

## 1. INTRODUCTION

This document reviews the data reduction process for the SOAR Adaptive-Module Optical Spectrograph (SAMOS) (Roberto et al. 2016). The current version is equipped for reduction of test data from the Goodman High-Throughput Spectrograph. Eventually, it will be used to reduce multi-object spectroscopy data taken with the SAMOS. For information on the instrument, see (Roberto et al. 2016). This pipeline is based (heavily) on the Goodman Spectroscopic Pipeline, and the Flame data reduction pipeline for near-infrared and optical multi-slit spectroscopy. Flame is written in IDL, while the Goodman and SAMOS pipelines are written in Python3. I would like everyone to know that *\*this\** pipeline would not be nearly as good as it is (is it even good? lol idk yet) without the beautifully organized/documentated code that I used as a foundation.

- **SAMOS-working-dir/:**
  - \* **Docs/** you are here :)
  - \* **Readme.md**
  - \* **UNCOMP\_GDMN\_DATA/** test data from Goodman HTS.
  - \* **SAMOS\_DRP/** directory for main pipeline.
  - \* **comp\_refs/** reference spectra for various Goodman comparison lamp setups.
  - \* **slit\_refs/** manually selected pixel positions for slit edges (will be unnecessary with SAMOS data).
  - \* **SAMOSenv.yml** file for creating the environment in which to use SAMOS\_DRP.

## 2. INITIALIZATION

The current version of the pipeline can be run in iPython or JuPyter notebook. A new data reduction session starts with the initialization script ‘**SAMOS\_DRP.SAMOS\_NIGHT**’. This module is responsible for reading the headers from the sample data in **UNCOMP\_GDMN\_DATA/**, and organizing the information into data structures (buckets) for use by the rest of the code.

```

import os
import json
from astropy.io import fits
from astropy import units
import glob
import pandas as pd
import numpy as np
import ccdproc
from ccdproc import CCDData
import warnings
warnings.filterwarnings('ignore')
import logging
import argparse
import logging
from importlib import reload
from SAMOS_DRP.SAMOS_NIGHT import SAMOSNight
from SAMOS_DRP.ImageProcessor import ImageProcessor
from SAMOS_DRP.SAMOSHelpers import load_bucket_status, save_bucket_status
from SAMOS_DRP.SlitBuckets import SlitBuckets
from SAMOS_DRP.Spectroscopy.wavelength import WavelengthCalibration
from SAMOS_DRP.DoWavecal import WaveCalBuckets

```

**Figure 1.** Relevant SDRP packages for reducing SAMOS data.

The initialization script takes five input arguments, the name of the SAMOS observation ID after which the output information will be named (not based on real data yet so feel free to get wild with it), raw data directory path (Goodman test data, for now), path for processed data, and whether or not to ignore bias/flat frames. The Goodman test data has an overscan region which is used for bias correction, so the ‘ignore\_bias’ option should be set to true. The initialization then returns the main data structure of the pipeline. The call for this step should look like:

```

In [1]: from SAMOS_DRP.SAMOS_NIGHT import SAMOSNight
In [2]: SAMOS_reduction = SAMOSNight(obsid='anYtHinG',
                                    raw_data_dir='UNCOMP_GDMN_DATA',
                                    proc_dir='AnyWhErE',
                                    ignore_bias=True,
                                    ignore_flats=False)

```

If you are in a juPyter notebook, the initialization cell is shown in figure 2. This step uses `ccdproc.ImageFileCollection` to read your images and sorts them based on information in the FITS header. The main data buckets are for bias frames, quartz flats, comparison lamps, and science/comparison pairs. Figure 3 shows an example of the output once the data is organized.

```

LOG_FILENAME = 'testing_DRP.log'
if os.path.exists(LOG_FILENAME):
    os.remove(LOG_FILENAME)

raw_data_dir = os.path.join(os.path.abspath('../'), 'UNCOMP_GDMN_DATA')

logging.basicConfig(filename=LOG_FILENAME, level=logging.DEBUG)
GDMN_Night1 = SAMOSNight(obsid='QUARANTINE',
                          raw_data_dir=raw_data_dir,
                          proc_dir=os.path.join(os.getcwd(), 'testing'),
                          ignore_bias=True) #setup
GDMN_Night1() #call

