Overview

big image is as follows.

SONY TV plants are using cameras and mirrors to take photos for special patterns of panel, then calculate `CUC` (stands for `color uniformity correction`) for each photo.

anyways, i dont know the actual algorithm inside of CUC adjustment prg inside of jig PC. it's a total blackbox to me.

After CUC process is finished, operators check its effect and result. As it is processed by human, and human error always happens. there is a risk that operators omit CUC result, then NG products flow to worldwide market.

To retrieve CUC status from records in the jig PC, there must be a program to read source data and recover CUC status to identify possible NG panels.

I reverse-engineered the original dummy-broken "prg" from OSK, which is written in "Excel.exe", and extracted the core algorithm then implemented in **Python** and rewrote my own prg to do these erands. Because **VBA** is really slow and ugly.

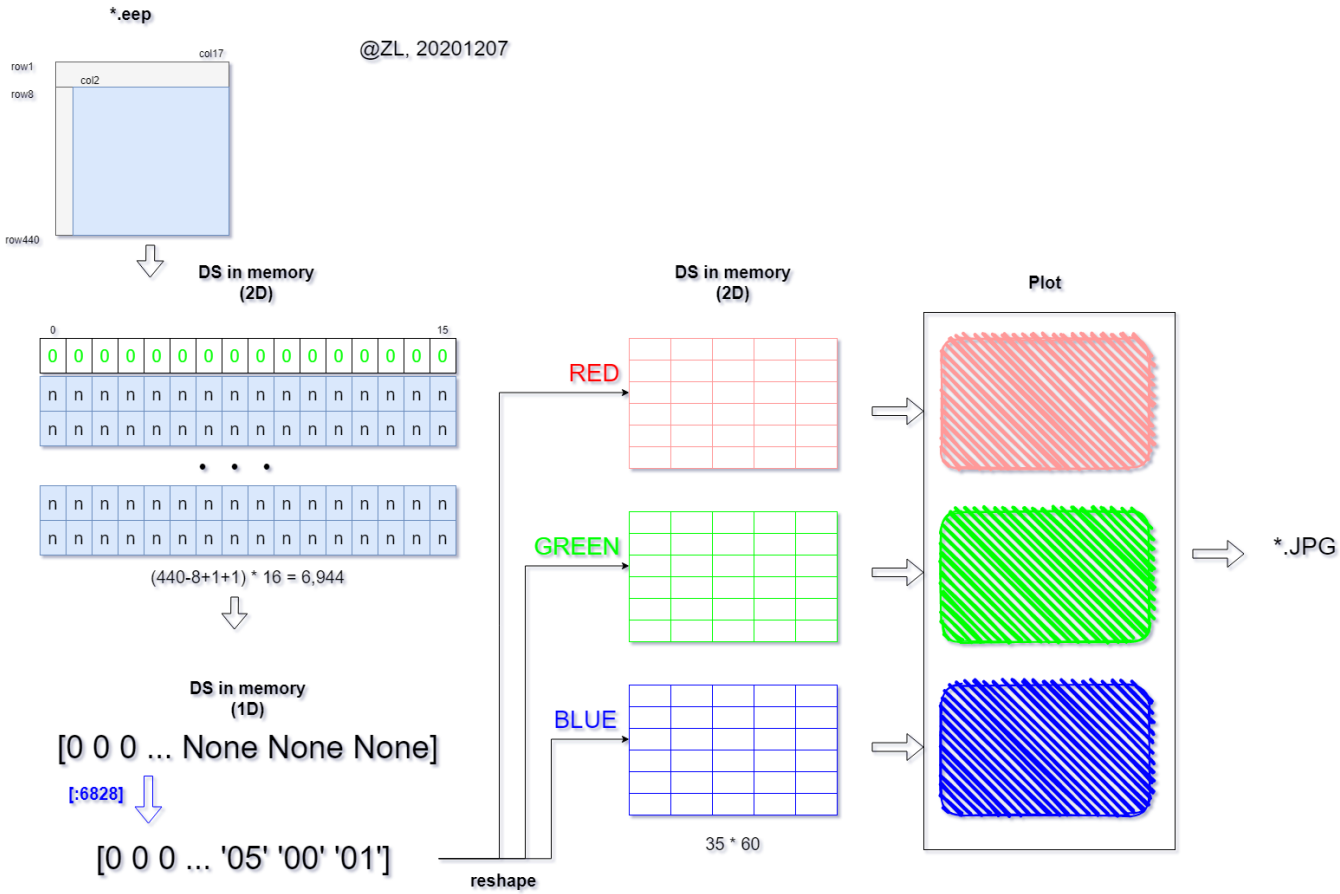
Preparation

confirmation beforehand

1. source file : "*.eep"
2. size of each source file: 25KB
3. number of source files: it depends. sometimes it's 40G+

Reversed algorithm

i had finally completed it after laborious reverse-engineering with hours and hours of blood and tears.



