorable channel conditions is zero because the receiver could not synchronize with the received signal and measure it.

The following graph compares the fitness of each scenario for the CE system:

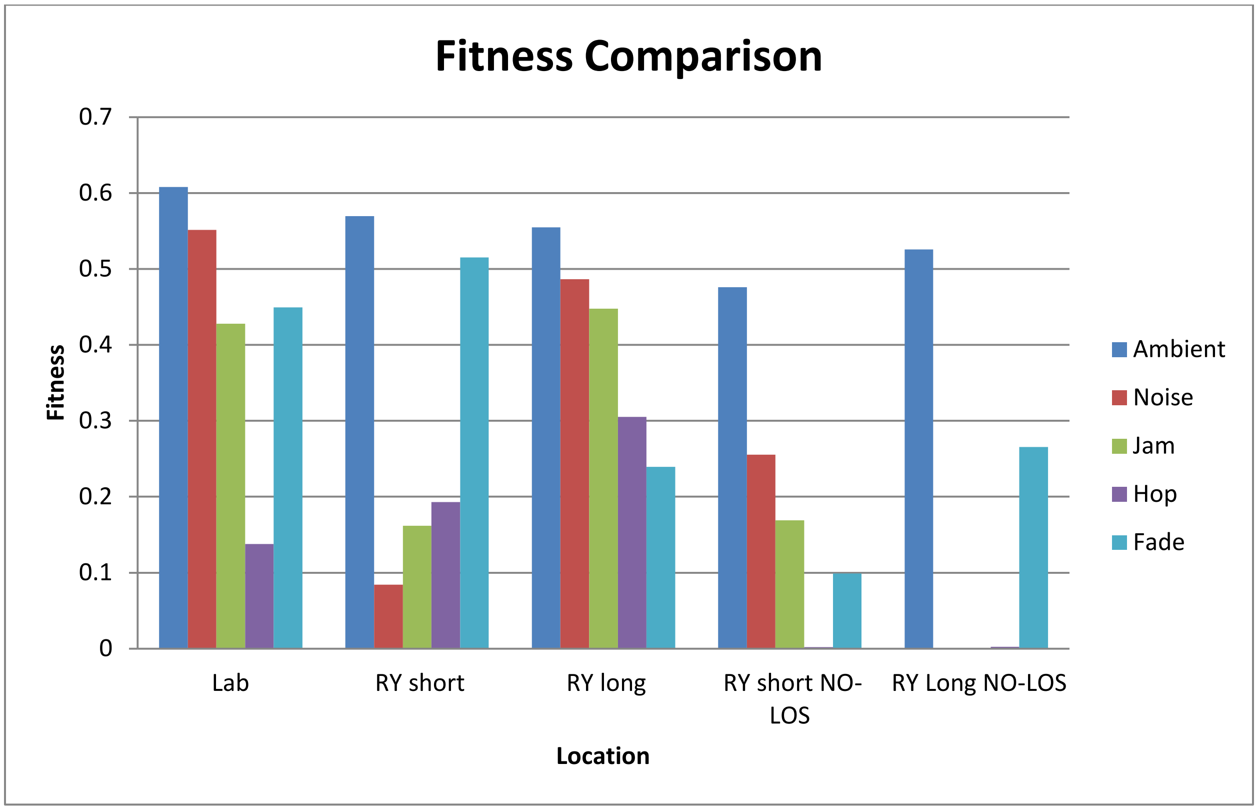


Figure 19. Fitness Comparison

6. Conclusions and Summary of Benefits

The fundamental operations of a CE include observing the spectrum and performance situation, orienting the system based on predefined goals, making a decision, and then acting on this decision by implementing new radio configurations. Results indicate that CE operation is able to mitigate interference conditions in situations where a traditional radio cannot. Although performance does degrade under difficult situations, the CE is able to adapt configuration parameters to still maintain connectivity. This finding has implications for railway safety, efficiency of operations, and revenue stream.

When a locomotive approaches a critical signal, wireless connectivity must be established and track authorizations exchanged and verified. If this connectivity is lost or unable to initiate upon approach, the locomotive should slow down or come to a complete stop. The locomotive should not proceed or proceed slowly until communication is initiated and the transfer of information is completed. Given the time it takes to slow a train down as well as return it to operational speed, this potential delay is significant. Such a delay will impact revenue stream because of the loss of time for the locomotive; additionally, there is the cumulative impact that the delay will have on transfer of goods and on other locomotives waiting to access the same signal.

Similarly, when a locomotive enters the rail yard, a large data transfer typically occurs between the engine and the command center. Here again, outside interference or impeded line-of-sight can impact this download of information. Cognitive approaches to observing performance and adapting radio configuration parameters has strong potential for improving this process, which will lead to increased operational efficiency.

Infrastructure patches to improve wireless connectivity can include increasing the number of fixed repeaters on railway right-of-way. However, the cost associated with such hardware modifications is enormous when one considers construction, grounding, towers, installation, and long-term maintenance. One goal of cognitive operations is to ensure that the automated operations improve performance enough to lessen the need for more costly infrastructure.

