mean_rms_overscan

October 28, 2020

[1]:

```
from lsst.daf.persistence import Butler
import numpy as np
import lsst.afw.display as afwDisplay
import matplotlib.pyplot as plt
import lsst.ip.isr as isr
afwDisplay.setDefaultBackend('matplotlib')
```

[2]:

```
def bias_isr(raw, overscan_correct=True):
    #Will return this assembled exposure.
    #This will be an ImageF
    #ImageF you can pull out an array of pixel values.
    raw_clone = raw.clone()
    task = isr.isrTask.IsrTask()
    #isr = "instrument signature removal"
    #ISR: Runs process to get rid of strange effects from sensor and camera.
    task.config.doAssembleCcd = True
    #Assembling CCD from amplifiers (data if given in terms of just amplifiers)
    #HDU Header dictionary...something (gives infor about each of the␣
 ↪amplifiers)
    task.assembleCcd.config.doTrim = True
    #This takes off overscans
    task.config.overscanFitType = 'MEDIAN_PER_ROW'
    task.config.doOverscan = overscan_correct
    #When you do not specify overscan correct: defaults to True. If False: will␣
 ↪not overscan correct.
    task.config.doBias = False
    task.config.doLinearize = False
    task.config.doDark = False
    task.config.doFlat = False
    task.config.doDefect = False
    assembled = task.run(raw_clone).exposure
    return assembled.getImage()
```

[4]:

```
expIds = butler.queryMetadata('raw', 'visit', dataId=dataId)
repo = '/project/shared/BOT/'
repo = '/lsstdata/offline/teststand/BOT/gen2repo/'
butler = Butler(repo)
```

```
run = '6790D'
raftName = 'R01'
detectorName = 'S00'
imageType = 'FLAT'
dataId = dict(detectorName=detectorName, run=run, imageType=imageType,␣
 ↪raftName=raftName)
expId = expIds[0]
dataId['visit'] = expId
exp = butler.get('raw', dataId)
print(dataId, dataId["detectorName"])
```

```
{'detectorName': 'S00', 'run': '6790D', 'imageType': 'FLAT', 'raftName': 'R01',
'visit': 3019101200468} S00
```

[5]:

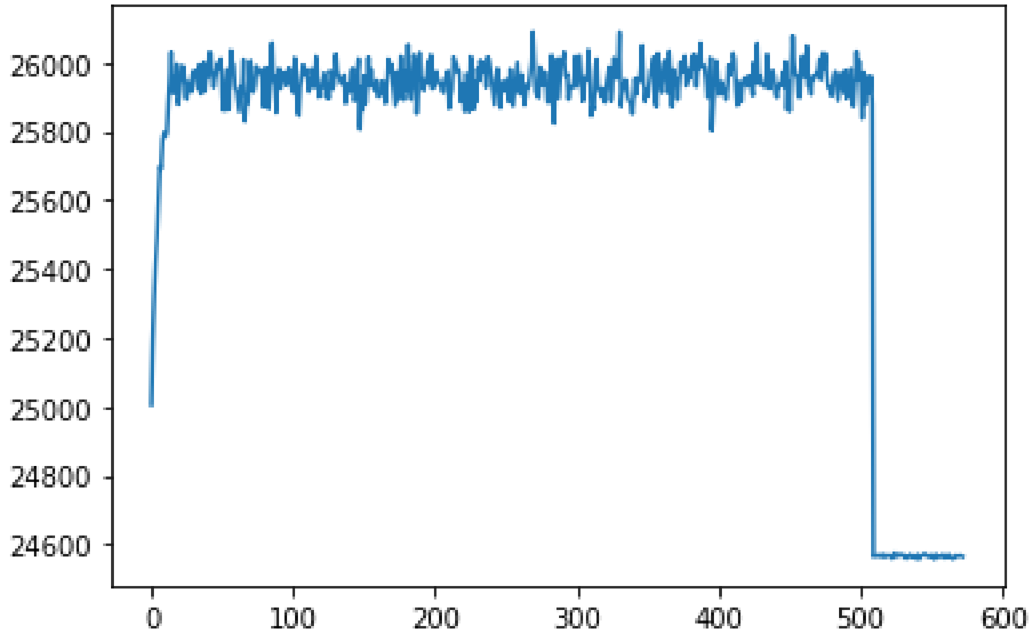
```
ccd = exp.getDetector()
amp = ccd[0] #got an amplifier object from ccd.
dataBBox = amp.getRawDataBBox()
#Here we are getting a Box2I object that defines the physical imaging region.
oscanBBox = amp.getRawHorizontalOverscanBBox()
#Here we are getting another Box2I Object that gives the serial overscan region.
 ↪
#prescanBBox = amp.getRawPrescanBBox()
overscanImage = exp.maskedImage[oscanBBox]
#Now we are getting a MaskedImageF object for the overscan image.
overscanArray = overscanImage.image.array
#Here we are actually doing 2 things: We took an array of the ImageF object␣
 ↪which is returned via the .image method.
ampImage = exp.maskedImage[dataBBox]
#Gives a MaskedImageF object for the physical imaging region of the amplifier.
imArray = ampImage.image.array
#Here we did the exact same thing that we did for the overscan to get an array␣
 ↪of the imaging region.
#print(overscanArray.shape)
#print(imArray.shape)
#Tells us how many pixels are in each array.
#Each rectangle above (not sure which specifically we have below) is 2000 tall␣
 ↪vs. 509 wide.
#print(imArray)
#Each of the values in the array below represent a value for each.
#The value ~ # of electrons at each pixel.
#Cue bucket brigade thinking.
#Units of gain is electrons to Analog Digital Units.
#ADU corresponds to a number of electrons via the gain.
row0 = [imArray[0], overscanArray[0]]
#We put the image array and the overscan array in a list.
row0_flat = [item for sublist in row0 for item in sublist]
#print(row0)
```

```
#print(row0_flat)
#Here we flattened the list (goes through list and makes a new list with all␣
 ↪items).
plt.plot(np.arange(len(row0_flat)), row0_flat)
#returns an array
print(np.mean(overscanArray[0]))
print(np.mean(imArray[0]))
print(np.sqrt(np.mean(imArray[0])))
print()
plt.show
```

```
24565.89
25936.754
161.04892
```

