Implementation and On-orbit Testing Results of a Space Communications Cognitive Engine

Timothy M. Hackett, Sven G. Bilén, *Senior Member, IEEE,* Paulo Victor R. Ferreira, Alexander M. Wyglinski, *Senior Member, IEEE,* Richard C. Reinhart, and Dale J. Mortensen

***Abstract*—Cognitive algorithms for communications systems have been presented in literature, but very few have been integrated into a fielded system, especially space communications systems. In this paper, we describe the implementation of a multi-objective reinforcement-learning algorithm using deep artificial neural networks acting as a radio-resource-allocation controller. The developed software core is generic in nature and can easily be ported to another application. The cognitive engine algorithm implementation was characterized through a series of tests using both a ground-based system and a space-based system. The ground system provided engineering-model software-defined radios, commercial modems, and RF equipment emulating the targeted space-to-ground channel. The on-orbit communication system, including a space-based, remotely controlled transmitter, resides on the International Space Station and operates with a ground-based receiver at NASA Glenn Research Center. Through a series of on-orbit tests, the cognitive engine was tested in a highly dynamic channel and its performance is discussed and analyzed.**

***Index Terms*—cognitive engine; neural networks; reinforcement learning; SCaN Testbed; space communications; machine learning.**

I. Introduction

COMPUTING performance continues to increase with every new generation of general purpose processors (GPPs), graphics processing units (GPUs), and field-programmable gate arrays (FPGAs). This makes the use of software-defined radios (SDRs) increasingly feasible for space communications applications [1], [2]. Cognitive communication algorithms have been proposed in the past, but many of those implemented and tested often have employed dynamic spectrum access techniques or other sensing approaches. These new technologies strive to achieve multi-objective goals by modifying multiple radio parameters through rewarded behavior and provides the foundation for deploying cognitive systems, especially for autonomous space applications.

The International Space Station (ISS) currently hosts three SDRs on a research platform called the Space Communications and Navigation (SCaN) Testbed [3]. NASA is interested in characterizing the applicability of SDRs for future missions in terms of new operational aspects, new software development paradigms (as opposed to building specialized hardware), and the ability to change the signal waveforms [4]. This paper describes the implementation and testing of a ground-based space communications cognitive engine (CE) that controls an S-band radio on the SCaN Testbed along with ground SDRs and commercial modems in order to test the proposed multi-objective reinforcement learning (MORL) using deep neural networks algorithm for use as a radio-resource-allocation controller outlined in [5].

NASA's SCaN Testbed experimental state-of-the-art communication system consists of an adaptive coding and modulation scheme, which changes the transmitter parameters through the use of a lookup table with closed-loop feedback. The ground receiver senses the signal-to-noise (SNR) or E_s/N_0 (energy per symbol to noise power spectral density ratio) level and then looks at a pre-generated table that provides the optimal modulation-coding pair (MODCOD) for that given signal quality. The chosen MODCOD is then sent through a control channel back to the space transmitter to update its properties for the next transmitted frame. This table is generally built to maintain a quasi-error-free performance in an additive white Gaussian noise (AWGN) channel. The table is developed by a team of experts that run series of tests to characterize the radio's performance. This type of adaptive communications is a bi-objective radio-resource-allocation optimization, in which the two balancing objectives are throughput and frame error rate (FER). The authors' proposed reinforcement-learning neural network (RLNN) algorithm in [5] is, in a sense, the generalization of the developed adaptive communications model, allowing for more flexibility in terms of adaptation and achievement of multiple goals at the cost of computational power. Instead of just balancing between minimizing FER and maximizing throughput, the proposed algorithm can handle any number of objectives, such as frame error rate (FER), throughput, occupied bandwidth, spectral efficiency, transmit power efficiency, and DC power consumption. Likewise, in addition to just the MODCOD, our proposed RLNN algorithm has the ability to change the transmit power, filter roll-off factor, and symbol rate to meet these objectives. In contrast to the static table used in the adaptive architecture, the CE *learns* (using reinforcement learning) the optimal transmission parameters (referred to as "action tuples") given the sensed E_s/N_0.

To demonstrate the RLNN algorithm within a space communications architecture, we have chosen to leverage the existing

T. Hackett and S. Bilén are with the School of Electrical Engineering and Computer Science, The Pennsylvania State University, University Park, PA 16801, USA, {Email: tmh5344@psu.edu, sbilen@psu.edu}

P. Ferreira and A. Wyglinski are with the Department of Electrical Engineering & Computer Engineering, Worcester Polytechnic Institute, Worcester, MA 01609, USA, {Email: prferreira@wpi.edu, alexw@wpi.edu}

R. Reinhart and D. Mortensen are with NASA John H. Glenn Research Center, Cleveland, OH 44135, USA, {Email: richard.c.reinhart@nasa.gov, dale.mortensen@nasa.gov}

Manuscript received ; revised .

978-1-5386-3988-7/17/$31.00

architecture used in [6] since it decreased the development time and risk. The adaptive workstation that provides the lookup table was replaced with a workstation containing our RLNN cognitive algorithm. As in [6], our implementation of the RLNN algorithm was designed to be operate with the Digital Video Broadcasting–Satellite–Second Generation (DVB-S2) standard [7], a widely available commercial broadcast satellite waveform with a bank of MODCODs available to use as link conditions allow. It supports both variable coding and modulation (VCM) and adaptive coding and modulation (ACM) with an on-the-fly MODCOD and roll-off adaptations. Unfortunately, the waveform does not support on-the-fly symbol rate changes, which means the implementation and test of the CE did not include an adaptive symbol rate.

This paper provides details on porting the RLNN MATLAB algorithms presented in [5] into a fielded system, its characterization on the ground, and its performance in a real-world, challenging space-to-ground channel. Section II provides an overview of the mechanics of the CE. Section III provides a summary of the software implementation and architecture as first discussed by the authors in [8]. Section IV describes the characterization of the CE using a ground system emulating the ISS-to-ground channel and using engineering-model SDRs. Section V provides the on-orbit experiment testing architecture, test schedule, and key analyses from the data gathered. Finally, Section VI provides a discussion of the results and provides many areas to be improved upon in future work.

II. Overview of Cognitive Engine

The theoretical background and preliminary simulations for the multi-objective RLNN CE are described in detail in [5], which builds upon [9], [10]. This section provides a brief introduction to the algorithms to provide context for the reader with respect to the remaining portion of this paper.

The RLNN is a reinforcement learner that leverages neural networks for exploitation (as substitute for the RL Q-table) and exploration (for "virtual" exploration [5], [9]). The process flow of the algorithm is shown in Fig. 1. On each iteration, the CE can choose to exploit previous knowledge or explore new actions. The set of exploit NN ensembles takes in an input of the normalized performance values of each objective as well as the current E_s/N_0 and outputs the action that corresponds to that performance given the E_s/N_0. Each exploit NN ensemble outputs one of the elements in the action tuple (modulation scheme, encoding rate, filter roll-off factor, and transmit power).

The explore NN ensemble takes in an action tuple and outputs the fitness score weighted by the relative importance of each objective function, that comprise a set of conflicting goals for a certain communications mission. Each possible action (or a subset of the entire action space) is sent to the explore NN ensemble to predict the performance of each action. The performance values are thresholded into two bins: the "good" actions (which result in good performance) and the "bad" actions (which result in poor performance). A random action (uniformly distributed) is then chosen from one of these

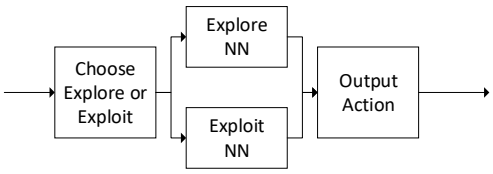


Fig. 1. General flow block diagram of the RLNN (adapted from [5]). On each iteration, the action parameters are updated either using virtual exploration or exploiting learned knowledge.

bins based on the rejection probability hyperparameter. The rejection probability enables good actions to be chosen most of the time but allows for bad actions to be chosen occasionally. The rationale why the good actions are not always chosen is because the knowledge of the environment model in the NN is not complete—what it predicts as a bad action may actually be a good action. This is known as virtual exploration [5], [9].

The action parameters are sent to the DVB-S2 transmitter, which changes its transmission parameters on the next frame. The E_s/N_0 and measured performances of FER, throughput, spectral efficiency, DC power consumption, and transmit power efficiency are recorded to a training buffer. When the training buffer fills up over time, the explore and set of exploit NN ensembles are retrained using the Levenberg–Marquardt backpropagation algorithm [11] based on the set of data in the training buffer [5].

III. Software Implementation

The software architecture for this CE was first introduced by the authors in [8]. This section provides a summary of the architecture as it applies to the ground testing and on-orbit testing results presented in later sections.

The performance of the proposed RLNN algorithms were assessed with MATLAB simulations, as described in [5], for proof-of-concept purposes. For experimentation with a real-world flight system, the CE was developed using an object-oriented architecture designed for performance, generality, and reuse. The CE was developed in C++ leveraging some of the constructs in the C++11 standard [12] and many software open-source libraries to maximize performance and provide more capability to port the engine to different (possibly embedded) platforms. The software was developed in a UNIX environment, but designed to be recompiled for any operating system. Software libraries (described in Section III-B) were used for the NNs, matrix algebra, external interfacing to the modems, and saving/restoring program sessions.

A. Licensing

For our CE, the software libraries chosen were limited to those with "permissive" licenses such as the Berkeley Software Distribution (BSD) license [13]. These types of licenses allow the use and redistribution of one's work without releasing the source code (but still require maintaining copyright notices). In effect, this allows any proprietary application to use the licensed work without having to release the source code. In contrast, "copyleft" licenses, such as the GNU General

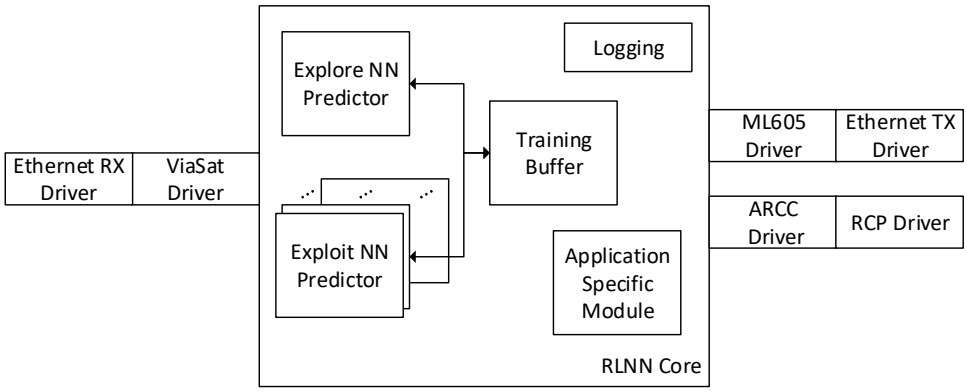


Fig. 2. Implemented software architecture of the RLNN. The RLNN Core hosts the RLNN algorithms; the drivers interface with the various receivers and transmitters.

Public License (GPL) [14] and the GNU Lesser General Public Licenses (LGPL) [15], require that the source code be released when redistributed. In the case of the GPL license, if a work uses a GPL-licensed work, the work must be released under the GPL license when redistributed. The LGPL license states that a proprietary work can use an LGPL-licensed linked library, and only the library must be released when redistributed. The caveat is that these libraries must be linked external of the proprietary application (*e.g.*, as dynamically linked library files), which can prove to be difficult for the complex build processes for space radios.

Consequently, we used permissive libraries to ensure that, when our source code is added to the Space Telecommunications Radio System (STRS) Repository [16] for redistribution, no software licenses will be violated if portions of the code are protected based on the International Traffic in Arms Regulations (ITAR) and other export control laws. Unfortunately, this severely limited the libraries that could be used for our CE.

B. Libraries

Each year, the number of publicly available artificial neural network (ANN) and machine learning libraries continues to expand. Many of these libraries have application program interfaces (APIs) for high-level languages, such as Python, but lack APIs for lower-level languages, such as C/C++. Furthermore, some of the C/C++ applicable libraries lack documentation for their lower-level APIs, possess very complex build processes [17] (which eliminates portability to embedded devices), and/or were designed for very large, complex applications [18] (such as deep, convolutional NNs).