This ability to adapt to changing spectrum conditions also ties to carrier interoperability. As spectrum use increases in fixed allocations, such as PTC, carriers will face the same issues that airport travelers see with slow connections at Wi-Fi hot spots. Cognitive techniques can assist in improving the efficient use of open spectrum gaps to increase the density of users. This will certainly be a problem at major distribution yards, such as Chicago. Similarly, interoperability between users who share common goals but use different frequencies is an important requirement for PTC. The reconfigurable nature of software defined radios enables previously stove-piped users to adapt their radio-frequency use to mimic another user. Cognitive approaches will be required to develop this capability and ensure smooth relations between wireless users.

Finally, the safety benefits of maintaining wireless link connectivity or improving the process of establishing a link is foundational to railway operations. Wireless links are proving to be a lynch-pin technology for the future success of PTC. Virtually every diagram of PTC shows some graphic, such as a lightning bolt, to symbolize an assumed wireless link to support PTC. We have shown that connectivity under difficult spectrum environments is far from assumed. A cognitive paradigm has the potential not only to improve link adaptation but to provide a learning mechanism that decreases decision time in similar situations. If this capability assisted in the avoidance of even one major accident at a signalized crossing, the benefits would outweigh the costs.

7. References

[1] B. Le, et al., "A public safety cognitive radio node," 2007.

[2] Y. Zhao, et al., "Development of Radio Environment Map Enabled Case- and Knowledge-Based Learning Algorithms for IEEE 802.22 WRAN Cognitive Engines," in *Cognitive Radio Oriented Wireless Networks and Communications, 2007. CrownCom 2007. 2nd International Conference on*, 2007, pp. 44–49.

[3] J. Hatton and D. Sytsma, "Software Defined Radio (SDR) The Holy Grail for Interoperability," in *Arema*, 2004.

[4] M. W. Ross and J. F. Mao, *Telecommunications Spectrum Use by the Energy, Water and Railroad*. New York: Novinka, 2002.

[5] (2009). *Spectrum solutions for PTC wireless applications*.

[6] S. Alibrahim and T. Tse. FRA research and development program review, signal and train control.

[7] (2006). *MeteorComm's MCC-6100 is World's First Multi-band Software Defined Data Radio*. Available: http://www.meteorcomm.com/news.aspx

[8] MeteorComm. (2007). *MCC-6100 Software Defined Radio*.

[9] E. Research. Universal Software Radio Peripheral (USRP). Available: www.ettus.com

[10] Lyrtech. Lyrtech Small Form Factor SDR. Available: http://www.smallformfactors.com/news/Technology+Partnerships/4581

[11] GNU Radio.

[12] T. Tse. (2003). *FRA Initiatives - PTC*. Available: http://www.ntsb.gov/Events/symp_ptc/presentations/02_Tse.pdf

[13] B. Le, et al., "A Public Safety Cognitive Radio Node," presented at the 2007 SDR Forum Technical Conference, Denver, CO, 2007.

[14] A. He, T. R. Newman, J. Gaeddert, J. H. Reed et al., "Virginia tech cognitive radio network testbed and open source cognitive radio framework," *5th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities and Workshops*, pp. 1–3, 2009.

[15] G. Feng, et al., "Cognitive Radio: From Spectrum Sharing to Adaptive Learning and Reconfiguration," in *Aerospace Conference, 2008 IEEE*, 2008, pp. 1–10.

[16] C. J. Rieser, et al., "Cognitive radio testbed: further details and testing of a distributed genetic algorithm based cognitive engine for programmable radios," 2004, pp. 1437–1443, vol. 3.

[17] M. McHenry, et al., "XG dynamic spectrum access field test results [Topics in Radio Communications]," *Communications Magazine, IEEE*, vol. 45, pp. 51–57, 2007.

Appendix A. Open Access Manual for Train Control Systems

Objective: The objective of this report is to provide relevant and useful information for users of the Rail-CR. This will include the details of its interworkings, as well as the appropriate procedures for communicating with the engine. The details of this interface are presented here in an example implementation using GNU Radio and the USRP RF frontend with WBX daughter cards.

A.1 Introduction

This section will present an overview of the CE and its integration with an SDR. For the purposes of making the most effective use of the CE, a thorough understanding of this integration is critical to choosing where in the processing chain the replacement, vendor-specific radio should go. We demonstrate our application of the CE integrated with Liquid-DSP (Digital Signal Processing), a lightweight, platform-independent SDR library used with the Universal Hardware Drivers (UHDs), and the Universal Software Radio Peripheral 1 (USRP1) device to demonstrate the software's structure and capabilities. A second library called Liquid-USRP is used to further interact with the USRP1.

A.1.1 High-Level Overview

Within an SDR, there are three main components: the software driving the digital signal processing (in this case, Liquid-DSP), the interface with the device (UHD middleware), and finally the hardware itself (USRP). The CE exists on top of these three components, needing only to interact with the SDR, which is assumed to send the proper configuration signals to the hardware. Liquid-USRP will be used as the software entity that interfaces the hardware's dynamic configuration parameters with the engine. By creating a bridge between the two processes, Liquid-USRP and the CE, we effectively control the radio front end from the engine. This is made possible by having control over the transmission properties of each outgoing packet. Liquid-USRP uses a robust and reliable header transmission to contain the information about the payload of the packet. This provides a convenient trade-off between using a robust transmission scheme for important metadata information and a more efficient one for user data. Here, we discuss how the engine interacts with the proper system components to push new transmission parameters to this header information.