Core algorithms

this section covers most of core algorithms in this program.

Algorithm 01: read source data and convert

```
save2array = np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
## extract src data
df = pd.read_csv(f_path, header=None, skiprows=7, sep=r'\t|\s+',
engine='python').drop([0], axis=1)
save2array = np.vstack((save2array, df.values.tolist()))
b = save2array.flatten()[:6828]
```

Algorithm 02: color conversion

algorithm decides which color conversion is used for each source data.

```
# ssv_pc_office_auto_pkg\myapp_02_muracuc_gui.py
def find_index(self, r, c, color='RED'):
    """
    Purpose:Return calculation result(index) based on color
    author:Z.Liang, 20190505
    """
```

```
    if color == 'RED':
        return int('6019', 16) - int('6000', 16) + c * 36 + r + 1 - (r % 2) * 2
    elif color == 'GREEN':
        return int('6919', 16) - int('6000', 16) + c * 36 + r + 1 - (r % 2) * 2
    elif color == 'BLUE':
        return int('7219', 16) - int('6000', 16) + c * 36 + r + 1 - (r % 2) * 2
```

Algorithm 03: get RED/GREEN/BLUE source data

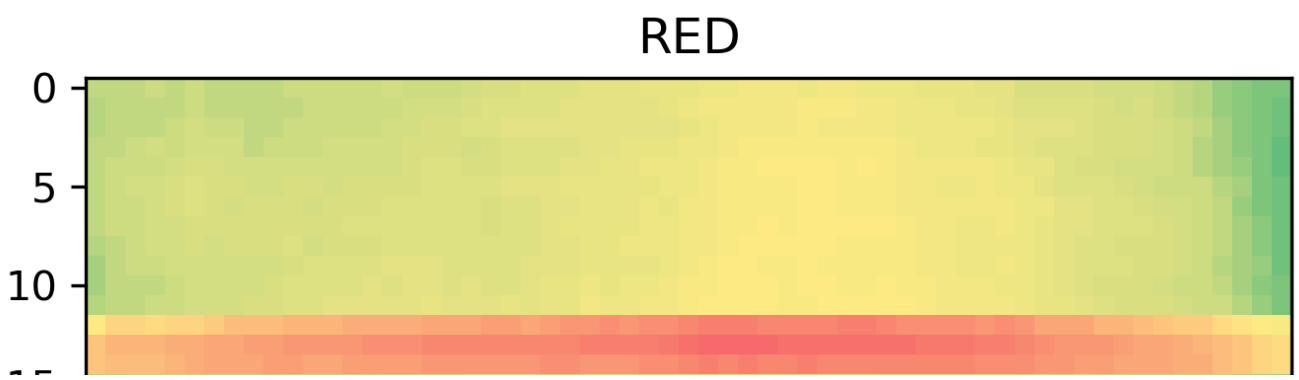
```
## plot data for each color
subplot_datas = []
for color in colors:
    ##  hexdata: the hardest part
    arrZvalue = (np.array([[int(b[self.find_index(r, c, color=color)-1], 16)
                if int(b[self.find_index(r, c, color=color)-1], 16) < 127
                else int(b[self.find_index(r, c, color=color)-1], 16)-256]
                for r in range(0, M) for c in range(0, N)]).flatten().reshape(M,
N))
    subplot_datas.append(arrZvalue)
```