MLPack [19] was chosen to implement the NNs in our CE. At the time of developing the software, the ANN library was only available on the "master" branch on GitHub; it was not included in any of the stable releases. It also did not include a Levenburg–Marquardt backpropagation implementation for training; hence, this functionality was written from scratch by the authors.

Armadillo [20] was chosen for handling matrix/vector operations and storage. MLPack uses this library internally, which made interfacing much easier. Armadillo provides a simple, MATLAB-like API for linear algebra functions and also seamlessly handles multi-threading for matrix operations.

The Boost.Asio [21] library was leveraged for interfacing Ethernet and UDP/IP with the ground BPSK transmitter and ViaSat DVB-S2 modems, respectively. This library abstracts any operating system–specific constructs to handle sockets. The Boost.Serialization [22] was chosen for saving/resuming the state of the CE upon program exit/start.

C. Architecture

The block diagram of the software architecture (Fig. 2) shows that it consists of an RLNN core and external drivers. The external drivers interface to the Viasat DVB-S2 modem (for receiving E_s/N_0 measurements), the ML-605 BPSK transmitter (for sending new actions to the space DVB-S2 transmitter), and the Advanced Radio for Cognitive Communications (ARCC) for initial ground testing. The cognitive algorithm resides in the RLNN Core. The Core itself provides the "glue" code for all of its submodules as well as provides the RL framework.

The Explore NN Predictor and set of Exploit NN Predictor modules in the RLNN Core abstract the MLPack libraries to an ensemble of NNs in each NN Predictor. The CE, implemented on the workstation, provides online training, which allows the cognitive application to simultaneously predict new actions while training. This key feature, required for a useful real-time system, was implemented by duplicating the NN Predictor modules: a training NN Predictor module and an execution NN Predictor module. Whenever the training buffer fills up, the training NN Predictor starts training its ensemble of NNs. When complete, the training module copies the weights to the execution NN Predictor module.

Both the Explore and set of Exploit NN Predictors share a common training buffer which holds the latest N unique actions tested and their corresponding multi-objective performances. When the NN Predictors need to be trained, the

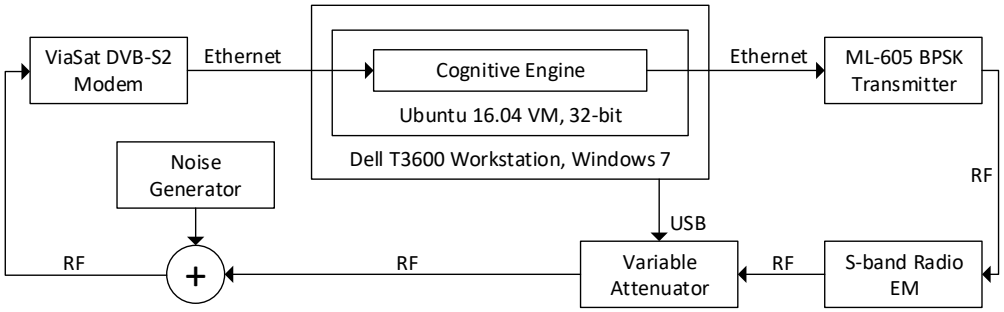


Fig. 3. RLNN implementation ground test setup. The variable attenuator is programmed to emulate the fading of a typical ISS pass over the GRC ground station.

training buffer transforms its buffer into labeled training and validation sets for each type of NN (the explore and exploit NNs do not have the same inputs and outputs). This is implemented using Armadillo vectors and matrices.

The Application Specific Module (ASM) provides the "context" for our communications application. The other aspects of the RLNN Core are generic and can be used for other applications outside of communications. The ASM provides the functions necessary to transform communications measurements into fitness scores used by the RLNN core and accomodates deviations from the original architecture [5] to meet software implementation constraints. For example, one function in the ASM limits changes to the transmit power level to less than 1.5 dB between consecutive times steps. Preliminary testing with the Viasat DVB-S2 modem showed that larger power steps caused the receiver to struggle to stay locked onto the incoming signal resulting in a degraded performance. Another function in the ASM monitors the bit error rate (BER). If the BER is measured to be 0.5 (a completely corrupted frame), then all objective fitness scores are zeroed, indicating the system failed at meeting any of its objectives. If this function was not applied, the system would be rewarded for sending unreadable frames. Depending on the chosen multi-objective weights, the system could potentially optimize to ignore the BER and instead use high-order MODCODs and low transmit powers regardless of the channel condition.

D. Implementation Results

The storage and run time memory requirements of the implemented CE software package are very lightweight. The executable size is only 3.8 MB and while running only takes about 12 MB of RAM. However, it does use all of the CPU usage allocated to it, especially during training. On the ground and flight testing workstation, all 8 cores were maxed out at 100% utilization while training occurred. Outside of training, CPU usage tended to run at 75% on each core.

The CE is a console application and outputs important data, such as the state of either exploiting or exploring, when training is occurring, the action tuple chosen, the measurement vector, the multi-objective fitness vector, and the overall fitness observed on each iteration. Similar data are written to the log file. A text log file, designed to be human readable, for a typical nine-minute pass occupies about 3 MB (uncompressed).

IV. Ground Testing

A. Test Setup

Once successfully implemented, the RLNN underwent extensive testing to characterize the system before the on-orbit experiments. A testbed at NASA Glenn Research Center (GRC) was used to create an emulation of the expected on-orbit environment (a simplified block diagram of the setup is shown in Fig. 3). The CE resides in an Ubuntu-16.04-LTS virtual machine on a Windows 7–based Dell T3600 Precision workstation. The virtual machine was allocated 4 GB of dedicated memory and shared all eight CPU cores (with hyperthreading) with the host operating system. The CE communicated to the ML-605 BPSK transmitter and the ViaSat DVB-S2 receiver over a private local area network. The ML-605 radio transmitted the new actions (chosen by the CE) to the S-band Radio engineering model. The SDR transmitted DVB-S2 frames through an emulated satellite channel from the SDR on SCaN Testbed to the GRC S-band ground station. During execution, the Windows host controlled the variable attenuator according to a measured SNR profile from a previous on-orbit experiment presented in [6]. The noise generator provided the capability to add additional noise into the system for tuning the expected E_s/N_0 at the input of the ViaSat modem. Antennas were not used in this testing, so all RF signals were carried via coaxial cables. For clarity, Fig. 3 does not show the up-converters, down-converters, and various attenuators in the system.

The parameterized design of the CE implementation provides many execution configurations. These parameter configurations include the number of parallel exploration NNs, the number of parallel sets of exploit NNs, and the size of the training buffer. Over time, these parameters can affect the overall fitness score and the time consumed by the various tasks running on the processor, such as executing the NNs and training. The goal of the ground testing was to characterize these relationships.

A series of seven groups of six tests were run according to the testing matrix shown in Table I. The first six groups of tests

TABLE I
MATRIX OF PERFORMED GROUND TESTS FOR CHARACTERIZING THE PERFORMANCE OF THE CE. THE WEIGHT VECTOR IS DEFINED AS [THROUGHPUT, BIT ERROR RATE, TARGET BANDWIDTH, SPECTRAL EFFICIENCY, TRANSMIT POWER EFFICIENCY, DC POWER CONSUMED].

Test	Mission	Weight Vector	Number of Parallel Explore NNs	Number of Sets of Parallel Exploit NNs	Training Buffer Size	Power Control
01–04	Emergency	[0.1, 0.8, 0.025, 0.025, 0.025, 0.025]	{1,2,3,4}	{1,2,3,4}	200	Yes
05–06	Emergency	[0.1, 0.8, 0.025, 0.025, 0.025, 0.025]	1	1	{100,50}	Yes
07–10	Cooperation	[0.05, 0.05, 0.4, 0.4, 0.05, 0.05]	{1,2,3,4}	{1,2,3,4}	200	Yes
11–12	Cooperation	[0.05, 0.05, 0.4, 0.4, 0.05, 0.05]	1	1	{100,50}	Yes
13–16	Balanced	[1/6, 1/6, 1/6, 1/6, 1/6, 1/6]	{1,2,3,4}	{1,2,3,4}	200	Yes
17–18	Balanced	[1/6, 1/6, 1/6, 1/6, 1/6, 1/6]	1	1	{100,50}	Yes
19–22	Power Saving	[0.05, 0.05, 0.05, 0.05, 0.3, 0.5]	{1,2,3,4}	{1,2,3,4}	200	Yes
23–24	Power Saving	[0.05, 0.05, 0.05, 0.05, 0.3, 0.5]	1	1	{100,50}	Yes
25–28	Multimedia	[0.5, 0.3, 0.05, 0.05, 0.05, 0.05]	{1,2,3,4}	{1,2,3,4}	200	Yes
29–30	Multimedia	[0.5, 0.3, 0.05, 0.05, 0.05, 0.05]	1	1	{100,50}	Yes
31–34	Launch	[0.2, 0.4, 0.1, 0.1, 0.1, 0.1]	{1,2,3,4}	{1,2,3,4}	200	Yes
35–36	Launch	[0.2, 0.4, 0.1, 0.1, 0.1, 0.1]	1	1	{100,50}	Yes
37	Emergency	[0.1, 0.8, 0.025, 0.025, 0.025, 0.025]	1	1	200	No
38	Cooperation	[0.05, 0.05, 0.4, 0.4, 0.05, 0.05]	1	1	200	No
39	Balanced	[1/6, 1/6, 1/6, 1/6, 1/6, 1/6]	1	1	200	No
40	Power Saving	[0.05, 0.05, 0.05, 0.05, 0.3, 0.5]	1	1	200	No
41	Multimedia	[0.5, 0.3, 0.05, 0.05, 0.05, 0.05]	1	1	200	No
42	Launch	[0.2, 0.4, 0.1, 0.1, 0.1, 0.1]	1	1	50	No

were configured with the six objective-function weight sets described in [5] to provide a comparison with the MATLAB simulations. These six objective-function weights correspond to suggested weights for launch/re-entry, multimedia, power-saving, balanced, cooperation, and emergency mission scenarios. In each group, four of the tests consisted of configuring the CE system for one-to-four parallel explore NNs and sets of exploit NNs while keeping the training buffer fixed at 200 samples (the buffer size used in [5]). The other two tests held the number of parallel explore NNs and sets of exploit NNs to one, while the training buffer was changed to 50 and 100 samples.

The seventh group of six tests consisted of trying each of the objective function weights for the case in which there was no functionality for changing the transmit power of the DVB-S2 transmitter. Based on the schedule of firmware upgrades to the SDR on the SCaN Testbed, one of our flight tests did not have variable transmit power functionality, so we wanted to characterize this performance.

It is worth noting that when the ground tests were conducted, the reality patch described in Section III-C had not yet been applied to the implementation. This patch was written and applied after discussions with our colleagues at NASA GRC just prior to flight testing. One of the reasons for the ground tests was to understand how well each "mission" objective will be achieved. Without the patch, the fitness scores would be overly optimistic. As a result, the following section only covers analysis related to relative trends of changing the CE parameters and not absolute performances. The absolute performances for each mission are be discussed in Section V-B with the flight results.