```

**Figure 2.** Initialization step for SAMOS data reduction.

**Bucket Grouping Information**

```

(small buckets inside bigger bucekts!)
BIAS Group:
    Group is Empty
FLATs Group:
Files in BUCKET
    psg_140319_210352_fri.fits
    psg_140319_210620_fri.fits
    psg_140319_210744_fri.fits
    psg_140319_210907_fri.fits
    psg_140319_211031_fri.fits
    psg_140319_211154_fri.fits
    psg_140319_211318_fri.fits
    psg_140319_211442_fri.fits
    psg_140319_211605_fri.fits
    psg_140319_211729_fri.fits
COMP Group:
Files in BUCKET
    psg_140320_005007_cri.fits
    psg_140320_005319_cri.fits
    psg_140320_005443_cri.fits
    psg_140320_005607_cri.fits
    psg_140320_005731_cri.fits
    psg_140320_005854_cri.fits
    psg_140320_010018_cri.fits
    psg_140320_010142_cri.fits
    psg_140320_010305_cri.fits
    psg_140320_010429_cri.fits
    psg_140320_010553_cri.fits

```

**Figure 3.** Initialization step for SAMOS data reduction.

### 3. CCD PROCESSING: OVERSCAN/BIAS

Next, we have to determine the overscan and trim the edges of the fits data. `ccdproc.subtract_overscan` reads a CCD image and performs the overscan correction and trimming. The relevant FITS header keywords are ‘BIASSEC’, which gives the overscan region in the format ‘[y0:y1,x0:x1]’, with ‘y’ as the spatial direction, and ‘x’ as the dispersion direction.

The bias is due to a combination of the camera noise from the readout process and electric “pre-charge” on a CCD chip by the electronics. The BIASSEC header gives the region of overscan, which contains information for this bias. An array of this overscan region is made by grabbing the rows and columns from the data array (`d = f[0].data.astype('f') → overscan = d[biassec_rows, biassec_columns]`). The function then takes the median (or mean) of the overscan regions along the rows (axis=1), and subtracts these values from each data value in their respective data rows. The general procedure is shown in the figure below.

$$\begin{pmatrix} \text{cropped} & \text{and} \\ \text{bias} & \text{subtracted} \\ \text{data} & \text{matrix} \end{pmatrix} = \begin{pmatrix} d_{11} & d_{12} & d_{13} & \dots & d_{1m} \\ d_{21} & d_{22} & \dots & \dots & d_{2m} \\ \dots & \dots & \dots & \dots & \dots \\ d_{n1} & \dots & \dots & \dots & d_{nm} \end{pmatrix} - \begin{pmatrix} [b_1] \\ [b_2] \\ [b_3] \\ [...] \\ [b_n] \end{pmatrix}$$

#### 4. FLAT FIELD CORRECTION

Once we have our cropped and bias subtracted fits files, we need to locate and combine the flat frames into one master flat, which will be divided from the science frames. First we make a stack of the flat field frames and scale them by their median. Then the stack is median combined and normalized, which gives the pixel-to-pixel variation in detector sensitivity. The function outputs a fits file named `LMask(1,2)master.flat`. Finally, the science frames are divided by the master flat and written out to new frames, which are placed in a directory named `flat_fielded`.

To call this routine, use

```
> ./NormDivFlats LMask(1,2).db
```

The function also creates thumbnail images of the output data frames.

This step also cleans cosmic rays by calling `astroscrappy`.

## 5. OUTLINESLITS

Since the headers for the test data do not contain slit information, I typed up a text file of slit positions called `LMask(1,2)_ycoords_c1.txt`. This file contains the top edge pixel for each of the slits in the field image. The module `Slit_id.py` contains the function `get_edges(...)`, which is responsible for tracing the slits, using the master flat output from `NormDivFlats` as a template. `get_edges` takes the master flat and the slit positions text file as input. The function steps along cutouts of the FITS data frame based on the y-axis positions given in the text. For each step, the median of the cutout along the x-axis is taken, and then the derivative along the y-axis is computed to locate the transition from the chip to the slit edge. This steps along the slit until it reaches the end, storing the x and y positions. The arrays of slit positions are written to a region file called `reg_txt.reg` within the corresponding mask directory, and can be loaded into DS9. An example of this for LMask2 is shown in figure 4. The slits outlined here are representative of the slits found in the image of the mask field, so not all of the slits in a science image will be included in this step.