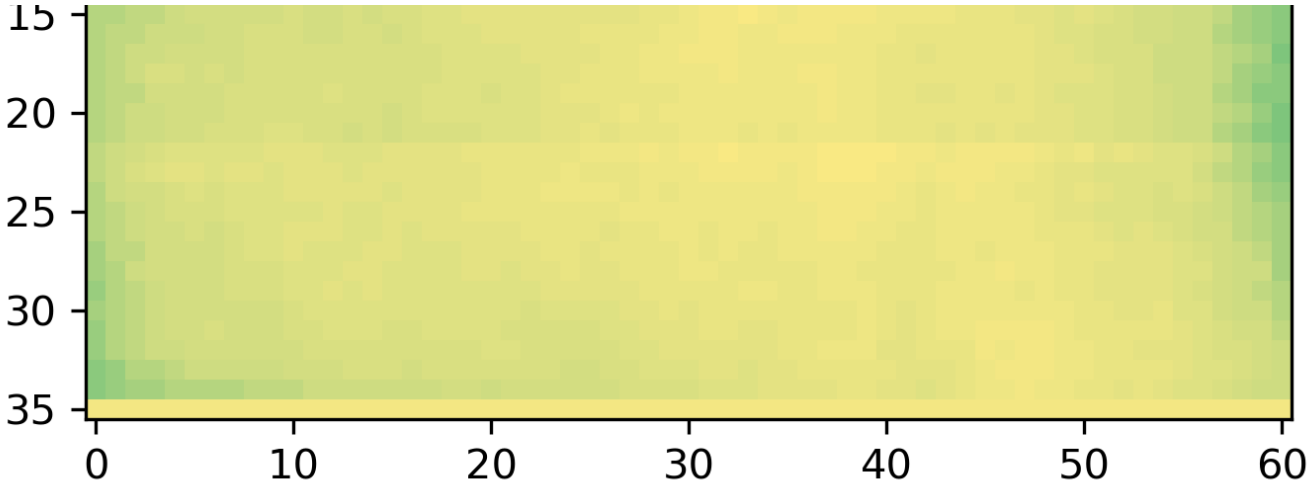
Algorithm 04: plot

```
## plot
subplot_titles = colors
for subplot_title, subplot_data in zip(subplot_titles, subplot_datas):
    if not np.all(subplot_data==0):
        _cmap = ryg
    else:
        _cmap = self.make_colormap([c('#63be7b')])
    plt.subplot(3, 1, i)
    plt.imshow(subplot_data, cmap=_cmap, interpolation='nearest')
    plt.title(f'{subplot_title}', fontsize=12)
    i += 1
```