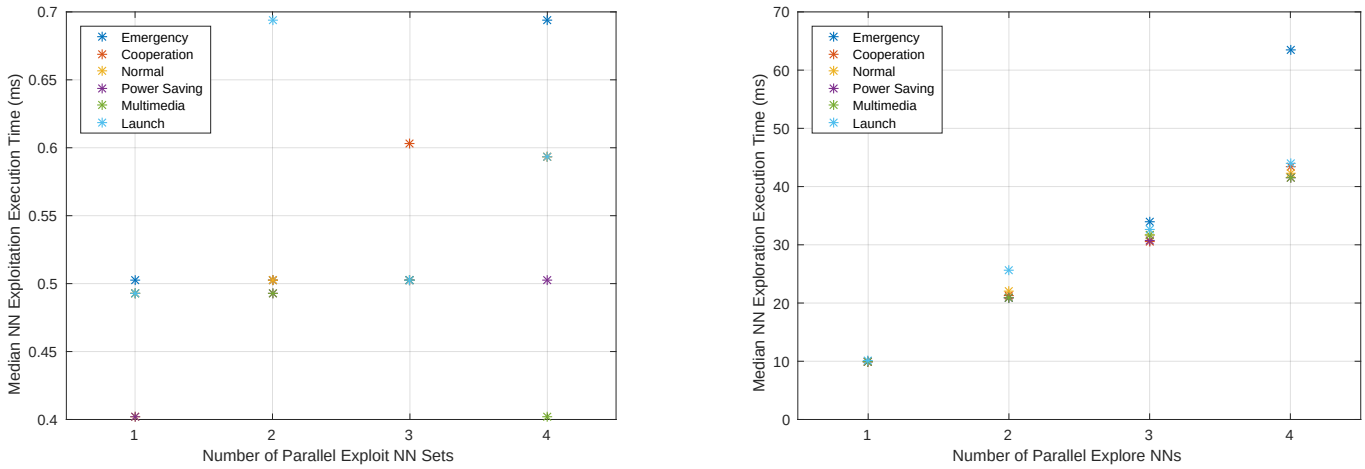
B. Results

There were 42 tests conducted and recorded to log files for offline post processing. The following sections discuss key analysis results that help characterize the CE's performance.

1) Exploitation and Exploration Execution Time: One of the most important parameters of the implemented CE is how fast it executes. One of the main reasons the simulations in MATLAB in [5] were manually ported into C++ was to dramatically increase the program execution speed to run with a reasonably fast update rate (tens of Hz) in real time. The largest contributor to execution time is the explore and exploit NN forward propagations. The NN execution time is mainly driven by how many NNs are operated in parallel. Figs. 4a and 4b explore the relationship between the execution of the exploit and explore NN predictions, respectively. Noting the time scales (on the vertical axes) on each of these two figures, the exploration NN prediction takes over an order of magnitude longer than the exploitation. This is because whenever the exploration NN runs, it must forward propagate each of the possible 1,152 actions through each explore NN. In contrast, the set of exploitation NNs only needs to forward propagate one desired performance through its array of NNs. Fig. 4a indicates the lack of a relationship between having more parallel NNs and the execution time. This is most likely because the execution time is so short that other processes running on the CPU (not related to the CE) are also being executed during the time of the exploit NN procedure. Fig. 4b shows the expected result of the execution time is at least linearly related to the number of NNs. This occurred because many more threads in parallel were spawned than there were processor cores available to execute.

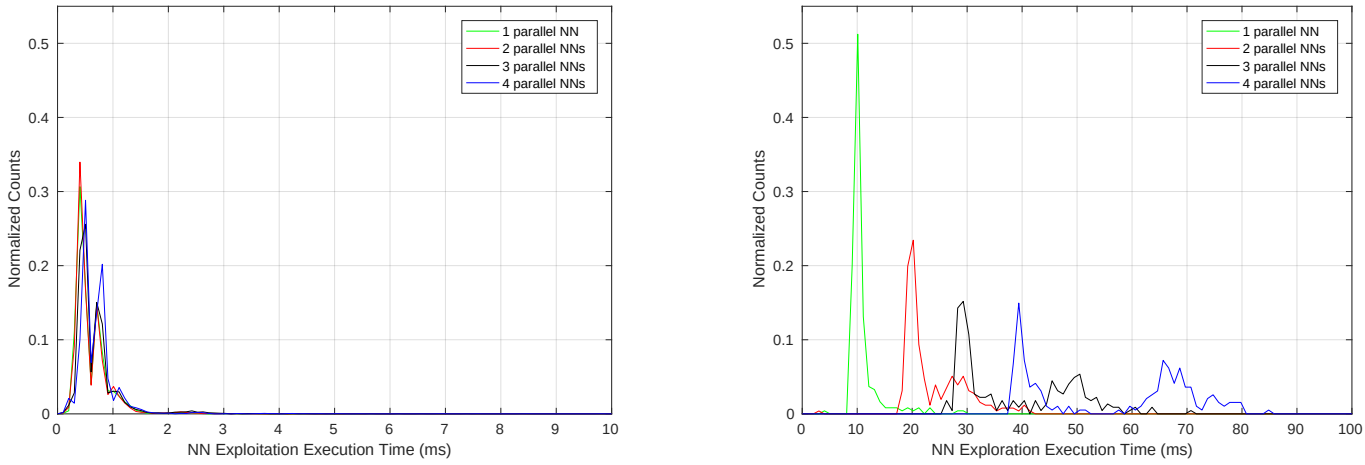
As an example of the processing difference, consider if a desired update rate during the exploit execution is 25 Hz. Any time a virtual exploration occurs, the update rate drops to 20 Hz with just one explore NN. With four explore NNs, this update rate would drop to 12 Hz—roughly half the update rate as when exploiting!

The median statistic was used because the histogram of execution times for exploitation and exploration is skewed as shown in Figs. 5a and 5b. Unlike the exploit NN distribution,



(a) (b)

Fig. 4. (a) NN exploitation execution time versus the number of parallel sets of exploit NNs and (b) NN exploration execution time versus the number of parallel explore NNs using a training buffer size of 200 samples.



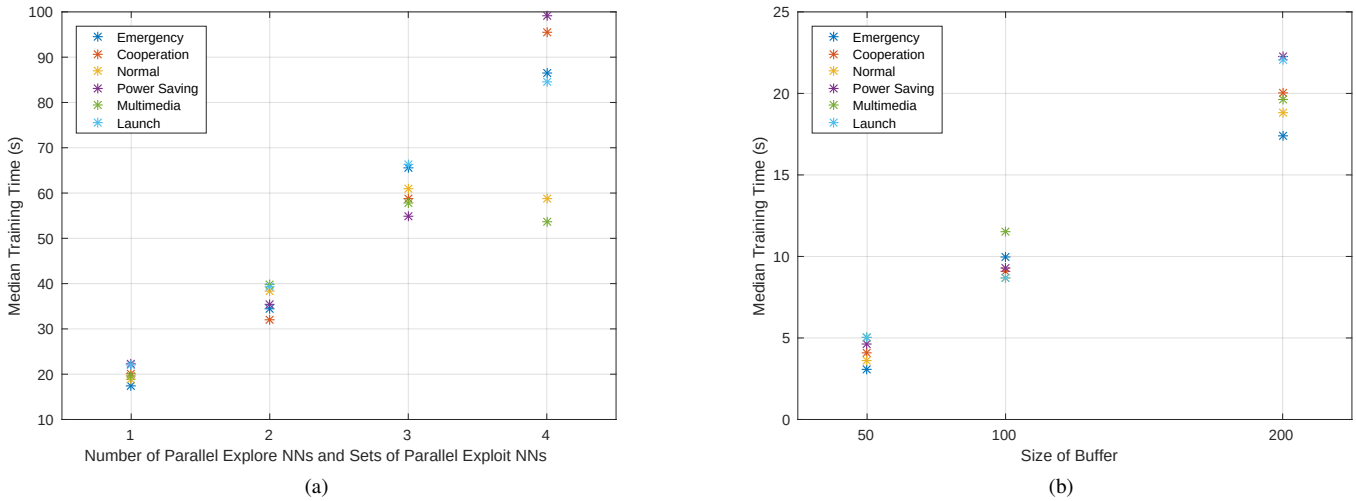
(a) (b)

Fig. 5. (a) NN exploitation execution time histogram and (b) NN exploration execution time histogram using a training buffer size of 200 samples.

the explore NN histogram appears to be bimodal. These two modes are most likely when the CE is and is not simultaneously training in another thread, which requires a large CPU load. Again, note the difference in time scales by an order of magnitude (on the horizontal axes) when comparing the two histograms.

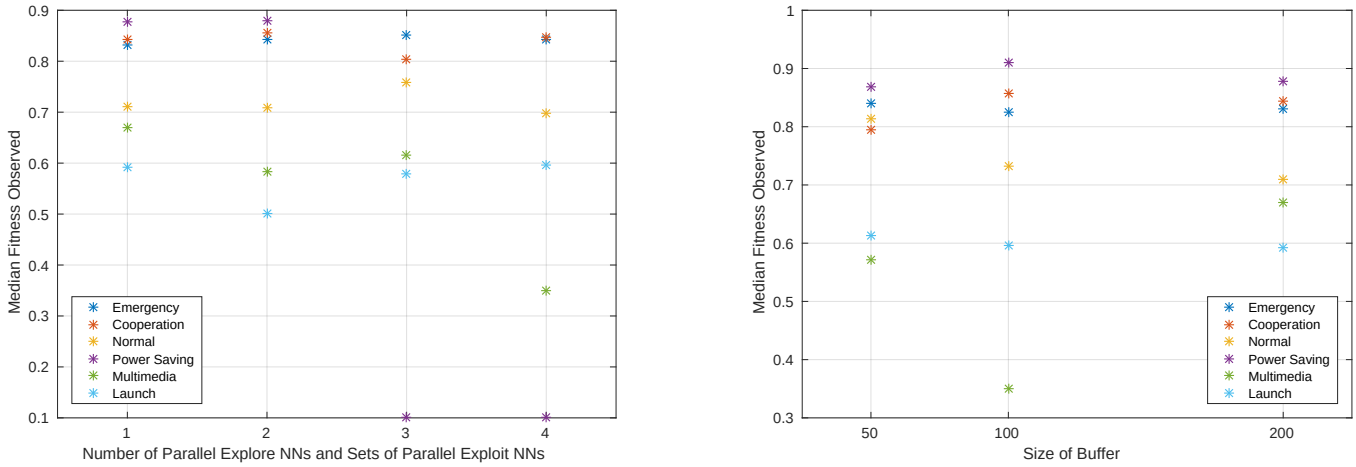
2) Training Time: Another key performance metric of the CE is the duration of time while training. As discussed earlier, although the NN training is completed online simultaneously with the NN execution (except for the initial training), the training threads compete with the execution threads on the CPU and take much of the limited computing resources. As a result, training will have an adverse effect on the exploration (and less so on the exploitation) execution time. Additionally, when the RLNN algorithm trains its NNs for the first time and there are no weights pre-trained into the execution NNs, the RLNN implementation falls back into a default state until training is complete. At this point, it can start executing with the knowledge it has learned. As shown in the flight results in later sections, this latency to leveraging learned knowledge can waste precious downlink time. Furthermore, the training duration will also affect how often the RLNN can re-train on new knowledge. If the NNs are currently training, a call for training again cannot occur until the first training process is complete.

One parameter that affects the training time is the number of parallel explore and exploit NNs. Since the amount of parallel CPU resources is limited, as the number of NNs increases, the training time increases roughly linearly for most scenarios as shown in Fig. 6a. For just four parallel explore NNs and four sets of parallel exploit NNs, the online training time reached a median value of upwards of 100 seconds on the workstation used for the implementation! Some passes of the ISS over the GRC ground station only have about nine minutes of total pass time, with the main lobe of the antenna seeing the ground station for only about four minutes. This means that, for non-pre-trained weights, over 25% of the best downlink time is lost waiting for the initial training to complete. During the rest of



(a) (b)

Fig. 6. (a) NN training time versus the number of parallel sets of exploit NNs and parallel explore NNs for a training buffer size of 200 samples and (b) NN training time versus the training buffer size using one parallel explore NN and one set of exploit NNs.



(a) (b)

Fig. 7. (a) Overall fitness observed versus the number of parallel sets of exploit NNs and parallel explore NNs for a training buffer size of 200 samples, and (b) overall fitness observed versus the training buffer size using one parallel explore NN and one set of exploit NNs.

the pass, the slow, online training means the NNs are slow to learn the changing environment.

The size of the training buffer also plays a role in the training time (Fig. 6b). A small size buffer results in very fast training times, but could result in lower performance (to be discussed in the next section).

Fig. 6 illustrates that training times vary over different types of missions/objectives. The relatively small sample size used to compute the median and the distribution of possible performance value changes (depending on the mission) each contribute to the varying training times. Certain performance distributions will train faster (*i.e.*, meet the stopping criteria faster) than others.