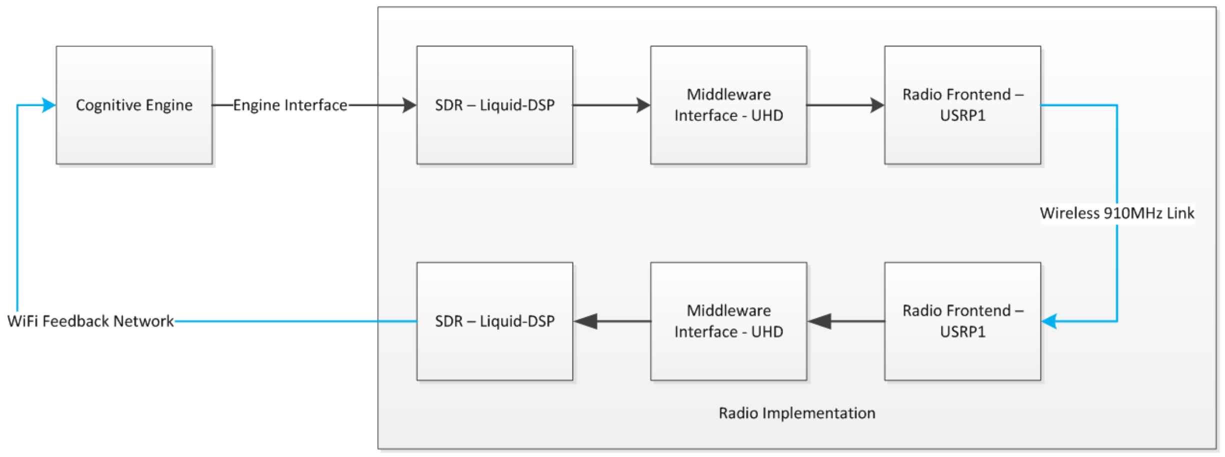


Figure 20. CE Interaction

There are, however, implications to the engine for replacing the assumed SDR setup because the CE relies on information fed back from the receiver. Information is currently being fed back through the wireless feedback network, which assumes a reliably linked connection. A more detailed discussion of these considerations can be found in Section 7.3**Error! Reference source not found.**.

Figure 20 shows that the engine requires two different interfaces to the radio implementation, one to push transmission parameter changes to the SDR interface, and the other to collect performance metrics from the receiving side. These interfaces were designed as two functions, `radio_pushKnobs` and `radio_getMeters`. The details of the code used will be discussed in Section 7.2. The four knobs available for the engine to change are transmission gain, modulation scheme, forward error correction scheme, and packet size. Changes in these parameters can be pushed multiple ways but are dependent on the radio's controlling process. The four metrics required for the radio to properly estimate that environment's effect on the transmission parameters are SNR, packet error, throughput, and spectral efficiency. The metrics can be collected in a similar fashion as the knobs are being pushed, but this is, again, up to the design of the radio implementation and what capabilities of process communication are available.

A.1.2 Interprocess Communication: Socket Interfaces

On most computers, there are several options for process communications (common files, pipes, message queues, etc.). These options can depend on the operating system (OS) being used, but are mostly platform independent. The most widely known method is sockets because of its use in almost all commercial networking operations. These can be used not only across networks and devices but also as inherent mechanisms to deal with message passing between processes on the same computer. This held a great advantage because it allowed us to set up a communications system that could be used not only on the same computer but also across an ad-hoc wireless connection, enabling the seamless integration of each computer process into a collective network by simply changing the connection IP address.

Previous research has focused on a communications architecture that involves a software *Broker*. Object request Brokers have been used to simplify component interface by using a central entity to provide a means of common connection between various components that may need to talk to each other. Because it is expensive in terms of design time, developing a specific protocol interface for every component on the system is not cost-effective when the same interface can be used. For our application, we developed an Interprocess Communications Broker (IPCBroker) that ran parallel to all processes in the system. The rest of the system is shown in the diagram below.

