Python Subroutines to Format Spacecraft Telemetry

Nina Rastogi

Electrical Engineering

Mentor: Professor Tim Rogstad

Kellogg Honors College Capstone Project

1. Background

When transferring data to and from spacecraft, there can be loss and alteration in the data due to noisy transmission channels. Some of these include radiation effects in space-based buffers, on-board protocol conversion anomalies, flight computer processing load issues, noise in space-ground transmissions, and file transfer protocol anomalies.[1] By implementing the BlackJack Data Link Protocol into a python byte-oriented state machine program called ReadStuff, the data loss and altercation can be mitigated and maintain the integrity of the data.

2. Objective

To write a python program that formats spacecraft telemetry utilizing the BlackJack Data Link Protocol.

3. Materials & Methods

The code was written in python version 3.6.0. An integrated development environment (IDE) called PyCharm was used to write and test the program.

In addition, different libraries were imported in order to maximize the efficiency of the program. These libraries include numpy, sys, binascii hexlify, intertools, and crcmod.

4. Results & Discussion

The State Machine (Definitions):

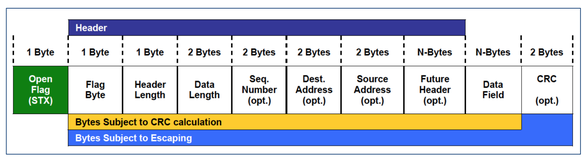

Figure 1. Complete packet description of the path for the raw data.

In figure 1, the format of the raw data file is shown. The first 11 bytes of the data make up the open flag to the source address. The CRC (cyclic redundancy check) consists of everything from the flag byte to the data length.

```
def __init__(self):
    self.FlagByte= None #definitions of each field
    self.HeaderLength = None
    self.DataLength = None
    self.SeqNumber = None #optional
    self.DestAddress = None
    self.SourceAddress = None
    self.FutureHeader = None #optional
    self.DataField = []
    self.CRC = None
```

Figure 2. Defining each of the states.

In figure 2, the definitions for each of the states for the state machines were created. This was all done under the *class Packe*t. The *self* is a reference to an object, and it's very similar to the concepts of *this* in a C program.

The State Machine (Cycling Through the Paths):

Figure 3. The defined state machine that takes an input raw file and outputs a list.

Figure 3 shows the defined Run_Statemachine. It reads the input raw rile, and outputs a list. The different states are numbered from 0-9, ending with the CRC. The state, current (state) both start at zero. The state machine will read a byte, and cycle through states until it finds an *if* statement that it matches. The open flag is denoted as x02, and once the state machine finds a x02 in the file, it will start the packet processing.

The State Machine (Cycling Through the Paths):

In figure, there is an *interp* function, which is taken from the numpy library. This function performs an interpolation on the unpacked file of the byte and the next state. Due to the s*truct.unpack* function, which unpacks a string according to the given format, it was necessary to define it by adding an big endian ('>h') to make it a short.

Reading the File:

The file fed into the state machine using a *sys.argv[]*. The *sys.argv[]* is a list in Python which has the command-line arguments passed to the script. This makes it easy for the user to input any file of choice into command line and run the program. ReadStuff is written so that if there is no file inputted into command line, it will automatically use the data file used for this project.

The Cyclic Redundancy Check:

```
elif state == 9:
        #int.from_bytes(byte, byteorder='big')
        Other=[int(hx) for hx in byte]
        crc16 = crcmod.mkCrcFun(Other)
        print (crc16)
        state = 0
```

Figure 4. CRC in the state machine.

Figure 4 shows the implementation of the cyclic redundancy check. The "Other" coverts the bytes into integers, making it compatible with the crc16, which is found under the crcmod library. After the crc16 is printed, the state will go back to state 0.

The cyclic redundancy check is a code that is used for detecting errors in the code. The files with clocks of data have a short check value attached to them. This check value is based on the remainder values of a polynomial division of the file contents. Once this number is retrieved, the calculation is repeated in the code and if the check values from the code and file do not match, there is an error in the code. The CRC polynomial can be hard coded, or implemented through the library crcmod.

5. Conclusion

The byte stream state machine was successfully implemented, listing the raw data into the desired order of fields. The cyclic redundancy test was successfully implemented using crc16The SNR plots were not able to be plotted, due to time constraints.

6. Future Work

An implementation of the signal to noise ratio plots (SNRs) will be conducted.

7. References

1. *A.H. Farrington BlackJack Data Link Protocol Interface and Implementation Description, JPL Document, 20675 (2001): p 1-61.*
2. *BlackJack Telemetry Dictionary, California Institute of Technology, GRACE 327-540 Rev 2.1 (2001): p 1-3.*
3. *BlackJack Command Dictionary, California Institute of Technology, GRACE 327-540 Rev 2.1 (2001): p 1-2.*

8. Acknowledgements

I would like to acknowledge my JPL mentor Tim Munson for his guidance, support, and patience. I would also like to thank my faculty advisor Tim Rogstad for allowing me to start this project. In addition, I would like to gratefully acknowledge support from my JPL coworkers and Cal Poly Pomona students Bradley Lunsford, Stephan Esterhuizen, and Giorgio Savastano.