Another observation was the training time sensitivity to the number of unique input values. For a small number of inputs (say, fewer than three), the training time actually increased as the training error kept decreasing in small steps, which makes the training stop only due to reaching a maximum number of iterations. This was observed in our implementation when the ability to change symbol rate was removed (making the number of possible symbol rates just one) when fitting into the constant symbol rate DVB-S2 standard.

3) Fitness Observed: The most important metric of the CE is how well it achieved its multi-objective goal: the overall fitness observed. It was expected that the overall fitness observed would be affected by the number of parallel explore and sets of exploit NNs and the size of the training buffer. Theoretically, as the number of parallel NNs increases, the mean of the output ensemble of NNs provides a predicted value closer to the true output value for a given input value. Fig. 7a does not show any significant performance increase by adding more parallel NNs. This could be due to the fact that the number of NNs in an ensemble needs to be larger to make a difference—we stopped at four NNs in parallel because the training and exploration times became infeasible due to so many NNs in parallel.

Similar to the number of parallel NNs, Fig. 7b does not show a significant performance increase using a larger buffer size. This could be dependent on the E_s/N_0 profile that was used. The required relative size of the buffer is proportional to the amount of possible actions that the CE could choose. In [5], their simulations included the symbol rate as a variable parameter, which increased the number of total actions by tenfold. This shows that for an action space of roughly 1000 actions and a highly dynamic channel, a 200-sample buffer may not be necessary.

V. On-orbit Testing

Upon the successful completion of ground testing (and validation and verification), the CE was tested with an on-orbit space communications system. To the best of the authors' knowledge, this is one of the first cognitive communications experiments with space assets in the published literature.

A. Test Setup

The setup used for on-orbit testing is the same setup used by our NASA GRC collaborators for their adaptive communications testing described in [6]. Fig. 8 shows the space radio transmitter functions, CE placement among the ground receiver, and feedback control uplink path. The difference compared to [6] was that, for this experiment, we configured the RLNN cognitive engine on the adaptive communications workstation. On-board the space station, the DVB-S2 transmitter transmits data frames to the ground that are received by both the ViaSat and Newtec modems. The ViaSat modem sends E_s/N_0 measurements to the CE at 100 Hz over UDP. The Newtec modem demodulates and decodes the incoming frames and sends them to the CE workstation where they are stored as a binary file for post processing. During each data frame, the CE records the previous action tuple with its performance, chooses a new (or the same) action tuple for the next frame, and sends this decision/subsequent action to the ML-605 BPSK transmitter, which is then uplinked to SCaN Testbed. The space-based BPSK software receiver decodes the action tuple and updates the DVB-S2 transmitter on the next frame. The worst-case-round trip time of the system is approximately 40 ms at a downlink baud rate of 1 Msymbols/sec.

Flight testing was completed during a two-week window from May 2, 2017 through May 12, 2017. Based on the ISS orbit and the location of NASA GRC, on any given day, there were generally only two usable passes during typical first-shift hours (7:00 AM through 6:00 PM local time). GRC used an in-house software tool to predict the C/N_0 (carrier to noise power spectral density ratio) for all possible passes, and we selected a collection of 20 passes ranging from a qualitative "excellent" to "poor" ratings to stress test the CE with a significantly more difficult and dynamic link than used by the simulations in [5]. Based on the relative position and orientation of the solar panels and other structures on the ISS with respect to the fixed S-band antenna on-board, very severe multipath can impair the link in addition to the changing range across the the low-earth

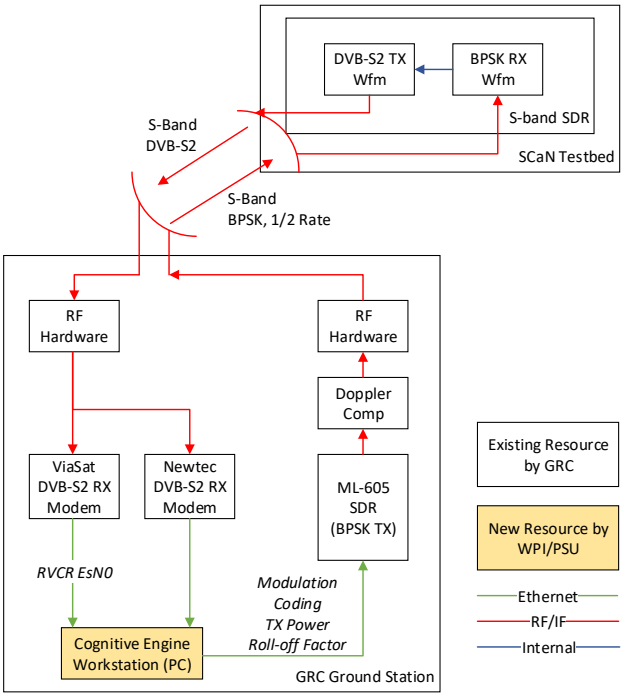


Fig. 8. Simplified block diagram of the on-orbit testing setup.

orbit. This harsh testing environment will consequently cast a pessimistic light on the results in the next section.

The testing schedule is shown in Table II. Due to the variability of the link quality and that no two passes are exactly the same, only a subset of missions were tested—emergency (BER focused), power saving (transmit-power-efficiency and DC-power-consumption focused), and cooperation (meeting a target bandwidth). These three missions show a unique aspect of the CE that is not present in existing adaptive architectures, which focus on the most throughput for a given BER. For each of the roughly six passes per mission, half of these were initialized with pre-trained weights offline from a previously recorded pass. For the other half, the weights were not initialized, so the CE had to learn initial weights during run-time. Over the span of a pass, these weights were updated multiple times with newer weights as the system records new data. The quality of the passes were distributed roughly equally as well with each mission being tested with "excellent", "great", "good", and "okay/poor" passes. Representative examples of recorded E_s/N_0 profiles for these four qualitative categories are shown in Fig. 9 from Tests 02, 15, 06, and 09, respectively, at 1 Msymbol/sec. Excellent and great provide multiple minutes of usable pass time, while good and okay/poor may provide a couple minutes or less of usable downlink time. These measurements were recorded from the ViaSat modem (subtracting out the changing transmit power levels by the CE) during the tests. When the modem lost lock because the signal was too low, the E_s/N_0 measurement may no longer be accurate; this is indicated by the red points on the plots.

For all passes, the symbol rate was fixed at 1 MSymbols/sec

TABLE II
MATRIX OF PERFORMED ON-ORBIT TESTS FOR CHARACTERIZING THE PERFORMANCE OF THE CE. THE WEIGHT VECTOR IS DEFINED AS [THROUGHPUT, BIT ERROR RATE, TARGET BANDWIDTH, SPECTRAL EFFICIENCY, TRANSMIT POWER EFFICIENCY, DC POWER CONSUMED].

Test	Date (2017)	Qualitative Link Quality	Mission	Weight Vector	Pre-Trained Weights
01	02-May	Good	Emergency (No Pwr Cntrl)	[0.15, 0.8, 0.025, 0.025, 0, 0]	No
02	02-May	Excellent	Emergency	[0.1, 0.8, 0.025, 0.025, 0.025, 0.025]	No
03	02-May	Poor	Emergency	[0.1, 0.8, 0.025, 0.025, 0.025, 0.025]	No
04	03-May	Great	Power Saving	[0.05, 0.05, 0.05, 0.05, 0.3, 0.5]	No
05	03-May	Good	Power Saving	[0.05, 0.05, 0.05, 0.05, 0.3, 0.5]	No
06	03-May	Good	Emergency	[0.1, 0.8, 0.025, 0.025, 0.025, 0.025]	No
07	04-May	Good	Power Saving	[0.05, 0.05, 0.05, 0.05, 0.3, 0.5]	No
08	04-May	Great	Emergency	[0.1, 0.8, 0.025, 0.025, 0.025, 0.025]	Yes
09	05-May	Poor	Power Saving	[0.05, 0.05, 0.05, 0.05, 0.3, 0.5]	Yes
10	05-May	Excellent	Power Saving	[0.05, 0.05, 0.05, 0.05, 0.3, 0.5]	Yes
11	08-May	Okay	Emergency	[0.1, 0.8, 0.025, 0.025, 0.025, 0.025]	Yes
12	08-May	Great	Emergency	[0.1, 0.8, 0.025, 0.025, 0.025, 0.025]	Yes
13	09-May	Great	Power Saving	[0.05, 0.05, 0.05, 0.05, 0.3, 0.5]	Yes
14	09-May	Good	Cooperation	[0.05, 0.05, 0.4, 0.4, 0.05, 0.05]	No
15	10-May	Great	Cooperation	[0.05, 0.05, 0.4, 0.4, 0.05, 0.05]	No
16	10-May	Good	Power Saving	[0.05, 0.05, 0.05, 0.05, 0.3, 0.5]	Yes
17	11-May	Good	Cooperation	[0.05, 0.05, 0.4, 0.4, 0.05, 0.05]	No
18	11-May	Great	Cooperation	[0.05, 0.05, 0.4, 0.4, 0.05, 0.05]	Yes
19	12-May	Excellent	Cooperation	[0.05, 0.05, 0.4, 0.4, 0.05, 0.05]	Yes
20	20-May	Poor	Cooperation	[0.05, 0.05, 0.4, 0.4, 0.05, 0.05]	Yes

because this baud rate would provide a large dynamic range of possible actions for the CE to try. The number of parallel explore and sets of exploit NNs were set to just one. This was because ground testing did not reveal a significant accuracy advantage for using up to four in parallel, and this would allow the CE to run at its fastest update rate. The training buffer size was set to 200 samples for all tests except Test 01. The ability to change the DVB-S2 transmitter output power on-the-fly was not available until Test 02. Test 01 acted as a system checkout with the CE. Without the transmit power option, the action space became significantly smaller, so the training buffer was set to 50 samples instead of 200 samples. Using MODCODs 01–10, 12–16, 18–22, and 24–27 [7], filter roll-off factors of 0.35, 0.25, and 0.20, and transmit powers of –7.5 dB to 0.0 dB (in 0.5-dB increments) relative to the transmit power used in [6] resulted in an action space of 1,152 possible actions. Finally, one of the control features of the CE is a threshold when the system will reset and refill the buffer and retrain. The threshold is the difference between the current overall fitness observed and the previous fitness value. Tests 01 and 04 used a value of 0.8 and 0.2 (a relatively conservative trigger for their missions), respectively, as this was what was used in simulation. The dynamic nature of the link (that was not observed in ground testing) triggered this threshold often, so the threshold was increased to 0.95 for all other tests, which makes it very unlikely for the system to automatically reset.

To prevent training the CE on corrupted frames during tests not using pre-trained weights, on each pass the CE was manually started once the ViaSat modem reported a lock on the incoming frames. The engine was manually stopped roughly at the end of the scheduled pass whenever the modems were not recovering from loss of signal because the received signal power was too low.

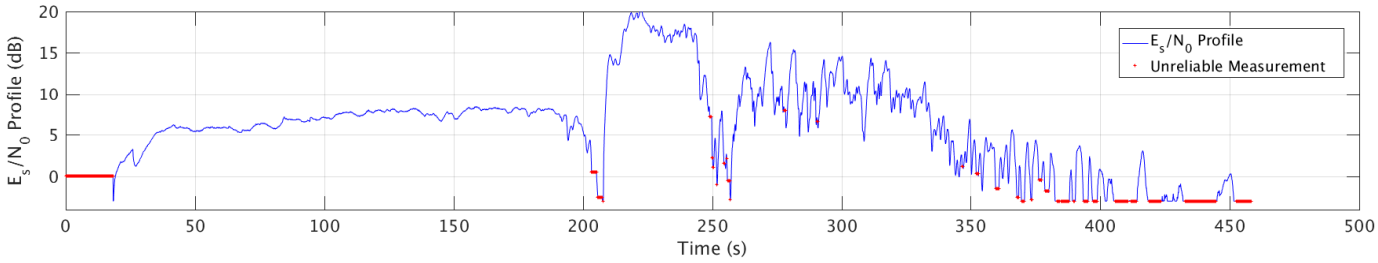
B. Results

Due to the volume of measurements during the on-orbit testing, this section provides selective insights from analyses not covered in the ground testing in Section IV-B.

