[illegible] 3-[illegible]0

3[illegible]38[illegible]

358170

p. 8

1997

NASA/ASEE SUMMER FACULTY FELLOWSHIP PROGRAM

MARSHALL SPACE FLIGHT CENTER
THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

SPECTRUM PREPROCESSING IN THE OPAD SYSTEM

Prepared by:	Dr. Constantine Katsinis, Ph.D.EE
Academic Rank:	Associate Professor
Institution:	University of Alabama in Huntsville
Department:	Electrical and Computer Engineering
NASA/MSFC:	
	Astrionics Laboratory
	Instrumentation Branch
MSFC colleagues:	W.T. Powers and Anita Cooper

Introduction

To determine the readiness of a rocket engine, and facilitate decisions on continued use of the engine before servicing is required, high-resolution optical spectrometers are used to acquire spectra from the exhaust plume. Such instruments are in place at the Technology Test Bed (TTB) stand at Marshall Space Flight Center (MSFC) and the A1 stand at Stennis Space Center (SSC). The optical spectrometers in place have a wide wavelength range covering the visible and near-infrared regions, taking approximately 8000 measurements at about one Angstrom spacing every half second.

In the early stages of this work, data analysis was done manually. A simple spectral model produced a theoretical spectrum for given amount of elements and an operator visually matched the theoretical and real spectra. Currently, extensive software is being developed to receive data from the spectrometer and automatically generate an estimate of element amounts in the plume. It will result in fast and reliable analysis, with the capability of real-time performance. This software is the result of efforts of several groups but mainly it has been developed and used by scientists and combustion engineers, in their effort of understanding the underlying physical processes and phenomena and creating visualization and report generation facilities.

Most of the software has been developed using the IDL language and programming environment which allows for extensive data visualization. Although this environment has been very beneficial, the resulting programs execute very slowly and are not easily portable to more popular, real-time environments. The need for portability and high speed of execution is becoming more apparent as the software matures moving out of the experimentation stage and into the production stage where ease of use and short response time are the most desirable features.

The purpose of the work described here is to assist the scientists who developed the original IDL-based version in the conversion of the software into the real-time, production version. Specifically, a section of the software devoted to the preprocessing of the spectra has been converted into the C language. In addition, parts of this software which may be improved have been identified, and recommendations are given to improve the functionality and ease of use of the new version.

Facilities

The Diagnostic Test Facility (DTF) at Stennis Space Center has a small rocket engine with different characteristics to the SSMEs that has been seeded with known amounts of alloys during firing. DTF data were produced with the OMA Spectrometer, and will be used to confirm the spectral model and methods for handling the OH component. The TTB (Technology Test Bed) data is gathered at MSFC using the OPAD instrument, together with engine data on the tests. There is also data from High RAS, which is the new resolution spectrometer with 3 banks.

Data from different instruments is stored in the directories dffdata (DTF data), rawdata (TTB data), raw3data (TTB data from High RAS 3), omadata (TTB data from the OMA), engdata (engine data for TTB). Tests are identified by two parameters. The first one defines an integer indicating the instrument and engine from which the data taken. The available options are: 1=ttb, 2=dtf, 2= high/ras, 3= oma from TTB. The second parameter defines the test, an integer number for the TTB test or a string representing the DTF test.

The naming convention of data files is not always obvious since they were created by different scientists at different locations and times. For example, in one case names were used to indicate the level of "plume seeding" which took place during the test.

Preprocessing of spectra

Spectrum preprocessing is the first of a sequence of steps, the final purpose of which is to determine the amounts of metals present in the plume. The input of the preprocessing step is the data produced by the spectrometer. Its output is a refined version (defined below) of the spectrum, which in following steps is further processed by the neural-network code and the SPECTRA code.

Each spectrum of the engine plume has a component due to metal erosion, but it also has components due to the emission of the OH and H_2O molecules generated by the burning of hydrogen in the engine. In addition, scattered background light produces another spectrum component. The major purpose of the initial stages of the software, the preprocessing stages, is to extract the first component (due to metal erosion) of the spectrum. Subsequent stages of the software use the theoretical model to predict the metal amounts in the plume. The OH component can vary from test to test, and is generally indicative of the power level, the fuel mixture ratio, and of change in power level. Unfortunately, the complex interaction of the OH and water vapor emissions are poorly understood and little quantitative data are available that would permit development of an accurate model. Currently a basic understanding of the process has resulted in the development of a simple method to extract the OH component of a spectrum and leave an estimate of the metallic component. This method is being improved since it is not successful during the critical start-up and shut-down phases of an engine test.

