The 1580 Spectrum Challenge!!!!

Christopher Rose & Alberto Trovamala '20

***Abstract*—Teams of ENGN 1580 students will implement MATLAB transmitters and receivers that transport bits over a channel. There are three challenges: I] Solo bit delivery in the presence of noise, II] Competitive delivery where two teams try to deliver payloads in the presence of ambient noise/interference and also potentially deliberate jamming from the other team and III] Cooperation where two teams will try to maximize payload transfer collectively and individually.**

I. Introduction

ENGN 1580 is a theory course. And while theory can be exhilarating, there is nothing like trying one's hand at practice. And there is nothing MORE exhilarating than a death match competition 🙂. Since we do not have the hardware (nor the spectrum licenses 🙂) to build actual radio systems, we do the next best thing and implement transceivers in MATLAB. In the **ENGN 1580 Spectrum Challenge** teams of students will build MATLAB transmitters and receivers to carry bits across a channel. In meeting these challenges you will exercise your digital communications knowledge and perhaps even discover new methods, heretofore unknown.

We will supply you with the "arena" on which you can exercise your MATLAB code for testing. For the actual competition, you will supply your code to course staff where it will be run in real time and various performance indicators displayed – sort of a combination between bingo and a horse race. 🙂

There are three component challenges:

Solo: you will design a transmitter/receiver pair that delivers a payload in the presence of ambient noise.

Competition: you will design a transmitter/receiver pair that delivers a payload in the presence of ambient noise and potential jamming from your competition.

Cooperation: you will design a transmitter/receiver pair that cooperates with another team to deliver a payload in the presence of ambient noise.

We now describe the arena in which you will rise to these challenges, and the scoring algorithms to be employed.

II. System Overview

The overall system architecture is depicted in FIGURE 1. You will design a sending transceiver ($\mathcal{S}$) and a receiving transceiver ($\mathcal{R}$). We will denote the output of $\mathcal{S}$ as $\mathbf{s}$ the information-carrying waveform sent over the forward channel ($\mathcal{C}$), and the output of $\mathcal{R}$ as $\bar{\mathbf{s}}$, the information sent over the feedback channel ($\bar{\mathcal{C}}$). The index of these vectors is implicitly the time step, n. The total number of time steps, N, will be fixed for a given competition round. Furthermore, given

$$\mathbf{s} = [s_1, s_2, \ldots, s_N] \tag{1}$$

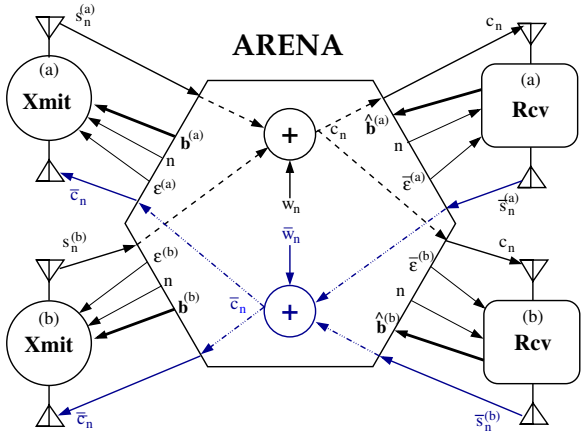

Fig. 1. **ENGN 1580 Spectrum Challenge System Architecture**

and

$$\bar{\mathbf{s}} = [\bar{s}_1, \bar{s}_2, \ldots, \bar{s}_N] \tag{2}$$

we will impose an energy budget $\mathcal{E}$ as

$$|\mathbf{s}|^2 \leq \mathcal{E} \tag{3}$$

and

$$|\bar{\mathbf{s}}|^2 \leq \mathcal{E} \tag{4}$$

It is to allow adaptive modulation (changing the way information is sent in response to measured channel conditions) that both $\mathcal{S}$ and $\mathcal{R}$ are TRANSCEIVERS as opposed to only transmitters and receivers, respectively.