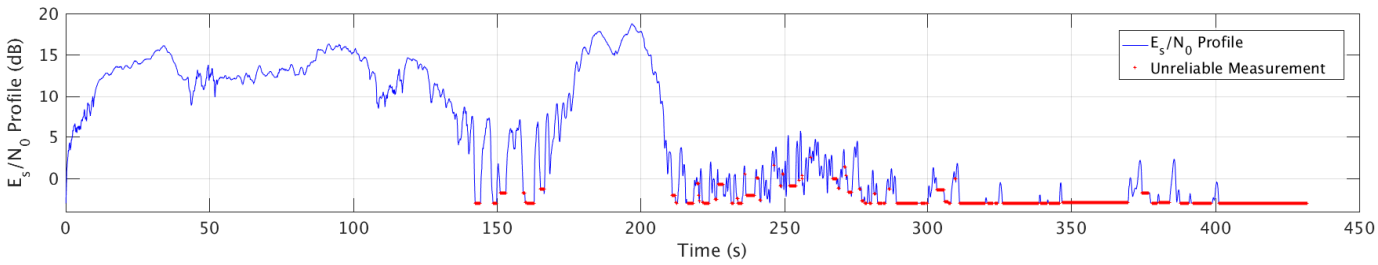
1) Sample Time Series Analysis: To understand how the CE behaves over the period of a pass, the recorded measurements from Test 08, which was optimized for the emergency weights with pre-training during a great pass, were analyzed. Looking at Fig. 10a, the usable downlink duration before severe multipath fading was approximately 200 s with a peak E_s/N_0 of about 13 dB.

Internally, the CE uses measured Newtec modem FER performance curves based on the ViaSat E_s/N_0 measurements when estimating the FER in real-time for computing its observed fitness. It is possible that the CE may record an optimistic fitness score if the FER estimation indicates that a packet was received when, in actuality, the received packet was corrupted. To account for this, during post processing the binary file saved from the Newtec modem is used to determine when the Newtec modem actually received valid packets. Valid packets occur when the modem records a packet to the logger file and the packet is considered corrupted when the packet is not recorded. For corrupted packets, the observed fitness is set to zero. This provides a fair method of analyzing the true performance of the CE despite only *estimating* the FER during execution.

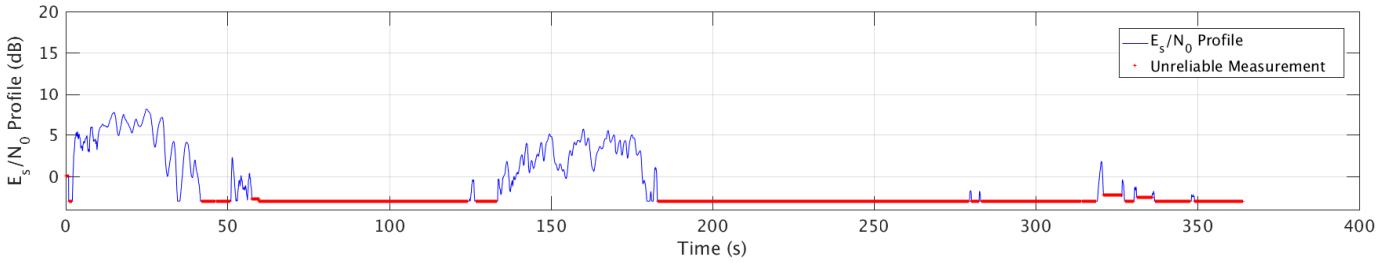
Fig. 10b shows this post-processed recorded fitness observed over time. This fitness score is achieved by choosing the action tuple of MODCOD, filter roll-off factor, and transmit power shown in Figs. 10c, 10d, and 10e, respectively. During the first section of the pass before the first sharp fade ($t = 150$ s), the MODCOD increased as the channel quality increased. Since the emergency mission is dominated by maintaining a low FER, increasing MODCOD changes were very conservative to ensure that the FER remained



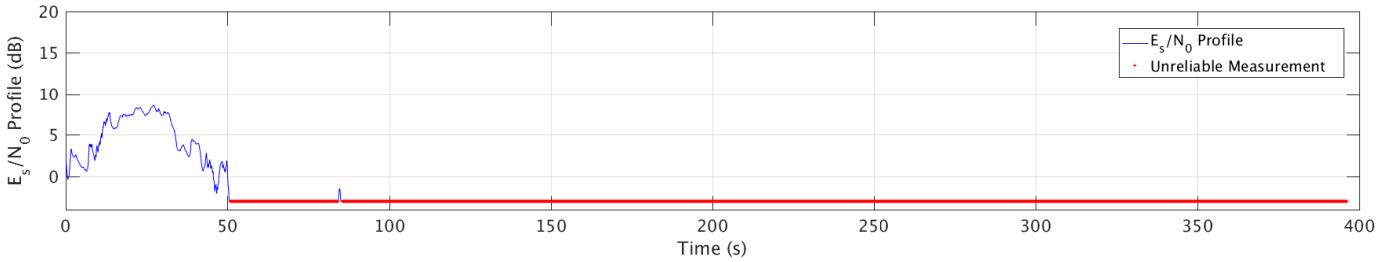
(a) Sample of measured excellent E_s/N_0 profile. Severe multipath fading does not occur until the end of the pass.



(b) Sample of measured great E_s/N_0 profile. Severe multipath fading occurs about halfway through the pass.



(c) Sample of measured good E_s/N_0 profile. Only two small time segments can support a reliable link.

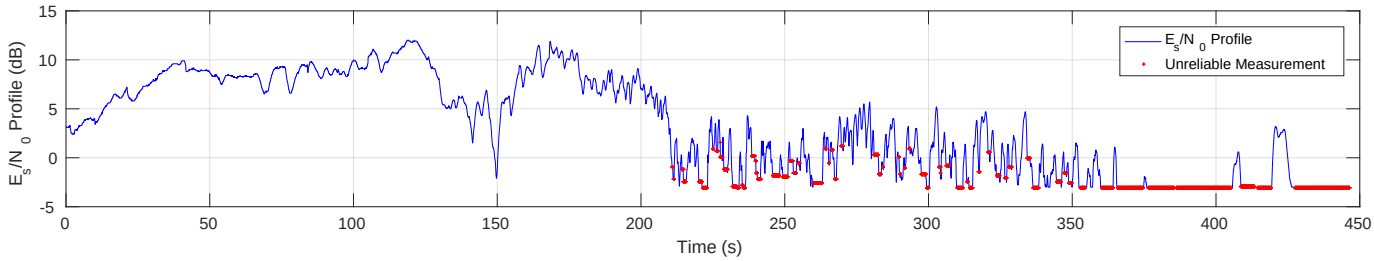


(d) Sample of measured okay/poor E_s/N_0 profile. Only the beginning of the pass can support a reliable link margin.

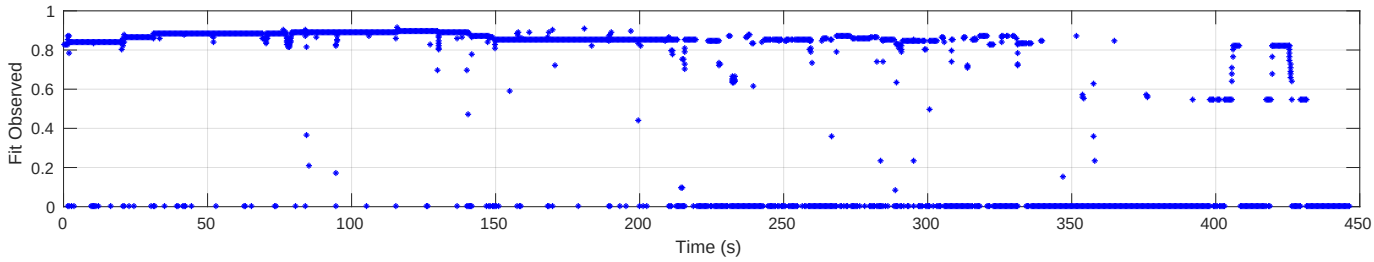
Fig. 9. (a) Excellent, (b) great, (c) good, and (d) okay/poor E_s/N_0 profiles measured during on-orbit experiments. The unreliable measurements indicate when the ViaSat modem lost receive lock and thus was not providing accurate E_s/N_0 measurements.

approximately zero. The upward and downward spikes indicate when the CE was exploring rather than exploiting. After the first deep fade, the CE struggled to recover and use higher MODCODs when the link quality increased again. Once the pass moved into the severe multipath section ($t > 200$ s), the CE started to run through its history buffer to find a suitable MODCOD to use, but was unsuccessful (hence the rapid fluctuations). At $340 < t < 380$ s, a control mechanism was triggered to re-explore the action space and then retrain its weights because its current weights were not providing satisfactory performance. When the engine was training, it defaulted back to MODCOD 1 using full transmit power ($380 < t < 440$ s). Since it was trained on very poor channel conditions, the engine still could not find a suitable action to use once training finished.

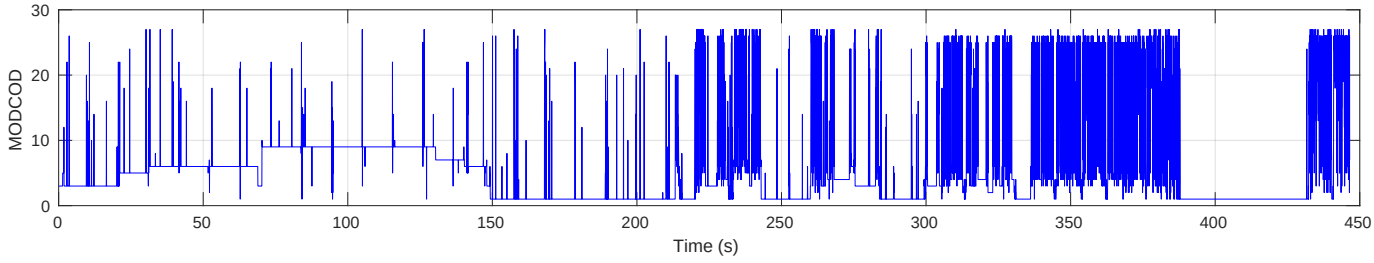
The emergency mission provides little weight value to meeting a target bandwidth or conserving power. The target bandwidth was set to the maximum bandwidth, which was why the roll-off factor moved to 0.35. Ideally, the transmit power would remain close to its maximum in order to gain more throughput (weighted higher) at the cost of losing power efficiency (weighted less). Looking at this specific time series, the engine did not learn this intuition very well. One reason for this may be because of the transmit power patch applied. This patch allowed for only 1.5-dB instantaneous jumps in transmit power between each action tuple—even when exploring. This makes it less likely for the CE to explore actions with very high or very low power levels. This can be seen when the engine started randomly exploring at ($340 < t < 380$ s). The maximum and minimum transmit powers were very rarely



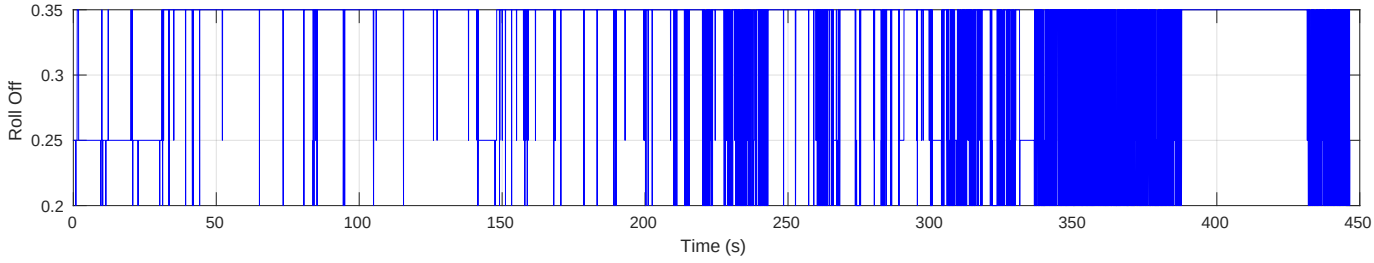
(a)