To run this step, use

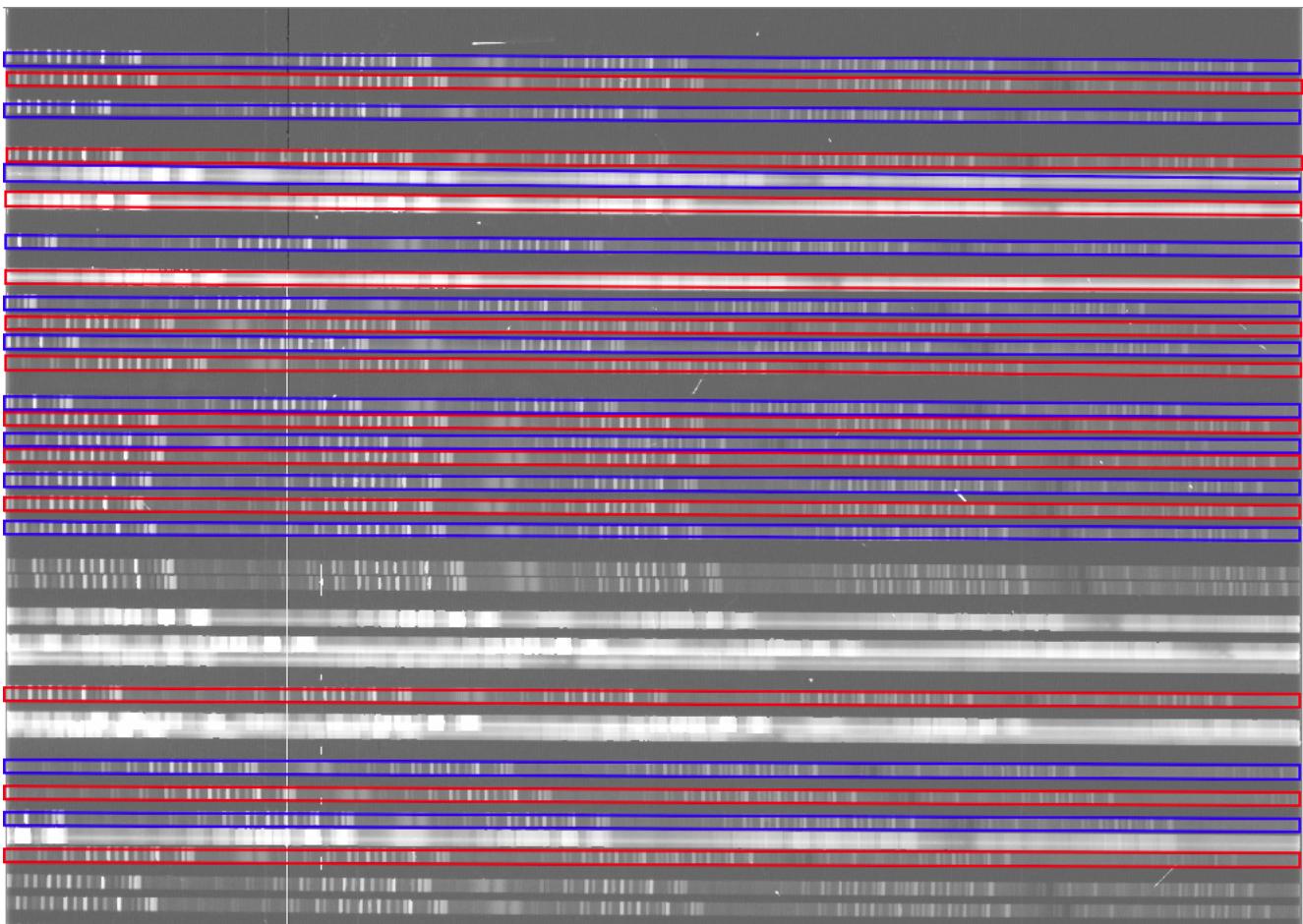
```
> ./OutlineSlits LMask(1,2).db
```

## 6. TL;DR

This is the most up to date version of the documentation for the SAMOS pipeline thus far. Currently, there is no main script to run all the processes at once, so the user will have to run them step by step. To summarize, the user input for data reduction up to and including slit identification is,

```
> ./WhichFITSFiles 2017-11-30 LMask*
> ./OverscanAndTrim LMask*.db
> ./NormDivFlats LMask*.db
> ./OutlineSlits LMask*.db
```

(1)



**Figure 4.** helper\_files/slitsLMask2.reg overlayed onto science image in DS9.

#### REFERENCES

Roberto, M., Donahue, M., Ninkov, Z., et al.  
2016, Ground-based and Airborne  
Instrumentation for Astronomy VI,  
doi:10.1117/12.2233094