The channel is additive and your $\mathcal{R}$ transceiver will be presented with

$$\mathbf{c} = \mathbf{s}_1 + \mathbf{s}_2 + \mathbf{w} \tag{5}$$

where $\mathbf{s}_i$ are competitor signals and $\mathbf{w}$ is noise provided by the system. Likewise your $\mathcal{S}$ transceiver will be presented with

$$\bar{\mathbf{c}} = \bar{\mathbf{s}}_1 + \bar{\mathbf{s}}_2 + \bar{\mathbf{w}} \tag{6}$$

$\bar{\mathbf{s}}_1$ and $\bar{\mathbf{s}}_2$ are competitor feedback signals and $\bar{\mathbf{w}}$ is also noise supplied by the system (which may or may not be independent of $\mathbf{w}$). The components of $\mathbf{s}$ and $\bar{\mathbf{s}}$ will be provided in sequence to $\mathcal{C}$ and $\bar{\mathcal{C}}$ respectively which will in turn instantaneously supply $\mathbf{c}$ and $\bar{\mathbf{c}}$ components to your $\mathcal{R}$ and $\mathcal{S}$, respectively. Unlike a real communication system where propagation delay and clock drift can wreak havoc, there will be no need for timing/carrier recovery as we will assume the transceivers are perfectly synchronized. We have also taken mercy on you and not distorted the channel (LTI Filter interposed between your $\mathbf{s}/\bar{\mathbf{s}}$ and the receivers – maybe next year's students will be so tortured!).

Your $\mathcal{R}$ must produce a vector of bits containing your estimate of the payload vector of bits provided to $\mathcal{S}$. Performance will be judged on total energy used

$$\mathcal{E}_{\text{tot}} = |\mathbf{s}|^2 + |\bar{\mathbf{s}}|^2 \leq 2\mathcal{E} \tag{7}$$

and the number of correct bits in the payload bit-vector estimate.

Please note that you may never exceed your energy budget(s). That is, we require

$$s_K^2 \leq \mathcal{E} - \sum_{k=1}^{K-1} s_k^2 \tag{8}$$

with a similar restriction for the feedback channel. If an attempt is made to defy equation (8), then the offending sample will be clipped to meet the energy requirement, after which subsequent samples will be set to zero. The forward and backward channels will operate independently, so an offense on one channel will not affect the other.

III. The Challenges

Your team will design two separate programs $P_{\mathcal{S}}$ and $P_{\mathcal{R}}$. The inputs to $P_{\mathcal{S}}$ are the payload bit-vector, $\boldsymbol{v}$ (to which $P_{\mathcal{S}}$ will have complete access so that you may "package" it for transmission as you'd like), and at any given time, the feedback $\bar{c}_n$. The output of $\mathcal{S}$ at a given time n is s_n which will be used to compute the channel output c_n. Likewise, the outputs of $\mathcal{R}$ are $\bar{s}_n$ and, ultimately, the estimate of the payload bit-vector, $\hat{\boldsymbol{v}}$, $\mathcal{R}$'s input is c_n.

There are also ancillary inputs $\Sigma_{\mathcal{E}}(n)$ and $\bar{\Sigma}_{\mathcal{E}}(n)$, the energy consumed by $P_{\mathcal{S}}$ and $P_{\mathcal{R}}$ transmissions, respectively, up to and including time n. A given competitor's energy usage will be visible only to them and not the opponent. In addition, the Px energy usage will not be available to Pf and *vice versa*. In contrast, the time n will be globally available to $P_{\mathcal{S}}$ and $P_{\mathcal{R}}$ for all competitors.

A. CHALLENGE I

Design an $\mathcal{S}/\mathcal{R}$ pair that takes the supplied bit-vector and generates a waveform to be sent over the channel and decoded. Both the forward and backward channels will be corrupted by noise of at least two types: Gaussian noise and Mystery noise designed to bedevil you. The noise intensity may vary between trials, but will be known.

B. CHALLENGE II

Design an $\mathcal{S}/\mathcal{R}$ pair for spectrum combat. As in Challenge I you will attempt to send a bit-vector from sender to receiver, but will do so in the presence of interference from your rival. You may also choose to use some of your energy budget $\mathcal{E}$ or point budget N to explicitly jam your rival. Such jamming may be employed on both the $\mathcal{C}$ and $\bar{\mathcal{C}}$ channels as you wish. Each $\mathcal{R}$ will have shared access to the output of $\mathcal{C}$. And each $\mathcal{S}$ will have shared access to the output of $\bar{\mathcal{C}}$.

C. CHALLENGE III

Design a and $\mathcal{S}/\mathcal{R}$ pair as in Challenge II, but this time you and your opponent attempt to stay out of each other's way while still delivering your bit-vectors.

IV. Testing And Submission Details

Our 1580 Spectrum Challenge Rig is written in MATLAB. We will supply the arena program so you can test your programs on your own machines. For competition, you will supply us with your separate transmitter and receiver programs/routines and we will run them in real time on data payload bit-vectors of our choosing, reporting on the status of bits delivered and energy used in real time. We may also display competitor waveforms in real time.