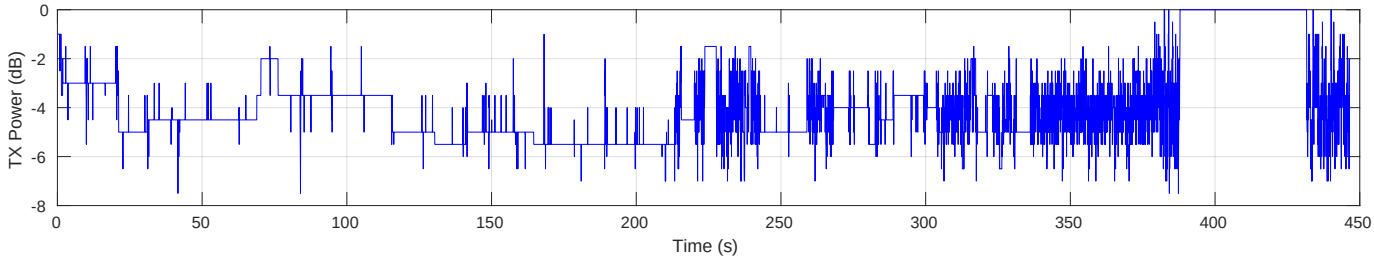
(b)



(c)

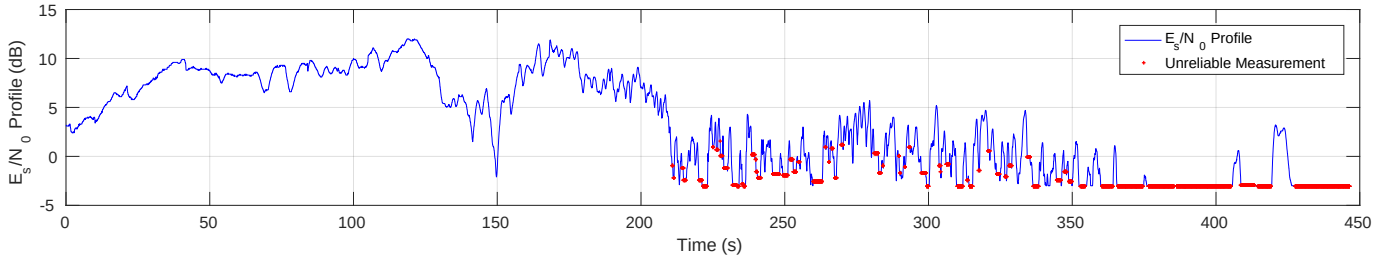


(d)

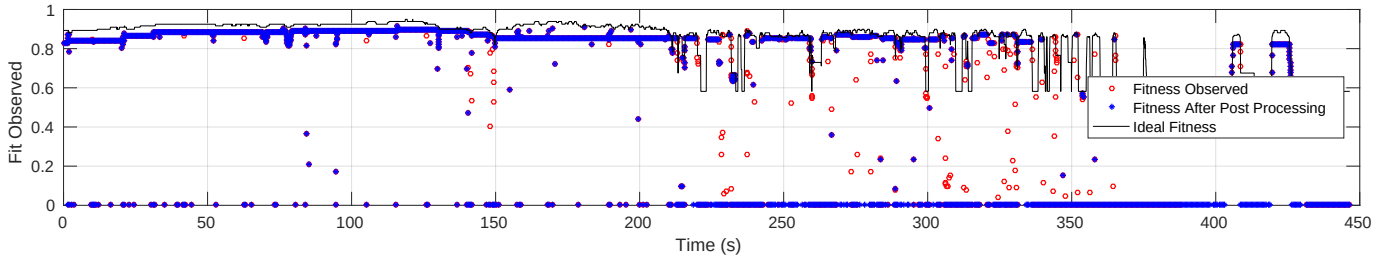


(e)

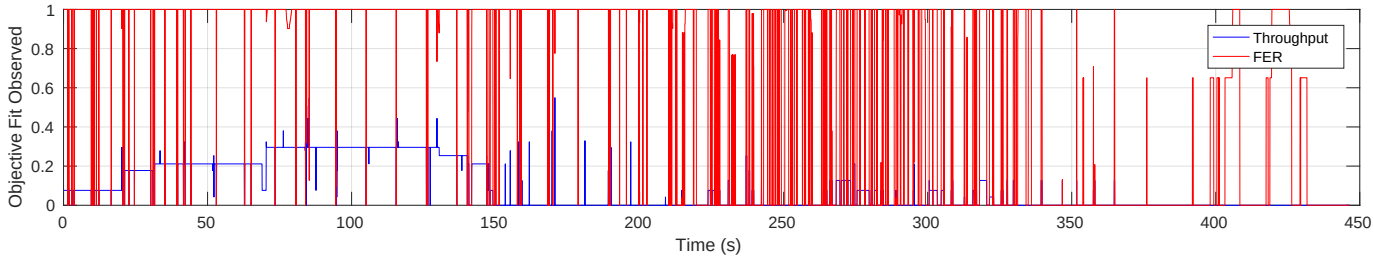
Fig. 10. Sample (a) E_s/N_0 profile, (b) fitness observed, (c) MODCOD, (d) filter roll-off, and (e) transmit power time series from Flight Test 08 (emergency mission).



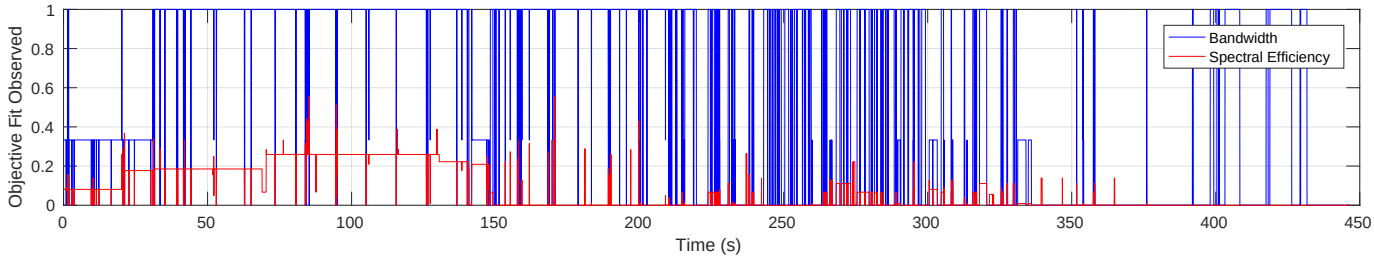
(a)



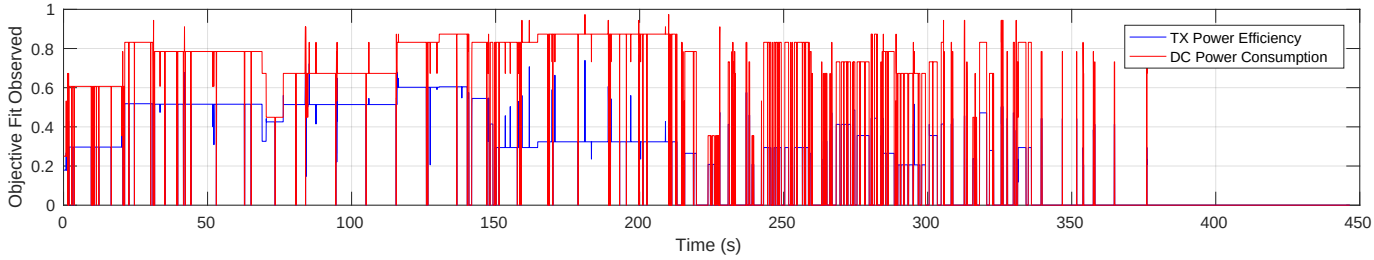
(b)



(c)

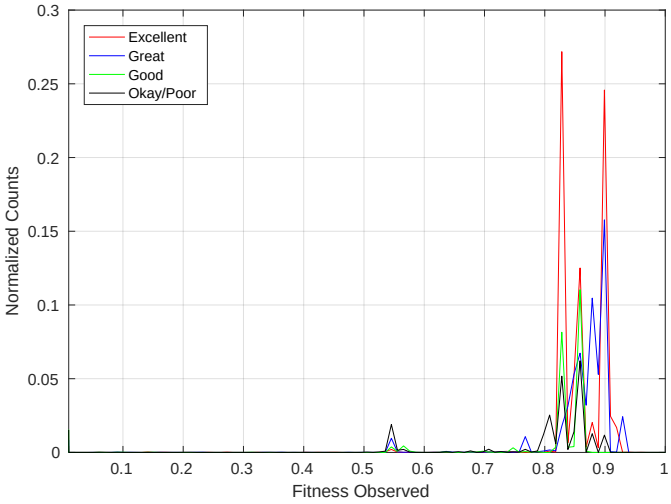


(d)

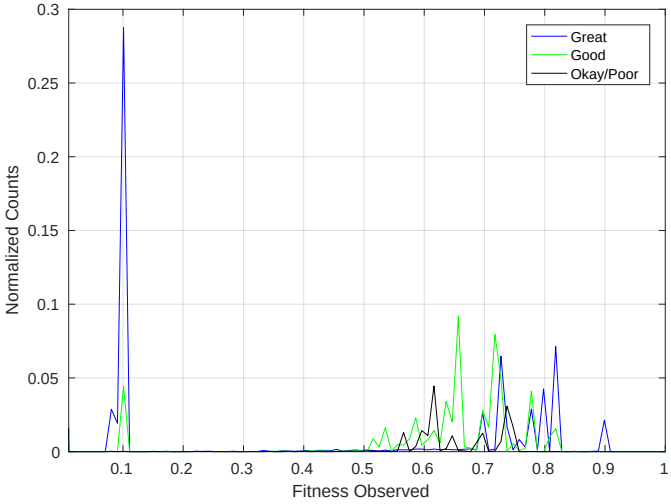


(e)

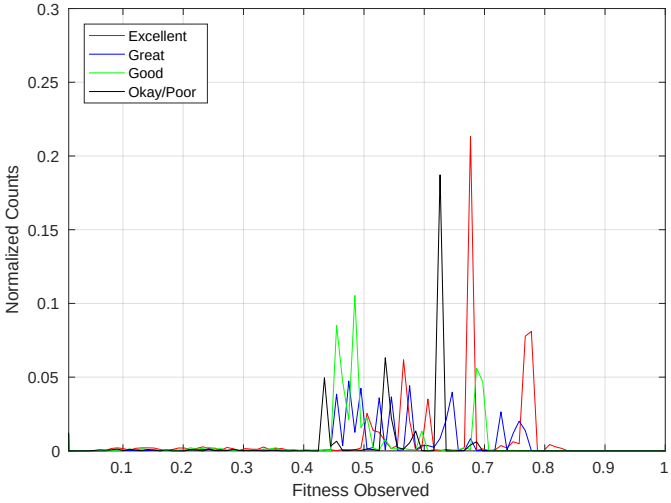
Fig. 11. Sample (a) E_s/N_0 profile, (b) fitness observed with comparison to the ideal fitness, (c) throughput and frame-error-rate objective fitnesses observed, (d) bandwidth and spectral efficiency objective fitnesses observed, and (e) transmit power and power consumption objective fitnesses observed time series from Flight Test 08 (emergency mission).



(a) Emergency mission fitness observed distributions.

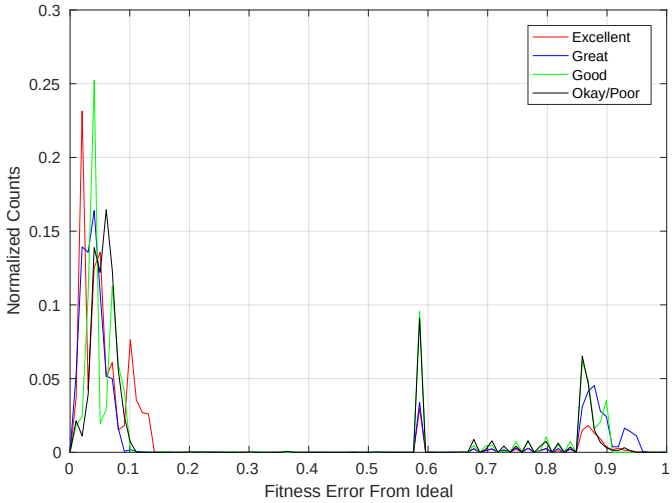


(b) Power saving mission fitness observed distribution.