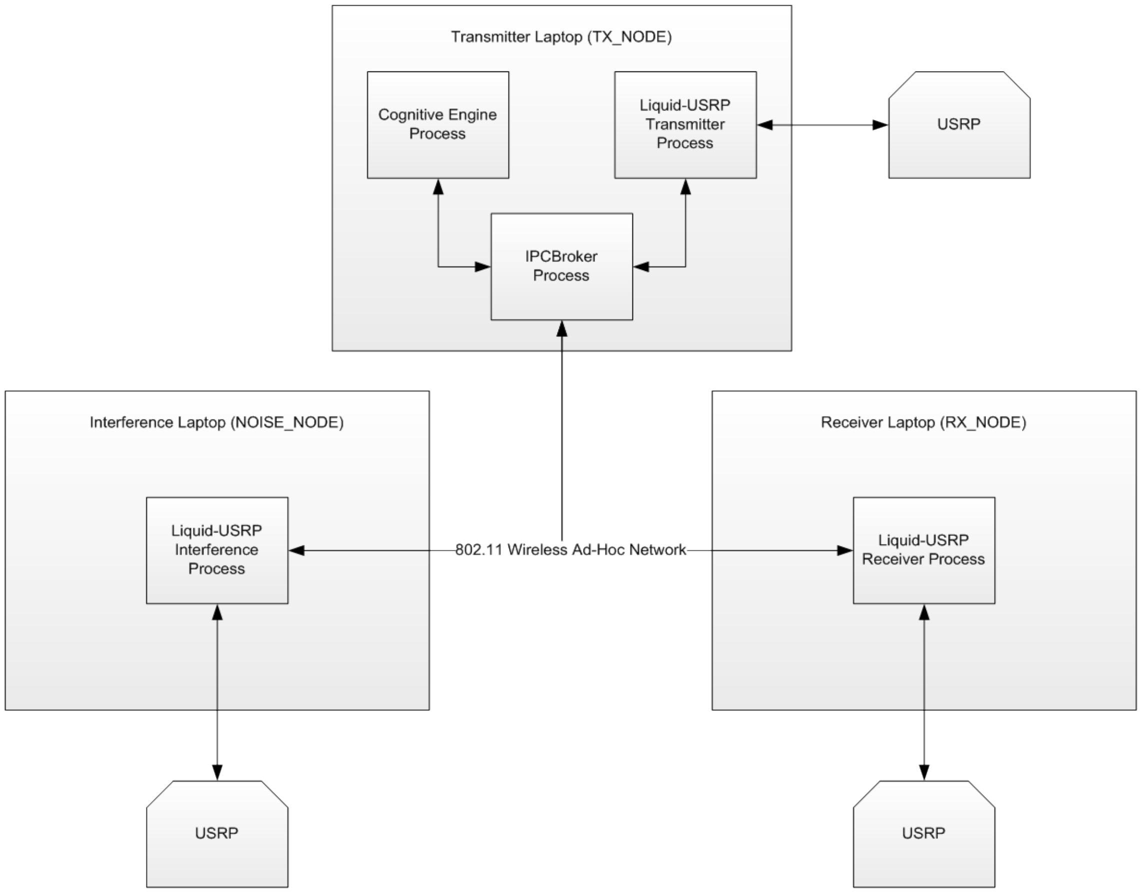

Transmitter Laptop (TX_NODE)
Cognitive Engine Process
Liquid-USRP Transmitter Process
USRP
IPCBroker Process
Interference Laptop (NOISE_NODE)
Receiver Laptop (RX_NODE)
Liquid-USRP Interference Process
802.11 Wireless Ad-Hoc Network
Liquid-USRP Receiver Process
USRP
USRP

Figure 21. System Interaction

Each process in the system is connected through the IPCBroker. The Broker's job is to take messages from a given process to its appropriate destination based on the message type. For example, if the Broker receives a METERS_REQUEST, it would know to forward the message to the receiver laptop. Most networked systems implement some kind of addressing system; however, to reduce the distributed maintenance cost of the common code, most of the functionality of an addressing system can be inherited through the types of messages the Broker receives. This puts all addressing functionality into the Broker, making it easier to develop the Broker alone, as opposed to changing the common header file of each process, pushing the changes to each node, recompiling, and then testing. This removes some ease of extendibility, but because this system is specific to our application and would be replaced with a vendor specific SDR, this tradeoff is acceptable.

A.1.3 Mex Files
Mex files are used throughout the system to interact MATLAB with underlying C++ code. This is used for two reasons; first, MATLAB has no inherent mechanism to interface with Sockets but does provide a mechanism to interface with C++, greatly increasing its extendibility to most things which work using the C/C++ language. By adhering to some guidelines used for Mex functions, most C++ code can be used just as if it were a function written in MATLAB code. The basic shell of a Mex file is provided in 0, and the functions used in this application are found in 0.

A.2 MATLAB CE Interface
In the MATLAB engine, the metrics collection is an integral part because its estimations are calibrated using that feedback. After going through the cognition loop, a different set of knobs may be chosen, which then need to be pushed to the SDR controlling process. In our case, this would be done through the IPCBroker, which would forward the message to the USRP controlling process, applying the decision change. Below, we discuss two methods of interfacing with the Rail-CR; we also discuss how these methods may be applied to a different SDR platform.