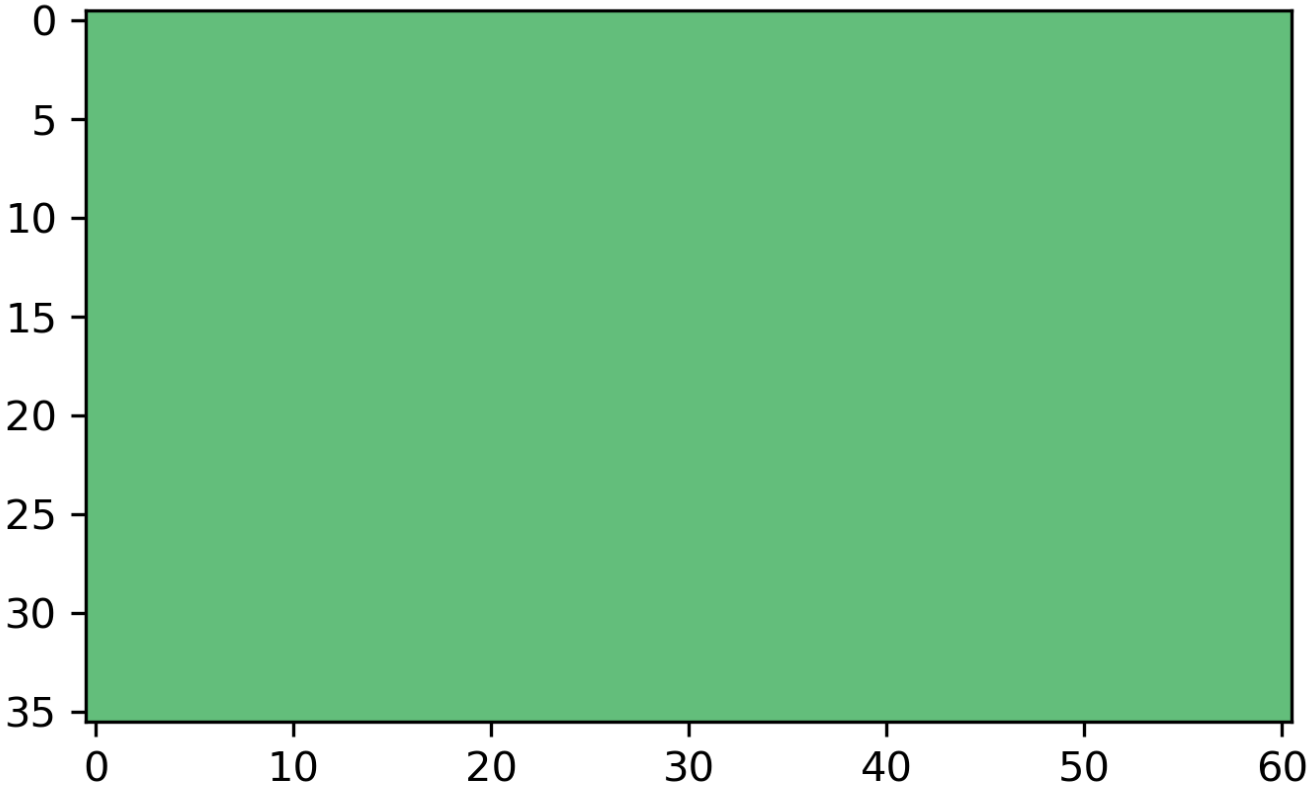
Result

CUC simulation

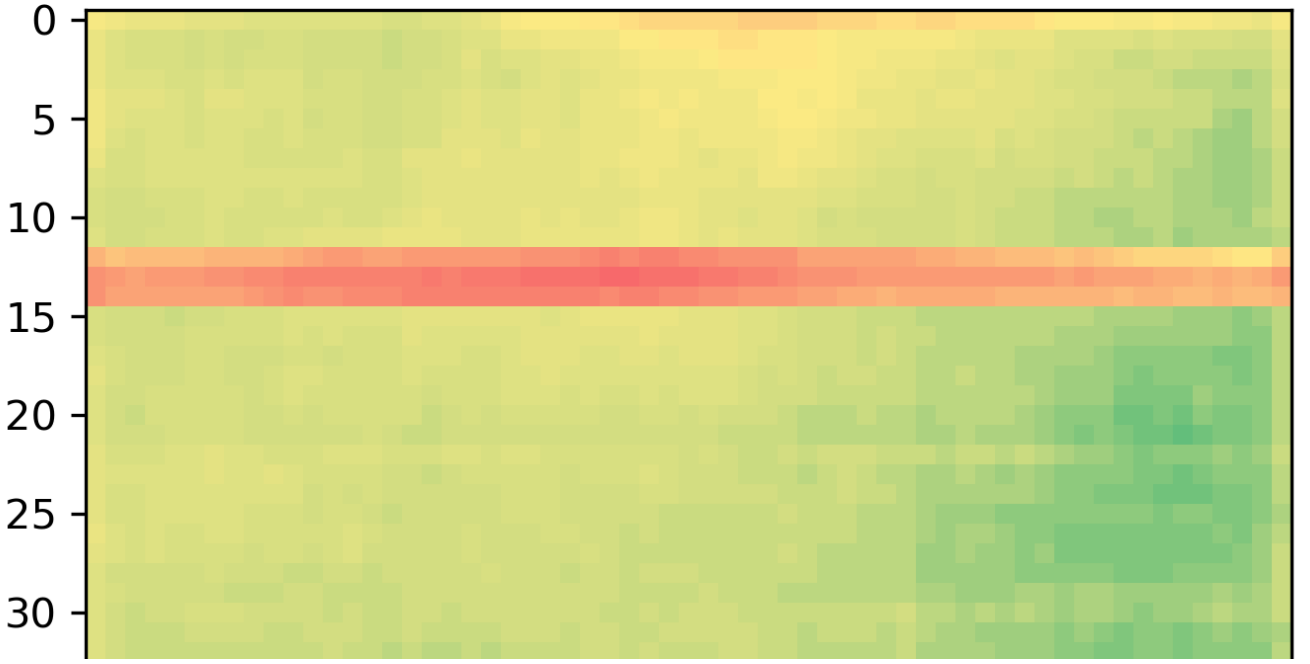


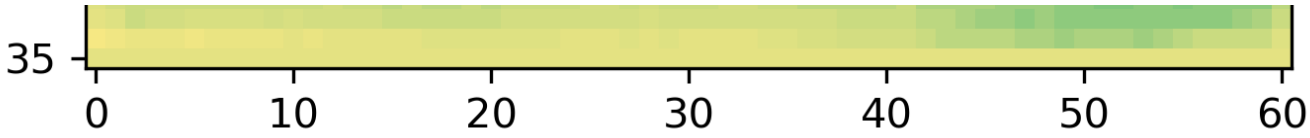

15
20
25
30
35
0
10
20
30
40
50
60

GREEN


0
5
10
15
20
25
30
35
0
10
20
30
40
50
60

BLUE


0
5
10
15
20
25
30



About