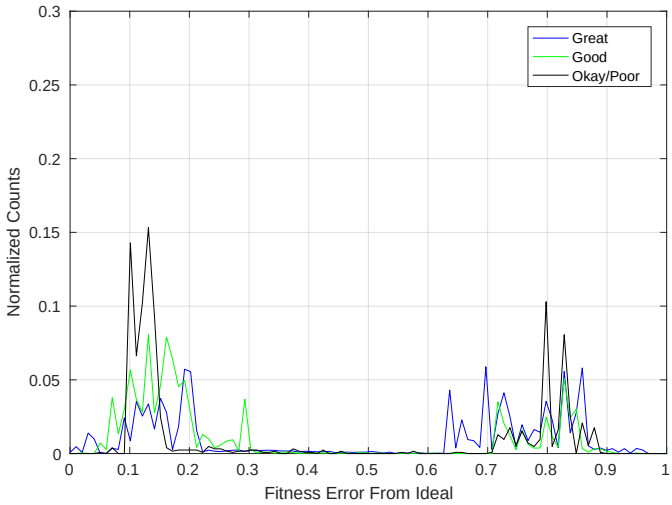


(c) Cooperation mission fitness observed distribution.

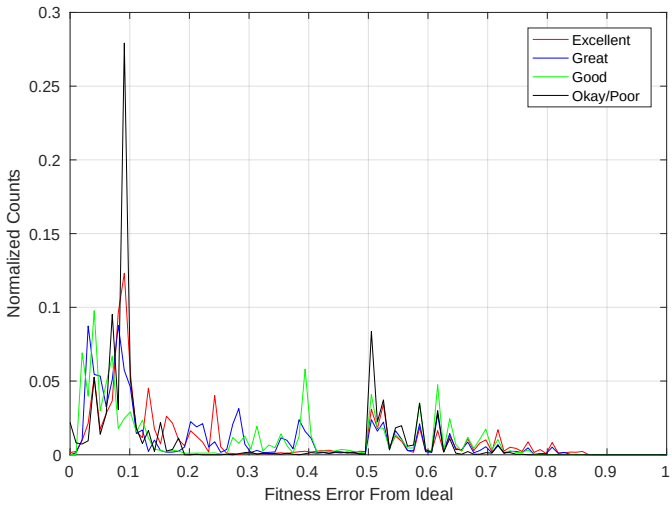
Fig. 12. Fitness observed distributions of (a) emergency, (b) power saving, and (c) cooperation mission tests for each of the qualitative pass ratings. Zero fitness scores were not plotted to show detail in the histogram. Tests 10 and 13 are omitted because the Newtec modem logging was inadvertently not started until the middle of the pass, which invalidates the recorded data for that pass. Test 01 is omitted because it did not have the ability to change transmit power.



(a) Emergency mission fitness error distributions.



(b) Power saving mission fitness error distributions.



(c) Cooperation mission fitness error distributions.

Fig. 13. Difference between the ideal fitness and the fitness observed distributions of (a) emergency, (b) power saving, and (c) cooperation mission tests for each of the qualitative pass ratings. Tests 10 and 13 are omitted because the Newtec modem logging was inadvertently not started until the middle of the pass, which invalidates the recorded data for that pass. Test 01 is omitted because it did not have the ability to change transmit power.

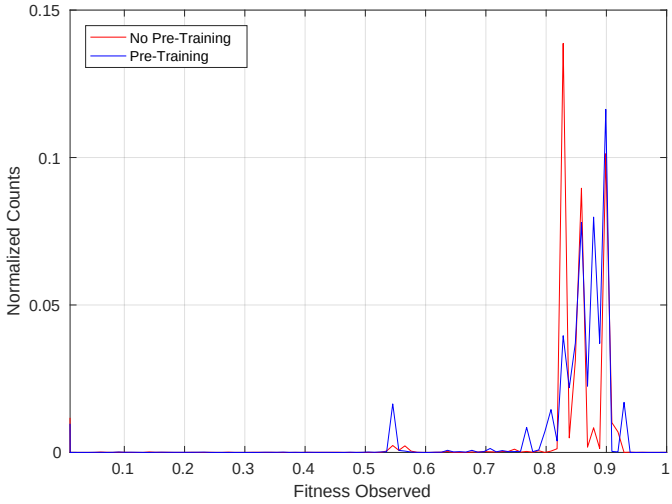
explored. The CE cannot exploit actions with high power levels until it has explored them, which it rarely will do due to the patch/restriction. This type of degradation in performance based on real-world constraints is a distinguishing feature from the simulation results in [5].

Whereas Fig. 10 sheds light on the action tuples chosen by the CE, Fig. 11 provides details on how the fitness observed is computed. The E_s/N_0 profile in Fig. 10a is repeated in Fig. 11a for the convenience of providing context to the fitness curves. Fig. 11b shows the effect of post processing the fitness observed discussed above. The blue points indicate the post-processed fitness values; the red circles show the recorded fitness by the CE (before post-processing). Empty red circles indicate those points where the fitness was zeroed out in post processing (*i.e.*, the CE was optimistic in its estimation of the FER). The black curve above the blue points shows the maximum fitness that could have been achieved during each time step. This was calculated by re-running the E_s/N_0 profile in post-processing and finding the action tuple that produced the maximum fitness score for each time step using exhaustive search. As discussed above, it is now obvious that a fitness score of one is not necessarily possible. Fig. 11b shows there is room for improvement for the CE, but that it performed relatively well during this pass.

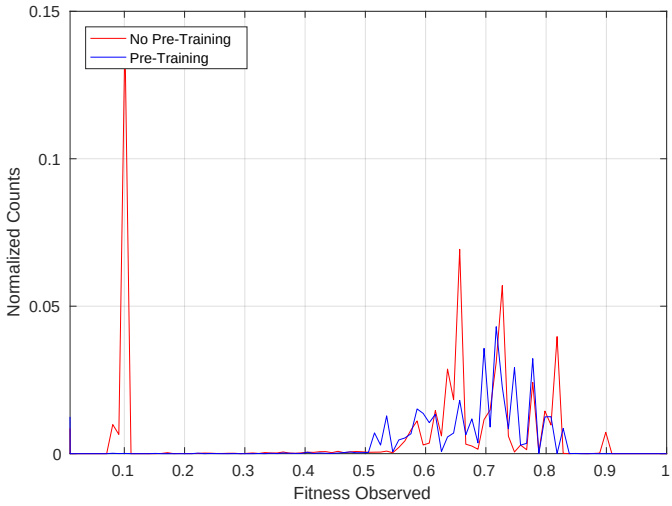
Figs. 11c, 11d, and 11e show the fitness scores for each individual objective (before getting weighted and summed into the curve in Fig. 11b). As minimizing FER was the predominant objective, the exploiting phase attempted to maintain a score of one. The throughput and spectral efficiency moved as the MODCOD changed. The bandwidth changed as the filter roll-off factor changed. The transmit power efficiency changed as a function of the MODCOD and transmit power. Finally, the DC power consumption varied as the TX power changed. Due to the low weighting, the DC power, TX power, spectral efficiency, and target bandwidth are not of much interest in the emergency case.

This type of analysis was conducted for each of the 20 flight tests to gain insight into the performance of the CE on a pass-by-pass basis. Such an analysis can present details that get lost when aggregating test results together and removing the element of time.

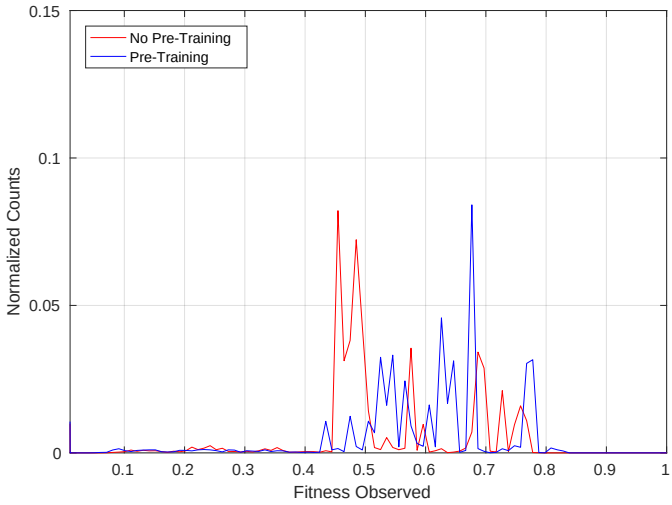
2) Aggregated Results: Histograms were generated to analyze the aggregate results from the 20 flight tests. These histograms exclude the counts for a zero fitness score to more easily compare non-zero fitness scores. Fig. 12 shows the fitness score observed for the three missions tested. In each plot, a separate distribution is provided for each type of pass quality. In Fig. 12a (emergency), the excellent and great passes had distributions with higher peaks in the higher fitness scores. As the pass quality decreased the height of the peaks decreased, the variance of the distribution increased, and the distribution shifted slightly to the left. Lower peaks indicate that there were more zero fitness scores and the CE used a larger amount of action tuples to maximize performance. A wider variance indicates the use of many unique action tuples but also indicates a more dynamic channel. A slight shift in the distribution to the left indicates the mean performance of the cognitive system decreased. This shift could have occurred



(a) Emergency mission fitness observed distributions.



(b) Power saving mission fitness observed distributions.



(c) Cooperation mission fitness observed distributions.

Fig. 14. Fitness observed distributions of (a) emergency, (b) power saving, and (c) cooperation mission tests for tests with and without NN weights pre-training. Zero fitness scores were not plotted to show detail in the histogram. Tests 10 and 13 are omitted because the Newtec modem logging was inadvertently not started until the middle of the pass, which invalidates the recorded data for that pass. Test 01 is omitted because it did not have the ability to change transmit power.

because the channel quality decreased and/or the CE struggled to maintain performance in worse channel conditions. In Fig. 12b (power saving), the variance and height of the peaks of the better performing scores remained the same regardless of the pass quality, but the distribution shifted left as the quality worsened. This indicates that the CE's performance gracefully decreased with decreasing channel quality. The relatively constant peak height and variance show that it operated in the same manner regardless of channel quality in terms of the amount of action tuples it was using. With the passes all being approximately the same length, the large spike for the great channel quality at 0.1 indicates that the CE dropped to zero fitness much less frequently. In Fig. 12c (cooperation), the height of the peaks varied and the variance generally remained the same. Once again, this indicates that the CE performed in a consistent manner between channel conditions. The distribution from excellent to all of the other quality levels shifted left, but the distribution shift is not apparent between the other quality levels. This is an interesting result because it can indicate that the CE performed better (relative to its channel quality) at worse channel conditions.

In addition to examining the absolute fitness scores, the distribution of the difference between the ideal fitness score and the measured fitness score was analyzed, which is shown in Fig. 13. In Fig. 13a (emergency), there is a slight distinction between each of the quality levels with a larger mean error from the ideal as the quality of the pass decreased. The variance of this error remained relatively constant between the four quality levels. These shift and variance characteristics indicate that the CE was not able to maintain the relative performance compared to the ideal as the channel conditions worsened. In Figs. 13b (power saving) and 13c (cooperation), the variance of relatively low and high errors increased with increasing quality, whereas the mean difference remained constant. A larger variance can indicate that when a higher quality pass allowed for more possible good action tuples, the CE struggled on narrowing in on the best action tuple to maximize its fitness score.