A.2.1 Interfacing with the Rail-CR: `radio_getMeters`
This implementation of the Rail-CR is based entirely in MATLAB, a well-established numerical computing software suite that provides a wide range of software tools to analyze difficult tasks. It is written in the MATLAB-specific programming language known as MATLAB (using a .m file extension). Within the code, we assume that the function `radio_readMeters` will provide the required meters to the engine. Within our system, we have code that allows for a switch between a simulation model and a hardware model depending on a global configuration matrix. This matrix can be found in `getParams.m`, which contains a number of configuration parameters that can affect the system. Given that the matrix allows for the hardware configuration, getParams.m configures the appropriate data structures to a specific template, the one for the meters structures shown below.

```
metersFormat = struct('SNR', NaN,            ...
                      'PER', NaN,            ...
                      'SpectEff', NaN,       ...
                        'ThroughPut', NaN);
```

Filing this structure is the only requirement of the `radio_getMeters` function. For our implementation, we used the aforementioned Broker architecture based on a Unix Sockets implementation in C++. To interface the CE with the rest of the system, multiple C++ functions were created to control the system from startup to shutdown. Each function makes use of the underlying code included in the project, all found in the header file through the Hardware/Socket_IO/FRA.h path. This system, however, uses a call to a Mex file, called `meters_request()`, which generates the proper message to send to the Broker, and thus the receiver. Once the request is placed, the Mex function waits on a response, throwing an exception if an error occurs. For a replacement SDR system, this is the point in the code that needs to be replaced. It is assumed that the process communications will be different and that this is the point where the interface to the engine will change for vendor-specific applications. Again, it is only assumed that this function will populate the structure and return it, regardless of implementation.

A.2.2 Interfacing with the Rail-CR: `radio_pushKnobs`

The other aspect to the radio's interface is the functionality of the `radio_pushKnobs` function. This function is called from the main engine when it has decided on a different set of transmission parameters. The expectation with this function is that when the main cognition loop of the radio calls it, the parameter changes will be applied to the hardware. The CE expects the next call to `radio_getMeters` to reflect an updated set of meters, based on the changed parameters. This synchronization can be guaranteed several different ways and is discussed in Section 7.3.2. In our implementation, we use another Mex function called `knobs_push`, similar to the `meters_request` function used for `radio_getMeters`, to send the configuration to the hardware.

Within the engine code, because environmental conditions are variable, we use a conversion function for the power range. To the GA and its estimation function, it acts on a power range from 0 to 100; however, this percentage is meaningless to channel estimation functions, which use SNR to calculate BER based on transmission parameters and environmental conditions. To switch between the two, we use two functions, `PwdB2Ratio` and `PwRatio2dB`. By modifying this function, accurate estimations can be made depending on the upper and lower power capabilities of the radio. The function `PwRatio2dB` is used here in `radio_pushKnobs` to convert the GA's decision on its percentage scale back to a USRP understandable dB range.

The configuration of the engine also requires the specification of the modulation and coding schemes supported, which are outlined in Radio/radioDefinition.m file. For each modulation scheme, a modulation type (Phase Shift Keying (PSK), Quaternary PSK (QPSK), Quaternary Amplitude Modulation QAM, Square QAM (SQAM), etc.) must be named along with a bit depth (bits/symbol). For each modulation type and bit depth, each higher index in the configuration vector is assumed to be an ascension in system performance. For our implementation, we used BPSK with 1 bit/symbol up through QAM with 6 bits/symbol. More bits per symbol naturally translate to more usefulness within the system as throughput is increased with each bit added. This, of course, has a natural tradeoff during poor channel conditions and is one of the objectives the CE is trying to carefully balance. The `modVec` vector is populated accordingly in the aforementioned `Radio/radioDefinition.m` file along with the supported forward error correction coding schemes available. Similarly to the power ratio, the actual names and depths mean nothing to the GA itself; it needs an index from which to choose the respective scheme for each. The index to these vectors acts as that selection mechanism; the GA itself chooses indexes in its evolution cycles but translates them to the appropriate modulation and coding in its objective function to accurately determine the expected response from the channel.

These conversions were used to simplify the GA's implementation because of the complex bit operations the evolutionary algorithms were expected to perform on its chromosomes. They are detailed here to give the future user insight into how to appropriately configure the software to adapt according to the new radio's hardware capabilities. More details are discussed in Section 7.3.1. Once these conversions are made, they are packaged similarly to a meters_request call and sent to the IPCBroker to be forwarded to the transmitter process for hardware configurations. This is where the code has the greatest chance of diverging given that the interprocess communication mechanism has changed. Even if C++ sockets were still used, a change in the protocol would require a rewriting of the Mex function to interact with the rest of the network properly.

A.3 Considerations

A.3.1 Modulation and Coding Schemes

The specific modulation and coding schemes supported by the current engine can be found in Radio/radioDefinition.m and are included in 0. These are the schemes that are made available to the GA's objective function where it estimates channel conditions and their impact on the transmission parameters. Although these schemes are generally accepted as common, different radios may support modulations and coding different from those currently available in the engine's implementation. To change this, one must first redefine the `modeVec` vector found in Radio/radioDefinition.m and then change the estimation function that is required of the GA.