Details about the programming interface follow.

A. The Arena

The main program is *testing_ground*. This is the program that we will use to run your code. You should not modify this program except to pass it arguments. The Arena execution discipline can be summarized as follows

1) $P_{\mathcal{S}}$ is provided:
   a) current logical time-step n
   b) transmit energy remaining $\mathcal{E}_n$
   c) feedback $\bar{c}_{n-1}$
2) $P_{\mathcal{S}}$ returns:
   a) signal output s_n

Likewise,

1) $P_{\mathcal{R}}$ is provided:
   a) current logical time-step n
   b) transmit energy remaining $\bar{\mathcal{E}}_n$
   c) forward channel output c_{n-1}
2) $P_{\mathcal{R}}$ returns:
   a) signal output $\bar{s}_n$

In addition, $P_{\mathcal{S}}$ will have access to the entire payload bit-vector, called **b**. Obviously, $P_{\mathcal{R}}$ will NOT have such access Furthermore, both $P_{\mathcal{S}}$ and $P_{\mathcal{R}}$ have access to a global time vector (MATLAB ish) as well as the archives of their channel inputs and outputs. Specifically, $P_{\mathcal{S}}$ has access to $\mathbf{s}$ and $\bar{\mathbf{c}}$ while $P_{\mathcal{R}}$ has access to $\bar{\mathbf{s}}$ and $\mathbf{c}$. All these vectors are initially zero and are populated in real time as data/feedback is sent by $P_{\mathcal{S}}$ and $P_{\mathcal{R}}$ respectively.

Time Implementation Detail: Apparently, the vagaries of MATLAB require a staggered approach to program execution. Thus, while logically there is a single time index n, in terms of implementation, $P_{\mathcal{S}}$ operates during even time steps and $P_{\mathcal{R}}$ operates during odd time steps. Thus, the logical n is actually $n = \lfloor \frac{k}{2} \rfloor$ where k is the system clock. So *Fs*, the sampling rate parameter is twice the rate of the logical clock. Thus N, the total number of samples sent and received during a competition round is $N = t\_max \cdot \text{Fs}/2$.

MODE CONTROL: *game_mode* $\Rightarrow$ *test*, *fight* or *co_op*

- *test* $\Rightarrow$ $P_{\mathcal{S}}$ and $P_{\mathcal{R}}$ are to be run with a single participant.
- *fight* $\Rightarrow$ ${P_{\mathcal{S}}}^A$ and ${P_{\mathcal{R}}}^A$ are to be run as a competition with opponent ${P_{\mathcal{S}}}^B$ and ${P_{\mathcal{R}}}^B$.
- *co_op* $\Rightarrow$ ${P_{\mathcal{S}}}^A$ and ${P_{\mathcal{R}}}^A$ are to be run as a competition with cooperator ${P_{\mathcal{S}}}^B$ and ${P_{\mathcal{R}}}^B$.

SCORING:

- *test*

$$S = \Theta_1 - \lambda \frac{|\mathbf{s}_1|^2 + |\bar{\mathbf{s}}_1|^2}{2\mathcal{E}}$$

- *fight*

$$S = (\Theta_1 - \Theta_2) - \lambda \frac{(|\mathbf{s}_1|^2 + |\bar{\mathbf{s}}_1|^2) - (|\mathbf{s}_2|^2 + |\bar{\mathbf{s}}_2|^2)}{2\mathcal{E}}$$

$$\text{winner} = \begin{cases} \text{user 1} & S > 0 \\ \text{user 2} & S < 0 \\ \text{tie} & S = 0 \end{cases}$$

- *co-op*

$$S = \min_i \Theta_i - \lambda \frac{|\mathbf{s}_1|^2 + |\mathbf{s}_2|^2 + |\bar{\mathbf{s}}_1|^2 + |\bar{\mathbf{s}}_2|^2}{4\mathcal{E}}$$

where Θ_i is the number of correct bits for user i, $\lambda > 0$ is a weighting factor and $\mathcal{E}$ is the energy budget.

VARIABLES: A list of variables and their descriptions are is supplied below. The same variables are listed in FIGURE 2 to show their scope (to what functions they are available when).

