MAS507 – Lecture 7

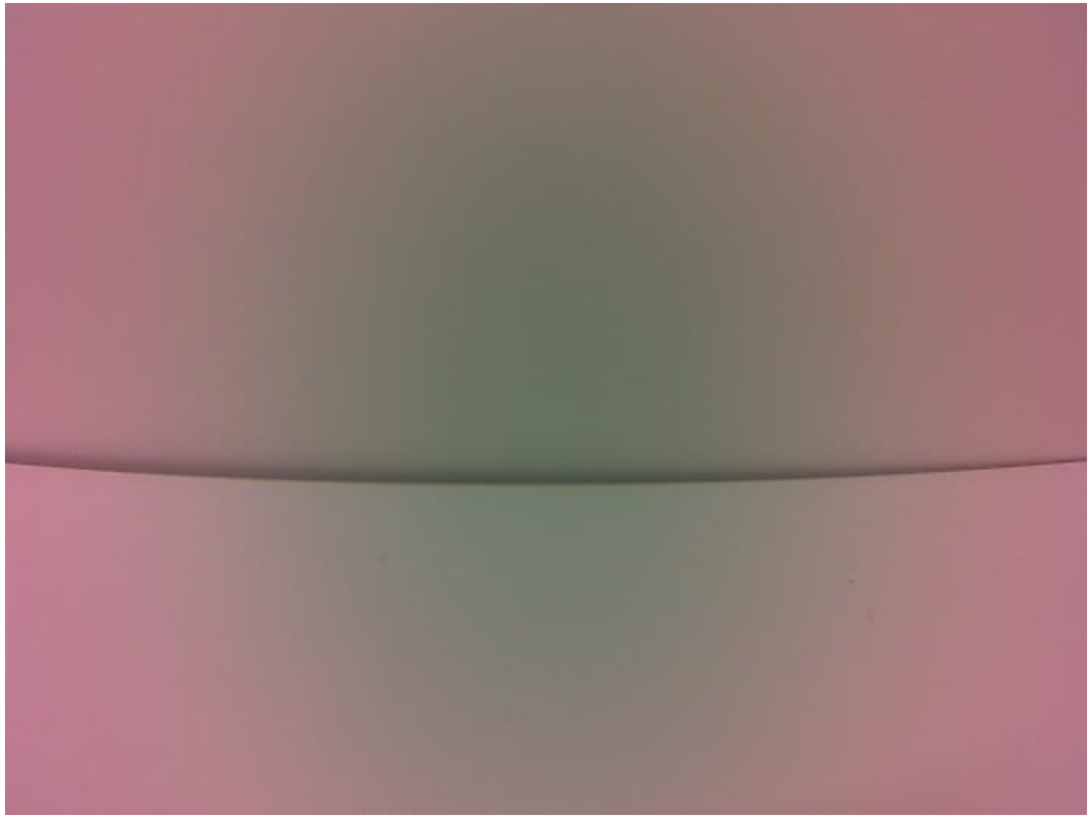
Calibrate color on Jetson camera

UiA

Current image

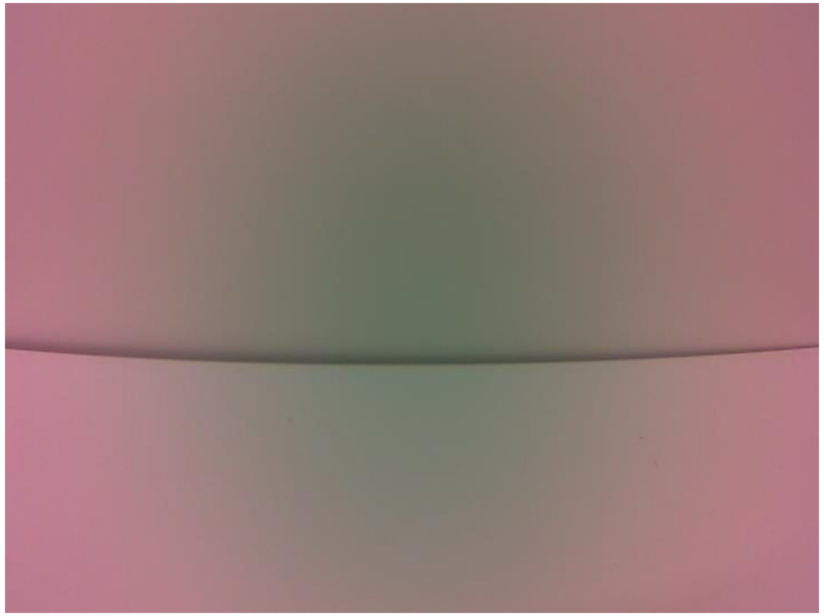
Smartphone

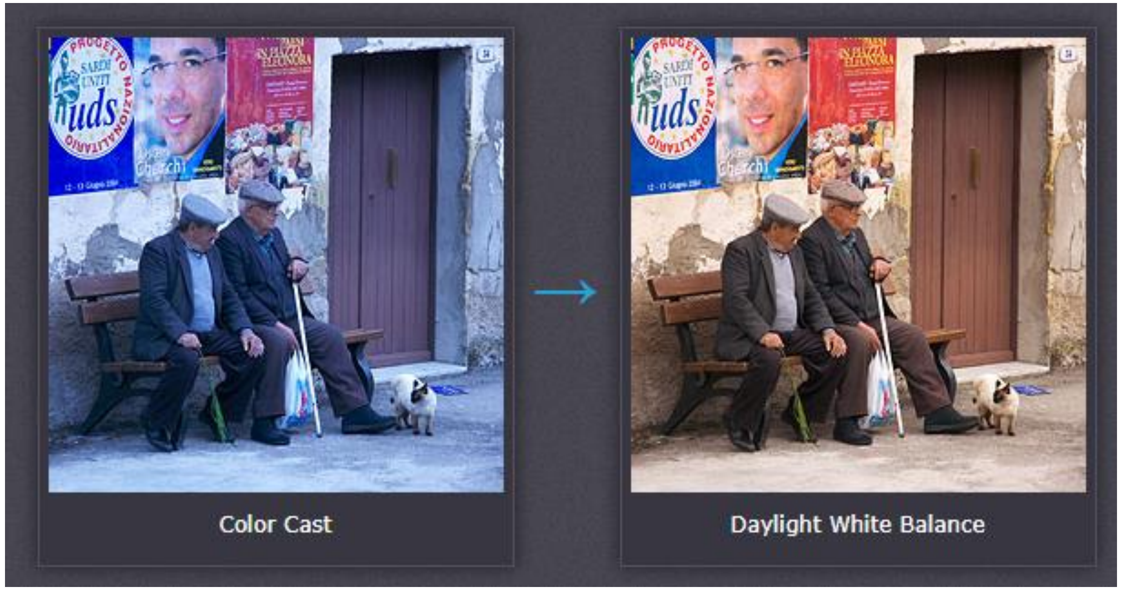
Jetbot



White balance

- Need to calibrate white balance so every pixel is showing the same value when displaying a white image
- Exception from normal white balancing:
  - Every pixel needs a different gain