Finally, it is interesting to note how the fitness distribution changed when pre-training was used versus no pre-training. In Fig. 14a (emergency), the mean performance in the main cluster is about the same regardless of pre-training. This shows that a non-pre-trained CE eventually converged to the same performance as a pre-trained engine over the course of a pass. The time spent in higher performance values is larger overall with pre-training, which is what is expected—pre-training helps provide good performance from the start of the pass. In Fig. 14b (cooperation), the fitness distribution with pre-training appears to have a flatter distribution than without pre-training. This could indicate the flexibility of using more unique action tuples based on its pre-trained knowledge. In Fig. 14c (cooperation), there is a clear shift to the right (higher performance) in the mean fitness observed when using pre-training and the variance appears to remain relatively the same between the two distributions. This shows once again that pre-training helped significantly at the beginning of the pass to provide good performance. Overall, between the three missions, the pre-training helps either increase the mean performance or at least gives the CE more action tuples to choose from when exploiting. These results also show that the algorithm can operate just fine without pre-training if the situation does not allow for pre-training.

VI. Conclusion

To the best of the authors' knowledge, the testing of this CE with a real space-based asset marks one of the first published experiments of a space communications CE. The goal of the implementation and testing of this engine was to provide a technology proof-of-concept demonstration that reinforcement learning–based multi-objective optimization is feasible and useful for satellite communications. Unlike many deep NN systems that require huge training sets and training time, this system does not need to be pre-trained (though pre-training does help) and uses a relatively modest ground workstation. The system responds slower than our colleagues' ACM system [6] to fading; however, it is also important to note the RLNN approach provides greater flexibility than just MODCOD adaptations. The RLNN could achieve improved response time with a larger training buffer and longer training time. It is evident that the cognitive system does start to learn good actions in a rudimentary sense, but the NN weights are by no means optimal by the end of a single pass. For example, during the emergency mission, the CE should have learned to use the lowest MODCOD and the highest transmit power to maintain a link at the end of a pass.

There are many areas in which the CE could be improved. The initial random explore and initial training time can last on the order of a minute. For some of the worse pass qualities (good and okay/poor), this could be the entire usable downlink time of the pass. Even if pre-training does not achieve a better mean performance, it at least provides better than pure random performance at the beginning of a pass. Currently, the CE does not handle very poor link conditions well. The algorithm begins to cycle through its history buffer rapidly trying to find an action tuple that will work, but generally does not find a suitable action set. This shows that the system lacks training in very low SNR conditions. The pre-training itself was very rudimentary—training the CE on the start of a previously recorded pass. By replaying thousands of passes through the CE for pre-training, the deployment performance will likely improve.

The training process itself could be significantly improved. The Levenberg–Marquardt backpropagation training optimizer trains on the entire dataset. The knowledge contained in the current NN weights are thrown out whenever the system retrains. This causes the CE to only retain knowledge of the recent past. Future work will entail changing the optimizer to a method that allows sample-by-sample or batch training so that every training session adds knowledge to the CE, rather than replacing it.

In summary, a CE previously defined and simulated in [5], [9], [10] was developed and integrated into a fielded space flight system. Implemented in C++ for speed and portability, the system leverages existing permissive-licensed software libraries that accelerated development and promotes reuse. It

was verified and tested on the ground using radio engineering models and a channel simulators prior to space link testing. The space flight architecture leveraged a previously used system by our colleagues at NASA GRC [6]. The goal of the flight testing was to provide a baseline performance and potential of CEs for communications systems and to inspire future research. The software developed for this implementation is planned to be available through the NASA STRS Application Repository [16]. The flight testing proved the potential for reinforcement learning–based systems for satellite communications, but there is much to be improved upon in the future.

ACKNOWLEDGMENT

This work was supported by a NASA Space Technology Research Fellowship (grant number NNX15AQ41H) and a cooperative agreement with NASA John H. Glenn Research Center (grant number NNC14AA01A). The authors would also like to thank Joseph Downey, Michael Evans, and the rest of the Cognitive Communications Project team at NASA GRC for all of their help and support.

REFERENCES

[1] M. C. Scardelletti, R. C. Reinhart, M. Andro, D. J. Mortensen, T. Kacpura, C. Smith, J. Liebetreu, and A. Farrington, "Software defined radio architecture for nasas space communications," *High Frequency Electronics*, vol. 6, no. 7, pp. 18–25, 2017.

[2] C. Morlet, "Software radio in space segment: applications, technologies, reconfiguration management and architectures," in *2006 International Workshop on Satellite and Space Communications*, Sept 2006, pp. 269–270.

[3] R. C. Reinhart and B. K. Smith, "Using international space station for cognitive system research and technology with space-based reconfigurable software defined radios," in *International Astronautical Congress*, October 2015.

[4] "Nasa technology roadmaps ta 5: Communications, navigation, and orbital debris tracking and characterization systems," NASA, Tech. Rep., July 2015.

[5] P. V. R. Ferreira, R. Paffenroth, A. M. Wyglinski, T. M. Hackett, S. G. Bilén, R. C. Reinhart, and D. J. Mortensen, "Multi-objective reinforcement learning for cognitive satellite communications using deep neural networks ensembles," *IEEE Journal on Selected Areas in Communications*, 2017, under review.

[6] J. A. Downey, D. J. Mortensen, M. A. Evans, J. C. Briones, and N. Tollis, "Adaptive coding and modulation experiment with NASA's Space Communication and Navigation Testbed," in *34th AIAA International Communications Satellite Systems Conference*, October 2016.

[7] ETSI, *Digital Video Broadcasting-Satellite-Second Generation (DVB-S2) Standard*, ETSI EN 302 307, Std., Rev. 1.2.1, August 2009.

[8] T. M. Hackett, S. G. Bilén, P. V. R. Ferreira, A. M. Wyglinski, and R. C. Reinhart, "Implementation of space communications cognitive engine," in *1st IEEE Cognitive Communications for Aerospace Applications Workshop*, June 2017.

[9] P. V. R. Ferreira, R. Paffenroth, A. M. Wyglinski, T. M. Hackett, S. G. Bilén, R. C. Reinhart, and D. J. Mortensen, "Multi-objective reinforcement learning-based deep neural networks for cognitive space communications," in *1st IEEE Cognitive Communications for Aerospace Applications Workshop*, June 2017.

[10] ——, "Multi-objective reinforcement learning for cognitive radio–based satellite communications," in *34th AIAA International Communications Satellite Systems Conference*, October 2016.

[11] H. Yu and B. M. Wilamowski, "Levenberg-Marquardt training," in *The Industrial Electronics Handbook*, 2nd ed., 2012.

[12] ISO/IEC, *C++11 Standard*, ISO/IEC 14882:2011, Std., September 2011.

[13] U. of California Berkeley, *Berkeley Software Distribution*, Std., 1999.

[14] F. S. Foundation, *GNU General Purpose License*, Std., 2007.

[15] ——, *GNU Lesser General Purpose License*, Std., 2007.

[16] J. C. Briones. (2016) STRS Application Repository. NASA Glenn Research Center. Available at https://strs.grc.nasa.gov/repository/.

[17] Y. Jia and E. Shelhamer. (2017) Caffe. Berkeley Vision and Learning Center. Available at http://caffe.berkeleyvision.org/.

[18] J. Redmon. (2013–2017) Darknet: Open source neural networks in c. Available at http://pjreddie.com/darknet/.

[19] R. R. Curtin, J. R. Cline, N. P. Slagle, W. B. March, P. Ram, N. A. Mehta, and A. G. Gray, "MLPACK: A scalable C++ machine learning library," *Journal of Machine Learning Research*, vol. 14, pp. 801–805, 2013.

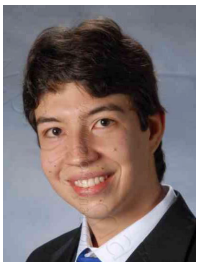
[20] R. Curtin and C. Sanderson. (2017) Armadillo: C++ linear algebra library. Available at http://arma.sourceforge.net.

[21] C. Kohlhoff. (2016) Boost.asio. Available at http://www.boost.org/doc/libs/1_63_0/doc/html/boost_asio.html.

[22] R. Ramey. (2004) Boost.serialization. Available at http://www.boost.org/doc/libs/1_63_0/libs/serialization/doc/index.html.

Timothy M. Hackett is a Ph.D. candidate in the Systems Design Laboratory (SDL), within the School of Electrical Engineering and Computer Science at The Pennsylvania State University in University Park, PA, USA. Timothy received his B.S. (student marshal) and M.S. degrees in 2015 and 2017 from Penn State in Electrical Engineering with an emphasis on communications and signal processing. He was awarded a NASA Space Technology Research Fellowship to fund his M.S. and Ph.D. work. He has worked as an intern at General Electric and The Boeing Company and as a graduate fellow at NASA Glenn Research Center and NASA's Jet Propulsion Laboratory. His research interests include satellite communications and machine learning.

Sven G. Bilén [S90, M98, SM08] received his B.S. degree from Penn State in 1991, and M.S.E. and Ph.D. degrees from the The University of Michigan in 1993 and 1998, respectively. He is a professor of engineering design, electrical engineering, and aerospace engineering at Penn State, and head of the School of Engineering Design, Technology, and Professional Programs. His research interests include software-defined radio techniques and systems, cognitive radio, and wireless sensor systems.



Paulo Victor R. Ferreira received his Ph.D. degree in 2017 from Worcester Polytechnic Institute, and B.S. and M.S. in 2010 and 2012 from Federal University of Uberlandia, Brazil, in Electrical Engineering with emphasis on Telecommunications and Electronics. Paulo Victor was awarded a Ph.D. scholarship from the Brazilian government program Science without Borders. He has worked as a Graduate Assistant at the Wireless Innovation Laboratory, within the Department of Electrical and Computer Engineering at Worcester Polytechnic Institute, Worcester, MA, USA, and as an R&D intern at General Electric Global Research Center Headquarters. His interests include satellite communications, machine learning, software-defined radios, cognitive radios, autonomous systems, and space weather.

Alexander M. Wyglinski is an Associate Professor of Electrical and Computer Engineering at Worcester Polytechnic Institute, Worcester, MA, USA and Director of the Wireless Innovation Laboratory. Dr. Wyglinski received his B.Eng. and Ph.D. degrees in 1999 and 2005 from McGill University, and his M.Sc.(Eng.) degree from Queens University in Kingston in 2000, all in Electrical Engineering. During his academic career, Dr. Wyglinski has published over 40 journal papers, over 80 conference papers, 9 book chapters, and two textbooks. Dr. Wyglinskis current research activities include wireless communications, cognitive radio, software-defined radio, dynamic spectrum access, spectrum measurement and characterization, electromagnetic security, wireless system optimization and adaptation, and cyber-physical systems. He is currently being or has been sponsored by organizations such as the Defense Advanced Research Projects Agency (DARPA), the Naval Research Laboratory (NRL), the Office of Naval Research (ONR), the Air Force Research Laboratory (AFRL) - Space Vehicles Directorate, The MathWorks, Toyota InfoTechnology Center U.S.A., Raytheon, the MITRE Corporation, National Aeronautics and Space Administration (NASA) and the National Science Foundation (NSF). Dr. Wyglinski is a Senior Member of the IEEE, as well as a member of Sigma Xi, Eta Kappa Nu, and the ASEE.

Richard C. Reinhart is a senior communications engineer with NASA Glenn Research Center, located in Cleveland, OH. He is the Principal Investigator for NASA's software-defined and cognitive-radio flight experiment aboard International Space Station. He is a principal architect to define NASA's future communications relay satellite and ground station architecture. He has worked with space communications technology for over 25 years on various satellite, radio, and array antenna technologies. He received his B.S. and M.S. in Electrical Engineering from The University of Toledo and Cleveland State University, respectively. Mr. Reinhart has published a number of technical papers and conference presentations associated with the SCaN Testbed, the application of cognitive technologies to NASA, SDR technology, and the Ka-band Advanced Communications Technology Satellite (ACTS). He is a principal author of the SDR Space Telecommunications Radio System (STRS) architecture, now a NASA-wide standard.

Dale J. Mortensen has worked as an electronics engineer at the NASA Glenn Research Center since 1990. He earned his B.S. and M.S. from The Ohio State University and Case Western Reserve University in 1990 and 1995, respectively. The majority of his work at NASA has supported space communications research and flight projects in the areas of free-space laser systems, high-speed digital modems, software-defined radio, and cognitive communication systems. Dale is a co-author of the NASA's Space Telecommunications Radio System (STRS) architecture standard, and now serves as a systems lead for the Cognitive Communications Project.