As Figure 1 shows, the preprocessing step is implemented by the following major routines (which in turn use numerous other routines for file handling and data processing):

1) GETTEST(test_number, wave_array, time_array, data_array)

Reads a wave array with wavelengths of the spectra, a time array with the time of each spectrum and a data array with a sequence of spectra. Data is in files originating from experiments at TTB (RAS or Highras or OMA) or DTF specified by test_number.

2) STANDARD_SCAN(wave_array,sscan)

Reads a clean metal-free spectrum from disk for use by the CLEAN program. The supplied wave array indicates the desired wavelengths at which the metal-free spectrum should be returned.

3) START_CLEAN(dist,magn,grid,wavarray,sscan)

Routine CLEAN is called repeatedly to process a sequence of spectra. It needs a set of data (interpolation parameters) which stays permanently in memory until the wavelength array is changed. Routine START_CLEAN initializes the data set for the CLEAN routine.

4) CLEAN(datarray,cleanarray)

Receives a sequence of spectra in data array and produces a cleaned sequence of spectra in clean array by removing the OH portion of a spectrum to leave the metal region.

5) SMOOTHAWAY(cleanarray,newarray)

Routine CLEAN may cause negative values to occur in the clean array. Routine SMOOTHAWAY smoothes the clean array to remove the negative values.

File data formats

Data format of the TTB files

The TTB (Technology Test Bed) data appears over a broad wavelength range of 280 nm (2800 Angst) to 718 nm (7180 Angst) thus covering the short UV to Visible wavelength range. The instrument has two detectors covering the 2800 to 5010 Angstrom, and 5010 to 7180 Angstrom range. The representative pixel width of 24.4 μm. It is possible, due to the efforts made to maximize sensitivity of each detector by various focus enhancing optical elements, to observe distortions in line shape and position. If the input optics do not fill the entrance slit with light, or if the divergence through the slit is insufficient to fill the mirrors and the grating, the line may shift sideways or become distorted. Detector 2 (B), which covers the 5010 to 7080 Angstrom range, has a skewed instrument profile as well as some non-linear behavior in wavelength shift. Each test firing is assigned a three-digit number (xxx). The detectors are calibrated at the start of each test firing and the resulting data files are stored at directory "rawdata/rascals/" with file names "inrespl.xxx" and "inresp2.xxx". One source of error is that the calibration factors, which allow conversion of data from Volts to Counts, are hard coded with the assumption that there is no drift in such instrument dependent parameters.

Two data files, with sampled data and corresponding time, are stored in binary format in the directory "rawdata/rasxxx" with names "ras.052" and "time.052".

Based on the detector to wavelength grid relation established by calibrations performed in the lab, the wavelength array corresponding to sample points is deduced. This conversion can be a source of error. The sampled data points for each scan are recovered, and processed to account for background data. The data is also calibrated with respect to the detectors which are re-calibrated at the start of the test firing.

Data format of the DTF files

The DTF (Diagnostic Test Facility) data files are supplied in binary format and located in the directory "dtfdata". The spectral coverage range for DTF is 300 nm to 430 nm. The first record in the data file contains the test number, total number of scans, total number of sample points, sample scan rate, time of engine start, and type. Files created at later times contain a more extensive first record. After the first record, subsequent data is organized into a record of four numbers indicating the wavelength at the sample point, the response function point value, the measured background data, and the data with the background subtracted.

Data format of the OMA data of the TTB

The OMA is the same kind instrument used to gather the DTF data, but the data files have a different format. For each test (xxx), there are two files omaxxx.bin with the raw data and caixxx.bin with the calibration data.

Conclusions and Recommendations

The major function of the preprocessing code is to remove the effect of the OH lines on the spectrum and extract the spectrum part due to metal erosion. It appears to have grown within a relatively long period of time as scientist sought to experiment with different ways of implementing various subfunctions. In addition, some programs have a large number of options (keywords) which select one of many possible processing paths. Although such software can be useful during experimentation, the resulting size and complexity make maintenance and optimization difficult. It would be very useful to closely examine the processing steps in this software and identify the ones that are absolutely necessary for the real-time (production) version of the software. Activities should be separated into "experimental" or "debugging" activities and "production" or "real-time" activities and two programs should be developed, one for each set.

Since experiments have been performed over a long period of time at different places and by different people, it is unavoidable that a multitude of file formats would be present. This complicates processing and results in maintenance difficulties. It would be very beneficial to create a generic file format, to play the role of the standard, and require all newly developed software to adhere to this standard. To support the standard format, a set of programs must be created to convert all old file formats. This entails no difficulty, as the current file processing routine contains all the necessary code.

Suggestions for additional changes to the code

1) In gettest(), most of the code is performing the same task in a slightly different way for each of the instruments. Having a consistent interface between the code and the input files would make for much cleaner, easier to read, and easier to maintain code.