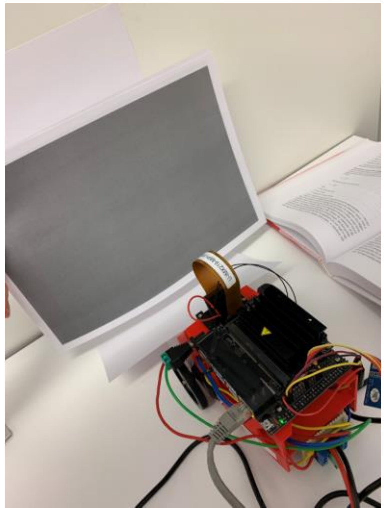




https://www.cambridgeincolour.com/tutorials/white-balance.htm

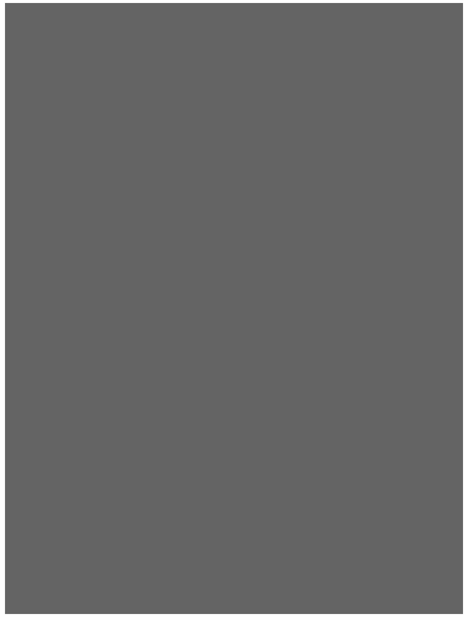
UiA

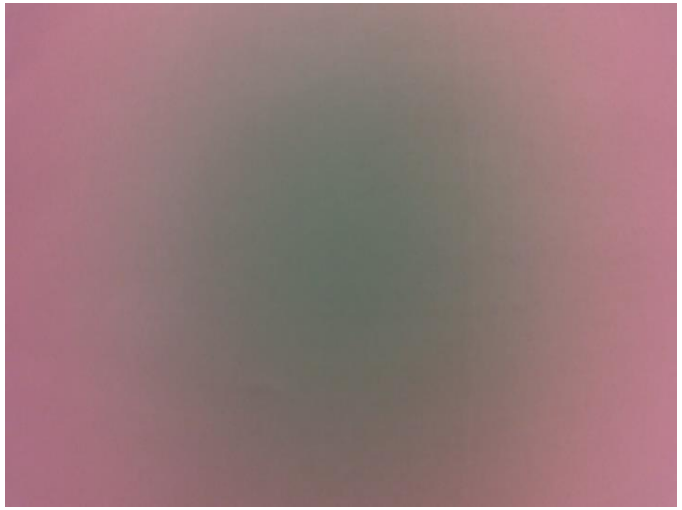
Jetson camera pre-processing



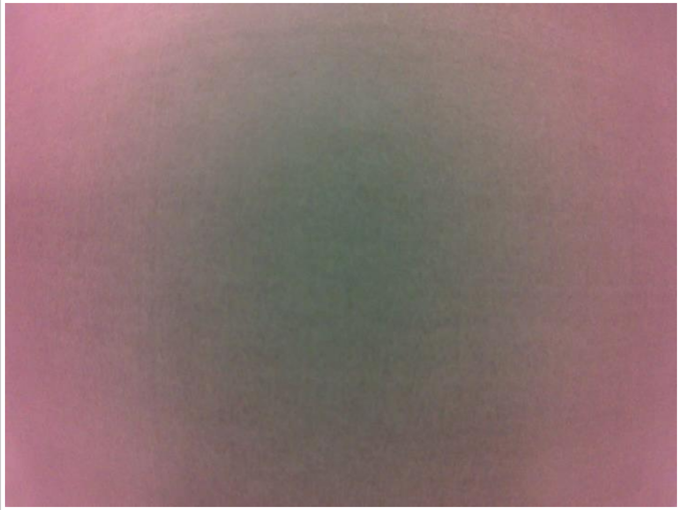
- Internally, the camera adjusts the image gain to get a mean value of brightness to 50% of bandwidth, i.e. 127 on 8 bit color

[100, 100, 100]





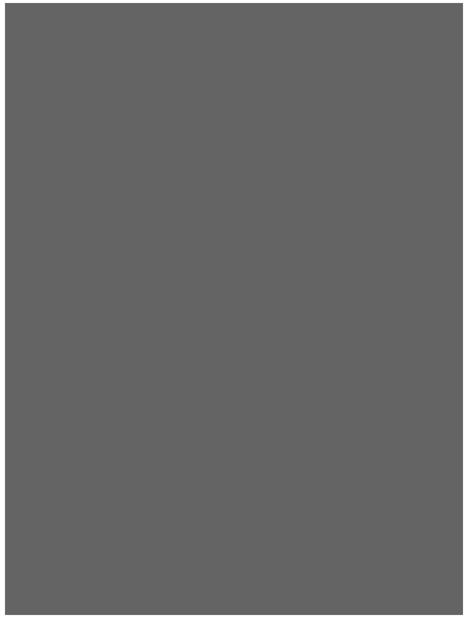
[200, 200, 200]