[5]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



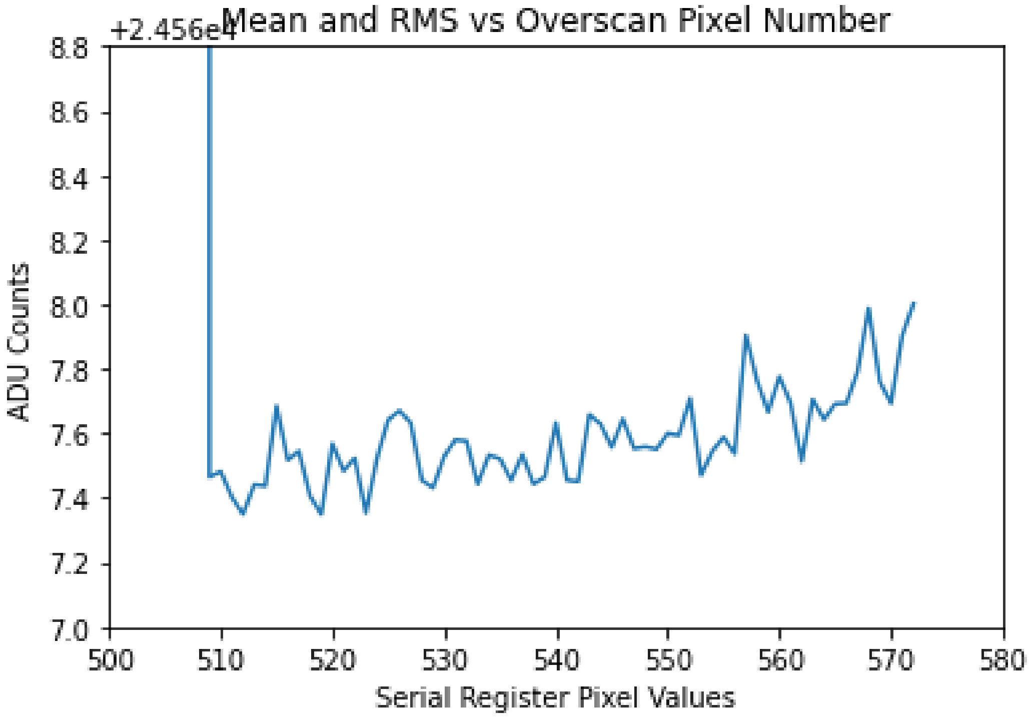
[8]:

```
ccd = exp.getDetector()
amp = ccd[0]
dataBBox = amp.getRawDataBBox()
oscanBBox = amp.getRawHorizontalOverscanBBox()
overscanImage = exp.maskedImage[oscanBBox]
overscanArray = overscanImage.image.array
overscanArraysq = overscanArray**2
ampImage = exp.maskedImage[dataBBox]
```

```
imArray = ampImage.image.array
imArraysq = imArray**2
print(imArraysq.shape)
print(imArray.shape)
#row number then column number.
print(len(imArray[:,0]))
print(len(overscanArray[:,0]))
rms_i = []
rms_o = []
for i in range(len(imArray[0])):
    rms_i.append(np.sqrt(np.mean(imArraysq[:,i])))
for i in range(len(overscanArray[0])):
    rms_o.append(np.sqrt(np.mean(overscanArraysq[:,i])))
rms = [rms_i, rms_o]
rms_flat = [item for sublist in rms for item in sublist]
plt.plot(np.arange(len(rms_flat)), rms_flat)
plt.xlim(500, 580)
plt.ylim(24567, 24568.8)
plt.xlabel("Serial Register Pixel Values")
plt.ylabel("ADU Counts")
plt.title("Mean and RMS vs Overscan Pixel Number")
```

```
(2000, 509)
(2000, 509)
2000
2000
```

```
[8]: Text(0.5, 1.0, 'Mean and RMS vs Overscan Pixel Number')
```


Mean and RMS vs Overscan Pixel Number
+2.456e4
8.8
8.6
8.4
8.2
8.0
7.8
7.6
7.4
7.2
7.0
ADU Counts
500
510
520
530
540
550
560
570
580
Serial Register Pixel Values

[ ]: