Space Communications Toolbox

Introduction

When the preliminary design for a space mission is performed, usually, a huge number of configurations is tested in a short amount of time to identify a promising concept. This means that the analysis is often simplified and several parameters are based on suppositions in order to quickly iterate a new design concept. Often the ease of use of the tool is very important and this is the main reason behind this communications toolbox.

This Space Communication Toolbox will be used as an educational tool for quick mission design thanks to a modular structure and an easy to use interface. It is targeted at Earth-bound and deep space missions.

This toolbox will be used for the complete design of the communications and radio-navigation sub-systems, allowing to quickly estimate their performances at the preliminary design stage.

Assignment

Create the structure of the communications toolbox: both a data processing part (back-end) and a graphical user interface (front-end or GUI) are needed. The assignment is focused on creating the backbone of the toolbox by implementing the processing logic for a link budget with minimal focus on the GUI.

Python has been selected as programming language as this is usually the preferred language by students. The selection is not mandatory and it can be discussed.

The toolbox should use a GUI that makes it easily accessible and that would not require extensive training, especially because it is supposed to be used by students. The selected language is HTML, such that the toolbox can be easily accessible via a web browser. Brython (https://brython.info/) was selected as a framework, allowing Python code to be easily integrated in a simple web page. An example for a web page will be provided, showing how to generate a simple table.

The link budget output is expected to be in table format, as in the example below. The table comes from the GAIA Space – Ground ICD (the full document will be provided as an example, together with similar ICDs). For this first iteration, the focus would be on implementing only the nominal values (NOM column). A more general description on the link budget can be found here http://everyspec.com/ESA/download.php?spec=ECSS-E-ST-50-05C_REV-2.048184.pdf).

LINK ID : P35GAIA **PAGE 3/4**
DATE : 17/06/2004 18:12
ORBIT : Lissajous L2 ALTITUDE (1000km): 1680
STATION : Perth 35-m ELEVATION (deg): 5

TELECOMMAND BIT RATE (kb/sec) : 2.00 RANGING : No
TELEMETRY BIT RATE (kb/sec) : 5000.00 with CONCAT. CODING : Yes

BASIC DOWNLINK (1/2)

	NOM	ADV	FAV	MEAN	VAR	PDF
S/C TX POWER.....dBW	*17.00*	*17.00*	*17.00*	17.00	0.00	TRI
DIPL. CIRCUIT LOSS....dB	*0.00*	*0.00*	*0.00*			
RFDU CIRCUIT LOSS.....dB	*0.30*	*0.30*	*0.30*			
CABLE LOSS.....dB	*0.26*	*0.30*	*0.26*			
VSWR, overall.....:1	*1.20*	*1.20*	*1.15*			
VSWR LOSSES.....dB	0.04	0.04	0.02			
TOTAL LOSS.....dB	0.60	0.64	0.58	0.61	0.00	UNI
S/C TX ANT GAIN.....dBi	*17.60*	*16.80*	*18.40*	17.60	0.11	TRI
S/C ANT TX AXIAL RAT..dB	*1.50*	*1.50*	*1.50*			
POINTING LOSS (*)..dB	*0.00*	*0.00*	*0.00*	0.00	0.00	TRI
EIRP S/C.....dBW	34.00	33.16	34.82	33.99	0.11	
FREQUENCY.....GHz	8.50	8.50	8.50	8.50		
SLANT RANGE.....1000*km	1686	1686	1686	1686		
PATH LOSS.....dB	235.57	235.57	235.57	235.57		
ATMOSPHERIC LOSS.....dB	0.50	0.60	0.40	0.50	0.00	GAU
IONOSPHERIC LOSS.....dB	*0.00*	*0.00*	*0.00*	0.00	0.00	GAU
COPOLAR ANT-GAINS(Y=1/N=0)?	*0*					
POLARISATION MISMATCH.dB	0.06	0.09	0.03	0.06	0.00	UNI
TOTAL PROPAG. LOSS....dB	236.12	236.26	236.00	236.13	0.00	
FLUX at G/S.....dBm/m^2	-131.52	-132.36	-130.71	-131.54	0.11	
POWER FLUX DENS..dBW/m^2	-164.99	-163.46	-166.66	(in 4 kHz)		
MAXIM FLUX DENS..dBW/m^2	-150.00	-150.00	-150.00	(S- or X-Bnd)		
FLUX MARGIN.....dB	14.99	13.46	16.66			
G/S RX ANT GAIN.....dBi	68.00	68.00	68.00	68.00	0.00	UNI
POINTING LOSS.....dB	*0.30*	*0.40*	*0.00*	0.20	0.01	UNI
G/S ANT RX AXIAL RAT..dB	*0.50*	*1.00*	*0.00*			
SYSTEM NOISE TEMP....dBK	17.90	18.40	17.90	18.15	0.01	GAU
RX G/T.....dB/K	50.10	49.60	50.10	49.85	0.01	
RX S/No.....dBHz	76.28	74.71	77.52	76.11	0.13	
S/N in RANGING BANDWIDTH						
S(Tone)/N in Videobd..dB	-12.13	-15.26	-9.53			
S(TC)/N in RG-Videobd.dB	-7.66	-5.27	-10.56			

Implementation

The target for this assignment is creating a modular framework to calculate link budgets using a web-based GUI and a Python back-end. The most important part is creating a backbone software structure to handle standard blocks that can be customized to implement all the required functionalities. Object-oriented programming is highly recommended, taking advantage of classes and inheritance in Python (the final solution is to be discussed).

A suggestion on the implementation would be to implement a standard block that provides the following interfaces:

- getName(): this can be used to identify each block providing a user-friendly name. This function is expected to return a string to be displayed in the GUI or to be used to identify the block.
- process(numeric value): this function expectes a numeric value representing the input of such block and it returns a numeric value as a result of the calculations. This function is used to provide the input to the processing block and retrieve the output (single output). This can be, for example, the EIRP input for a block calculating the free-space loss.
- getParameters(): this function is expected to return a python dictionary (ex. {"valueA": 1.1, "valueB": 5.4, "valueC": "setting"}) with the list of parameters and values. Parameters are values that the user could potentially edit, for example the distance ground – spacecraft for a block calculating the free-space loss. One or more parameters might be strings selecting, for example, different algorithms to be used for the calculation.
- setParameters(dictionary): this function expects a python dictionary (ex. {"valueA": 1.1, "valueB": 5.4, "valueC": "setting"}) with the list of parameters and values to be set. Parameters are values that the user could potentially edit, for example the distance ground – spacecraft for a block calculating the free-space loss.
- getResults(): this function is expected to return a python dictionary (ex. {"valueA": 1.1, "valueB": 5.4, "valueC": "setting"}) with the list of results and values. Results are values that the user cannot edit and they provide important values as a result of the calculations done in the block.

Three main classes of blocks will have to be implemented:

- “Source” blocks: this is a block not requiring any input, like a satellite transmitter (being the first element in a chain). The transmitter power can be set using a parameter but no input is required for the functioning of the block.
- “Intermediate” blocks: these are blocks having an input and an output as they are used in the middle of the chain.
- “Sink” blocks: these are supposed to be used at the end of the chain, with no block afterwards.

Considering the three block types, the main software is supposed to implement a generic link budget by chaining several blocks one after the other (by tying the output of one block to the input of the next block): the sequence would have to be specified in an external file, providing the configuration parameters as well. The format for this configuration file is open for discussion, a suggestion would be YAML:

```yaml
# An employee record
martin:
  name: Martin D'vloper
  job: Developer
  skill: Elite
```

Such configuration file could provide a sequence of blocks and parameters, used to generate the link budget structure, and then the python code could execute the code.

A minimum subset of blocks is expected to be created:

- Signal source using EIRP
- Free-space loss for a LEO satellite with a ground station as a function of altitude and elevation angle
- Atmospheric losses as a function of oxygen and water vapour (and elevation)
- General loss block
- General gain block
- Radio receiver
- Link margin

If time allows more blocks can be considered:

- Antenna gain considering also pointing losses (eventually loading an antenna pattern)
- Coding gain using pre-defined sets of codes
- More upon discussion

Connection to the GUI has the lowest priority: Bryton was selected as a framework as it allows to create static webpages (as compared to Django, for example, simplifying hosting). To be able to use the application, a web server would be needed. This can be accommodated by simply running:

python -m SimpleHTTPServer 8000

in the folder containing the HTML and Python files. The website can be displayed on a browser at the address: http://localhost:8000/index.html

The goal for this tool would be to make it available to external users, potentially making it also open-source (with explicit consent from the authors), ex. on our GitHub page (https://github.com/delfispace).