2) Gettest() has three major parts: a) retrieval of the TESTSTUFF structure b) retrieval of (and some processing on) the wavarray[], timarray[], and datarray[][] structures. c) cleaning of timarray, adjusting of wavarray[] and (based on keywords) the ability to perform the cleaning and smoothing operations within gettest. It would be better to make gettest() a smaller more specific function. The TESTSTUFF structure could be obtained by a separate function. The TESTSTUFF structure would then be an input into the gettest function. Gettest would then be limited to obtaining the timarray[], wavarray[] and datarray[][] structures and a minimal amount of processing on these arrays.

3) The number of keywords to the routines should be limited.

Credits

Some information on instruments and file data formats is summarized from the OPAD web page.

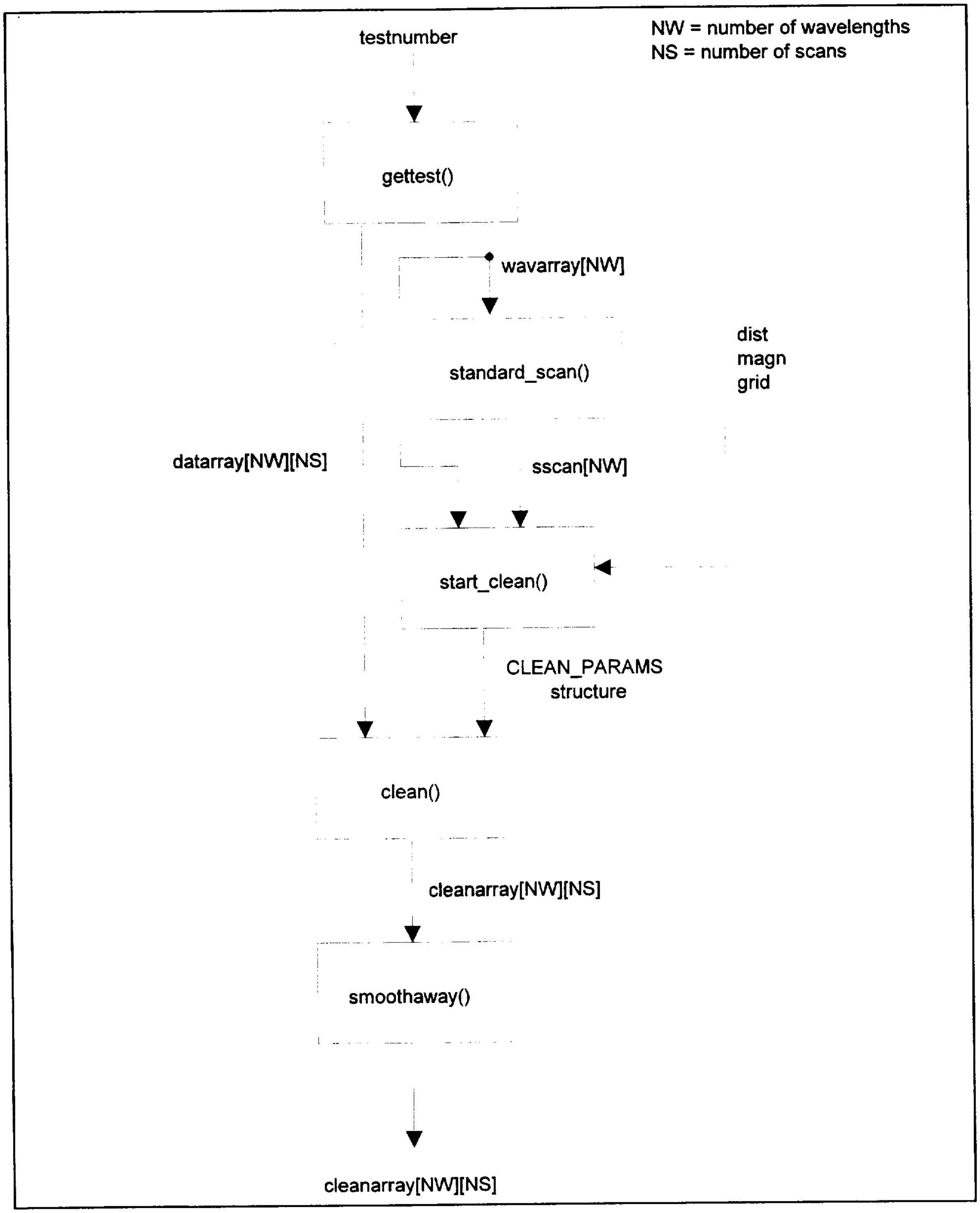

Figure 1: Preprocessing major routines.