A.3.2 Feedback Collection

As discussed earlier, feedback collection can be done in a variety of ways but does have some requirements specific to this implementation. The engine requires that after a knobs_push, the following meters_request will show the change in performance metrics caused by the change in transmission parameters. For this implementation, we use a windowed framework through which the receiver collects metrics on every packet it receives. However, this may not guarantee an accurate measurement on a packet-by-packet basis. For example, if a receiver cannot properly demodulate a packet's payload, because of a low transmission power or too much interference, the packet may effectively be considered dropped, giving a BER of 1. If, on the next packet, the transmitter appropriately increases transmission power, and the receiver successfully receives the payload without loss, the effective BER is 0. This demonstrates the need for averaging over some number of samples; in our case, we use an average of 50 packets for each meter request. As long as the receiver is able to decode the header, the packet information is stored, regardless of the payload validity. This allows the receiver to collect performance measurements as a ratio of good packets to bad packets. When a meter_request message comes to the receiver, it averages the information within the 50 packet window and sends the information back. Given that this window is not full, it either waits for it to fill or times out after 3 seconds. These numbers for the timeout and packet windows size were chosen on the basis of empirical experimentation. When the CE sends a knob_push, there is no guarantee that the 50 packet window has metrics that will reflect only the newly pushed transmission parameters; thus, we created a new function, `radio_flushMeters`, to clear this window out after the decision has been applied. This function makes use of the Mex function, `meters_flush`.

It is worth noting that the engine relies on fresh metrics after a `radio_pushKnobs` call. Where the synchronization happens is up to the user of the new radio. Here, however, we demonstrated it in the functionality of the `radio_pushKnobs` Mex file. The future user may decide to have the engine collect statistics constantly and average them when needed, or may have the transmitter collect metrics through the header information of packets sent from the receiver to the transmitter. These are just a few of the options the designer has.

A.3.3 Wrap-up/Conclusion

The architecture used in the testing results is presented. By showing the interaction of the engine with the rest of the system and discussing the different aspects of the system, the user should be able to replace the radio front end. The main interface functions, radio_pushKnobs and radio_getMeters, are discussed to help the user know the functionality of these methods. Important considerations such as changing possible coding and modulation schemes are discussed, as well as what the engine expects in terms of feedback from the system.

Appendix B. Radio/radioDefinition.m

```
%% Radio definition
% Defines the knobs and meters to be used by the Rail-CR
global knobsFormat knobRanges defaultKnobs metersFormat utilityFormat;
global codeVec modVec;

% Knobs
%     Transmit Power    dB
%     Packet Size       bytes
%     Modulation Type   Defined by ModulationTypes
%     Coding Types      Defined by CodingTypes
knobsFormat = struct('TxPower', NaN,        ...
                     'PacketSize', NaN,     ...
                     'ModulationType', NaN, ...
                     'CodingType', NaN);
% Meters
%     SNR                 Relative dB difference
%     PER                 Packet Error Rate
%     SpectEff            Spectral Efficiency in bits/second/Hz
%     ThroughPut          Throughput in bits/second
metersFormat = struct('SNR', NaN,           ...
                      'PER', NaN,           ...
                      'SpectEff', NaN,      ...
                      'ThroughPut', NaN);

% Utilities
utilityFormat = struct('Fitness', NaN,                    ...
                       'KnobUtilities', knobsFormat,    ...
                       'MeterUtilities', metersFormat);

Range = struct('Min', NaN, 'Max', NaN);
knobRanges = struct('TxPower', Range, 'PacketSize', Range,  ...
                    'ModulationType', Range, 'CodingType', Range);

% Modulation mapping
modMap = struct('Type', 0, 'Depth', 0);
modVec(1).Type = ModulationTypes.LIQUID_MODEM_BPSK;
modVec(1).Depth = 1;
modVec(2).Type = ModulationTypes.LIQUID_MODEM_QPSK;
modVec(2).Depth = 2;
modVec(3).Type = ModulationTypes.LIQUID_MODEM_PSK;
modVec(3).Depth = 3;
modVec(4).Type = ModulationTypes.LIQUID_MODEM_QAM;
modVec(4).Depth = 4;
modVec(5).Type = ModulationTypes.LIQUID_MODEM_SQAM32;
modVec(5).Depth = 5;
modVec(6).Type = ModulationTypes.LIQUID_MODEM_QAM;
modVec(6).Depth = 6;

% Coding mapping

% Reed-Solomon
```

```
codeVec(1) = CodingTypes.LIQUID_FEC_RS_M8;
% Hamming
codeVec(2) = CodingTypes.LIQUID_FEC_HAMMING74;
% No coding
codeVec(3) = CodingTypes.LIQUID_FEC_NONE;

% Knob Extremes
knobRanges.TxPower.Min = 0;     % Smallest
knobRanges.TxPower.Max = 90;
knobRanges.PacketSize.Min = 20;
knobRanges.PacketSize.Max = 300;       % Largest
knobRanges.ModulationType.Min = 1;
knobRanges.ModulationType.Max = 6;
knobRanges.CodingType.Min = 2;
knobRanges.CodingType.Max = 3;



% This is used exclusively in distance calculations
% TODO: Should probably move Beta goals into this file and source from
% there
meterKnees(1) = (10^(17/10) - 10^(-2/10));
meterKnees(2) = 1;
meterKnees(3) = 3 - 0.1;
meterKnees(4) = 1e6 - 1e5;

% Needs to match largest and smallest value of any range for GA
knobRanges.LargestValue = knobRanges.PacketSize.Max;
knobRanges.SmallestValue = knobRanges.TxPower.Min;

defaultKnobs = knobsFormat;
% % In matlab, transmit power ranges from 0 - 100. This is different in
% % hardware however, as it is in dBs. It gets converted automatically in the
% % radio_pushKnobs function
defaultKnobs.TxPower            = 50;
defaultKnobs.PacketSize         = 150;
defaultKnobs.ModulationType     = 3;
defaultKnobs.CodingType         = 3;
```