Local Variables

- Function Handles @
  - *tra_i*
  - *rec_i*

  IMPORTANT: The placeholders for your $\mathcal{S}$ and $\mathcal{R}$ functions are *tra_i* and *rec_i* ($i \in \{1, 2\}$), respectively, preceded by @. For example if you are testing your solo challenge, using a transmitter function called 'my_tra' and a receiver function called 'my_rec' just set game_mode to *test* and then set *tra_1* to @my_tra and *rec_1* to @my_rec. If you are in a competition mode with two transmitters and two receivers, then set the four variables *tra_i* to @my_tra_i and *rec_j* to @my_rec_j, $i, j \in \{1, 2\}$.
- P_* Scratchpad Input:
  - *scratchpad_tra_i*
  - *scratchpad_rec_i*
- P_* Scratchpad Output
  - *new_scratchpad_tra_i*
  - *new_scratchpad_rec_i*

  "Scratchpad" arrays are arrays that maybe be input to or output from your program to maintain continuity over time with application of your functions. *scratchpad_tra* and *scratchpad_rec* are input arrays for your $P_{\mathcal{S}}$ and $P_{\mathcal{R}}$ programs, respectively. Likewise, your programs can do their calculations at each step and output *new_scratchpad_tra* and *new_scratchpad_rec* You can use these variables any way you'd like. You could use them to keep track of which modulation scheme you employ, for how long you should send signals related to a bit, and/or the message you want to send. I decided not to just input the message you have to send because this might make tracking what bit you have to send harder. For instance, suppose you wanted to interleave bits (send bits in a (known to the receiver) pseudo-random order so that under a time-varying noise scenario (as you'll encounter with competitive signaling) you'd not be subject to bursts of bit errors. You could load your scratchpad accordingly and not have to worry about applying the interleave for each sample time. Likewise, you could use scratchpads as queues/stacks or sorts, popping a value off the end as each bit was sent. Or not. Completely up to you.
- Energy Remaining
  - *e_tra_i*
  - *e_rec_i*

  These are respectively the energies remaining in your transmitter and receiver, which will receive them as inputs. It is important to notice that each transceiver will only receive the relevant variable and not both. If either of your transceivers tries to exceed this energy budget the result will be set to zero.
- Signal Outputs s_n & $\bar{s}_n$
  - *signal_tra_i*
  - *signal_rec_i*

  These are the signals outputted by your transceivers. Notice that they are scalars and not arrays. This is because you are only allowed to output a single point of your waveform per time loop. These outputs will be passed through the channel and the resulting values will be archived in r_tra and r_rec.
- Bit Payloads for $\mathcal{S}$ *b_i* : This is the original message you have to pass across the channel. It is a long one-dimensional bit array. It will be given as an input to your transmitter at the first time loop. Obviously, it will not be available to your receiver.
- Personalized Payloads *my_msg_i*: In order to make it simpler for you to keep track of which bits you are sending or have sent, you will be able to receive as an input your personalized version of the payload. If, for example, you just sent a bit through and don't want to keep track of what to send next, you can set my_msg = my_msg(2:end) and always send the current first element. If you instead prefer using a counter you can just keep this variable untouched.
- Archive of $\bar{\mathcal{C}}$ *r_tra*: A record of everything that has been sent through the transmitter channel. An array on which each point represents the channel output at the respective time loop. Your transceivers will get this as an input.
- Archive of $\mathcal{C}$ *r_rec*: A record of everything that has been sent through the receivers channel. An array on which each point represents the channel output at the respective time loop. Your transceivers will get this as an input.

Global Variables

- Noise Power *sig*: The value of a standard deviation in the channel's Gaussian noise. You will not be able to modify this or any other global variable during the challenge.
- Master Clock Rate *Fs*: The frequency of the time loops. Notice that, because transmitters and receivers work respectively at odd and even time loops, the actual maximum sampling frequency will be half of Fs.
- Time Vector *t*: An array listing the time at each time loop. Can be matched with the archives to know when each value was sent.
- Current Time Index *n*: the index for the time array that indicates what the current time is. t(n) is the time during the current time loop.
- $\mathcal{S}$ Energy Budget $\mathcal{E}$ *initial_e_tra*: Initial energy budget given to the transmitters.
- $\mathcal{R}$ Energy Budget $\bar{\mathcal{E}}$ *initial_e_rec*: Initial energy budget given to the receivers.
- Maximum # of bits *res_size* : This is the number of bits in the payload and the maximum number of bits you can output before we stop recording your answers. It is also quite large, meaning that it is unlikely you will send his many bits.