Appendix C. Available Mex Functions in Rail-CR Implementation

```
/*
 * Function: generate_noise()
 *           This function is used to actually start the noise transmission.
 *           The noise_start function starts the process to wait and listen for
 *           this command, which will then start transmitting a waveform
designated
 *           by its startup command.
 *
 * Inputs:
 *           None.
 *
 * Outputs:
 *           None.
 *
 * Usage:
 *           generate_noise()
 *
 */


/*
 * Function: knobs_push(power, packetSize,
 *                                 modScheme, modDepth, codeScheme)
 *
 *           Knobs_push pushes a new set of knobs to the transmitter
 *           controller. These values are expected to have a format that
 *           will be understandable to the controlling SDR process. These
 *           values should be correctly defined in Radio/radioDefinition.m
 *           or ensured of correct formatting.
 *
 * Inputs:
 *           power_dB - Updated gain setting for the transmitter (dB)
 *           packetSize_bytes - Size of the user payload (bytes)
 *           modScheme - Modulation scheme (BPSK, QAM, SQAM, etc.)
 *           modDepth - Bits per symbol (1, 2, 3 etc.)
 *           codeScheme - Forward error correction coding
 *                        (None, Hamming74, Reed-Solomon, etc.)
 *
 * Outputs:
 *           None.
 *
 * Usage:
 *           knobs_push(power_dB, packetSize_bytes, modScheme, modDepth,
codeScheme)
 *
 */


/*
 * Function: meters_flush()
 *           Meters flush is used to flush the current window of the receiver.
That
 *           means regardless of whether the window is full or not full, the
receiver
 *           should disregard all averaged data it currently has in its window and
start
```

```
 *           with a new window. This is useful after a knob_push to ensure
accuracy is
 *           reported for the right set of transmission parameters.
 *
 * Inputs:
 *           None.
 *
 * Outputs:
 *           None.
 *
 * Usage:
 *           meters_flush();
 *
 */


/*
 * Function:
 *          [ SNR
 *           TotalBER
 *           AggPacketER
 *           CurPacketER
 *           RSSI
 *           PacketNum
 *           SpectEff
 *           ThroughPut
 *           GoodPut
 *           AggGoodPut ] = meters_request()
 *
 *           A request for the receivers metrics. When the receiver gets
 *           this message, it collects the statistics of its current window
 *           and returns it to the engine. This call blocks.
 *
 * Inputs:
 *           None.
 *
 * Outputs:
 *     SNR - Signal to noise ratio averaged over the run
 *      TotalBER - The average BER over the run
 *      AggPacketER - The aggregate packet error rate over the
 *           entire run. PER is measured via a rate of number of
 *           packets passing the CRC over the number that didn't.
 *      CurPacketER - The average PER of the last 50 packets. If
 *           numPackets < 50, then numPackets worth.
 *      RSSI - Received Signal Strength Indication at the receiver.
 *           PacketNum - The last packet number associated with the
 *           data sent back within this meters_response.
 *      SpectEff - Spectral efficiency (b/s/Hz)
 *      ThroughPut - Data rate (including erroneous bits) at
 *           receiver (b/s)
 *      GoodPut - The rate of good bits at the receiver over the
 *           length of the entire run (b/s)
 *     AggGoodPut - The goodput of the last 50 packets. Similar
 *           to AggPacketER

 *
 * Usage:
 *     [a b c d e f g h i j] = meters_request();
 *
 */
```

```
/*
 * Function:
 *          [ SNR
 *            TotalBER
 *            AggPacketER
 *            CurPacketER
 *            RSSI
 *            PacketNum
 *            SpectEff
 *            ThroughPut
 *            GoodPut
 *            AggGoodPut ] = meters_request_final(numPackets)
 *
 *            This function was used during testing for the purposes of
 *            collecting metrics over a specified number of packets.
 *            Upon receiving the request, the receiver would average
 *            the statistics over the number of packets requested and
 *            send them back.
 *
 * Inputs:
 *            numPackets - Packet number the receiver should wait for
 *            until return the statistics of the run.
 *
 * Outputs:
 *      SNR - Signal to noise ratio averaged over the run
 *       TotalBER - The average BER over the run
 *       AggPacketER - The aggregate packet error rate over the
 *            entire run. PER is measured via a rate of number of
 *            packets passing the CRC over the number that didn't.
 *       CurPacketER - The average PER of the last 50 packets. If
 *            numPackets < 50, then numPackets worth.
 *       RSSI - Received Signal Strength Indication at the receiver.
 *            PacketNum - The last packet number associated with the
 *            data sent back within this meters_response.
 *       SpectEff - Spectral efficiency (b/s/Hz)
 *       ThroughPut - Data rate (including erroneous bits) at
 *            receiver (b/s)
 *       GoodPut - The rate of good bits at the receiver over the
 *            length of the entire run (b/s)
 *      AggGoodPut - The goodput of the last 50 packets. Similar
 *            to AggPacketER
 *
 * Usage:
 *      [a b c d e f g h i j] = meters_request_final(100);
 *
 */


/*
 * Function: noise_stop()
 *            This function will send a message to the intereferer alerting it that
the
 *            noise it is transmitting should cease. It does not block and does not
 *            case the noise process to exit.
 *
 * Inputs:
 *            None.
 *
 * Outputs:
 *          None.
 *
 * Usage:
 *          noise_stop();
 *
 */


/*
 * Function: receiver_stop()
 *          Effectively stops the receiver, it blocks waiting for a response from
the
 *          receiver to ensure it has gracefully shutdown.
 *
 * Inputs:
 *          None.
 *
 * Outputs:
 *          None.
 *
 * Usage:
 *          receiver_stop();
 *
 */


/*
 * Function: system_stop()
 *          Shuts down the system by connecting to the Broker with the
 *          shutdown signal active, effective a true bool type being passed in.
 *          Once the connection is made, the broker sends the shutdown message
 *          to all components in the system.
 *
 * Inputs:
 *          None.
 *
 * Outputs:
 *          None.
 *
 * Usage:
 *          system_stop();
 *
 */


/*
 * Function: transmitter_running()
 *          This function is a simple request sent to the IPCBroker to check
whether
 *          the transmitter is still transmitting. Because the engine runs
 *          asynchronously it uses this function to ensure the transmitter is
 *          still accepting knob pushes.
 *
 * Inputs:
 *          None.
 *
 * Outputs:
 *          None.
 *
 * Usage:
 *          transmitter_running();
 *
 */


/*
 * Function: transmitter_stop()
 *          This function sends the appropriate message to the IPCBroker to
 *          tell the transmitter to stop transmitting. This will end the
 *          transmission and stop the transmitter process.
 *
 * Inputs:
 *          None.
 *
 * Outputs:
 *          None.
 *
 * Usage:
 *          transmitter_stop();
 *
 */


/*
 * Function: wait_for_broker_ready()
 *          This function ensures that the cognitive engine has the ability to
contact
 *          the IPCBroker. Since we allow the CE to control the system from here
it is
 *          crucial that this connection can be established before anything is
tested.
 *
 * Inputs:
 *          None.
 *
 * Outputs:
 *          None.
 *
 * Usage:
 *          wait_for_broker_ready();
 *
 */


/*
 * Function: wait_for_noise_ready()
 *          This function implements a simple blocking function for the system.
It
 *          is assumed to be used within the noise_start() function so that the
system
 *          allows the interferer time to allocate memory, prepare data
structures and
 *          the like for its DSP.
 *
 * Inputs:
 *          None.
 *
 * Outputs:
 *          None.
 *
 * Usage:
 *          wait_for_noise_ready();
 *
 */


/*
 * Function: wait_for_rx_stop()
 *          This function implements a simple blocking function for the system.
It
 *          is assumed to be used within the receiver_stop() function so that the
system
 *          allows the receiver to accept the packets that may still be in
transit in
 *          order to allow for a graceful shutdown of all nodes in the system.
 *
 * Inputs:
 *          None.
 *
 * Outputs:
 *          None.
 *
 * Usage:
 *          wait_for_rx_stop();
 *
 */

/*
 * Function: wait_for_rx_stop()
 *          This function implements a simple blocking function for the system.
It
 *          is assumed to be used within the receiver_start() function so that
the system
 *          allows the receiver to prepare for incoming packet traffic so the
transmitter
 *          doesn't start losing packets because the receiver is not ready.
 *
 * Inputs:
 *          None.
 *
 * Outputs:
 *          None.
 *
 * Usage:
 *          wait_for_rx_stop();
 *
 */
```