Scoring Variables

- Bits to be scored *new_bits_rec_i*: This will be an output from your receivers. Whenever you are done computing what some bits are, you can insert them here and we will save them. This will have to be an empty array during most time loops, as it will take quite a few values for your receiver to identify bits. It is an array, so you can output bits one at a time or in any other number you please.
- Bits correct *res_i*: This will be a running tally of how many bits have been identified and given as output through new_bits_rec_i correctly. It is also the value used to graph your results during the competitions.
- Final Score *result_i*: This is your final score, computed at the end of a challenge as explained in the scoring section.

FIGURE 2 lists variables and their scope (which program modules have access to which variables).

V. Transceiver I/O

A. Transmitter:

INPUT:

- r_tra ⇒ transcript of everything sent so far over the transmitters channel.
- r_rec ⇒ transcript of everything sent so far over the receivers channel.
- t ⇒ time vector.
- n ⇒ time vector index indicating current time.
- e_tra ⇒ energy left for the transmitter.
- scratchpad_tra ⇒ scratchpad for the transmitter.
- my_msg ⇒ personalized payload.

OUTPUT:

- signal_tra ⇒ next point in the transmitted waveform.
- new_scratchpad_tra ⇒ updated transmitter scratchpad.
- new_msg ⇒ updated personalized payload.

B. Receiver:

INPUT:

- r_tra ⇒ transcript of everything sent so far over the transmitters channel.
- r_rec ⇒ transcript of everything sent so far over the receivers channel.
- t ⇒ time vector.
- n ⇒ time vector index indicating current time.
- e_rec ⇒ energy left for the receiver.
- scratchpad_rec ⇒ scratchpad for the receiver.

OUTPUT:

- signal_rec ⇒ next point in the transmitted waveform.
- new_scratchpad_tra ⇒ updated receiver scratchpad.
- new_bits_rec ⇒ confirmed bits for the system to save.

VI. Using the Arena

OTHER USEFUL FUNCTIONS:

- *channel* ⇒ simple function that just sums the given signals with noise of power *sig*.
- *energ* ⇒ subtracts the current available energy and the energy spent for the last point in the signal.

EXAMPLES: As examples, we provide simple functions *send_1*, *send_2*, *reci_1*, *reci_2* so you can see how two receivers and two transmitters could work and then exercise the arena in various modes.

IMPORTANT: It may take a lot of c_n values for your receiver to find out what bit is being transmitted, so for most iterations in your loop your bit output should be [], an empty array, so that nothing new gets saved.

Good Luck!

If you have any questions about the software, feel free to contact Alberto at Alberto_Trovamala@Brown.edu. Chris Rose is a MATLAB naif and will be mostly useless for practical matters (OH but if you knew C!!!!). However, he might actually be useful for questions about theory and general approach 🙂.

DESCRIPTION	**ARENA**	$P_{\mathcal{S}1}$	$P_{\mathcal{S}2}$	$P_{\mathcal{R}1}$	$P_{\mathcal{R}1}$
function handles @		tra_1	tra_2	rec_1	rec_2
P_* Scratchpad Input (array)		scratchpad_tra_1	scratchpad_tra_2	scratchpad_rec_1	scratchpad_rec_2
P_* Scratchpad Output (array)		new_scratchpad_tra_1	new_scratchpad_tra_2	new_scratchpad_rec_1	new_scratchpad_rec_2
Energy Left (scalar)		e_tra_1	e_tra_2	e_rec_1	e_rec_2
Signal Outputs s_n & $\bar{s}_n$ (scalar)		signal_tra_1	signal_tra_2	signal_rec_1	signal_rec_2
Bit Payloads for $\mathcal{S}$ (array)		b_1	b_2		
Personalized Payloads (array)		my_msg_1	my_msg_2		
Archive of $\mathcal{C}$ (array)		r_rec	r_rec		
Archive of $\bar{\mathcal{C}}$ (array)				r_tra	r_tra

Noise Power (scalar) Master Clock Rate (scalar)	sig Fs				
Time Vector (array)	t				
Current Time Index (scalar)	n				
$\mathcal{S}$ Energy Budget $\mathcal{E}$ (scalar)	initial_e_tra				
$\mathcal{R}$ Energy Budget $\bar{\mathcal{E}}$ (scalar)	initial_e_rec				
Maximum # of bits (scalar)	res_size				

Available at End of Round					
Bits to be scored (array)				new_bits_rec_1	new_bits_rec_2
Bits correct (scalar)				res_1	res_2
Final Score (scalar)				result_1	result_2

Fig. 2. **Variable Names and Scopes:** column 1 is a the variable description. Columns 2 through 6 list variable accessible to the named program module.