Appendix D. Example Mex File – timestwo.cpp

```
/*
 * Function: timestwo(a, b, ...)
 *
 * Inputs:
 *           One or more numbers which will be multiplied by two. (a, b, ...)
 *
 * Outputs:
 *           The same number of input arguments with their values multiplied by
two.
 *           [a*2 b*2 ...]
 *
 * Usage:
 *           [a b c] = timestwo(a, b, c)
 *
 */
#include "mex.h"
void mexFunction(int nlhs, mxArray *plhs[],
    int nrhs, const mxArray *prhs[])
{
 int i, j, m, n;
 double *data1, *data2;
 if (nrhs != nlhs)
 mexErrMsgTxt("The number of input and output arguments must be the same.");


 for (i = 0; i < nrhs; i++)
   {
    /* Find the dimensions of the data */
    m = mxGetM(prhs[i]);
    n = mxGetN(prhs[i]);


    /* Create an mxArray for the output data */
    plhs[i] = mxCreateDoubleMatrix(m, n, mxREAL);


    /* Retrieve the input data */
    data1 = mxGetPr(prhs[i]);


    /* Create a pointer to the output data */
    data2 = mxGetPr(plhs[i]);


     /* Put data in the output array */
    for (j = 0; j < m*n; j++)
    {
    data2[j] = 2 * data1[j];
    }
   }

}
```