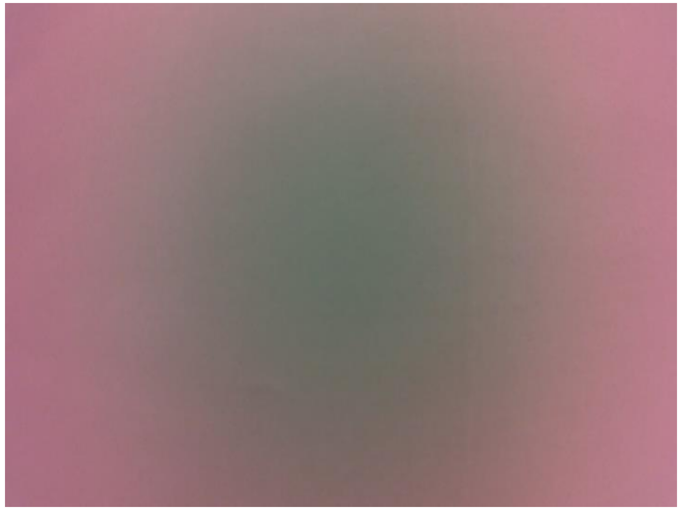


Multiple gray images

- I captured 16 images of gray, and calculated the mean pixel value of each image
- Each image is [540, 720, 3] (BGR)

```
i = 0
for filename in filenames:
    img = cv2.imread(str(filename))
    means[i] = img.mean()
    images[:, :, :, i] = img
    i += 1
```





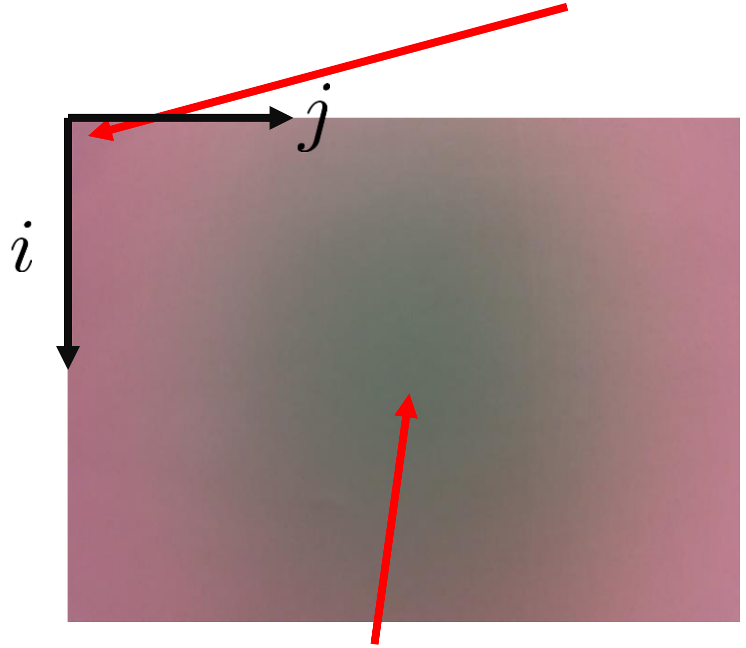
```
In [68]: means
Out[68]:
array([128.11046725, 128.34093364, 128.08906893, 128.39111368,
       125.05483539, 125.27937757, 125.45604767, 125.7525763 ,
       126.78083676, 126.90265947, 127.18718536, 127.3035811 ,
       127.27280436, 127.50649606, 127.5299177 , 127.44413066])
```

Calibration

- To get even color over the image, we need to find **a gain for each color on each pixel** to make a white image to be
  - [127, 127, 127]
  - Blue, Green Red (BGR)
- Make gain matrix such that:

$$\mathrm{img}[i, j, 0] \cdot G_B[i, j] = 127$$
$$\mathrm{img}[i, j, 1] \cdot G_G[i, j] = 127$$
$$\mathrm{img}[i, j, 2] \cdot G_R[i, j] = 127$$

```
In [69]: img[0, 0, :]
Out[69]: array([142, 119, 178], dtype=uint8)
```




```
In [73]: img[270, 360, :]
Out[73]: array([ 98, 110, 100], dtype=uint8)
```

Calibration

- Simple solution: Take 1 image of a white paper, and calculate the gains:

$$\begin{aligned} \text{img}[i,j,0]\cdot G_B[i,j] &= 127 \\ \text{img}[i,j,1]\cdot G_G[i,j] &= 127 \\ \text{img}[i,j,2]\cdot G_R[i,j] &= 127 \end{aligned} \longrightarrow \begin{aligned} G_B[i,j] &= 127/\text{img}[i,j,0] \\ G_G[i,j] &= 127/\text{img}[i,j,1] \\ G_R[i,j] &= 127/\text{img}[i,j,2] \end{aligned}$$

- However, there is some noise in the image between each capture.
- Better to **average a gain matrix over multiple images**

Average calibration: Example for blue

- Take multiple images (N samples) of white paper first:

$$\begin{bmatrix} \text{img1}[i,j,0] \\ \text{img2}[i,j,0] \\ \vdots \\ \text{imgN}[i,j,0] \end{bmatrix} \cdot G_B[i,j] = \begin{bmatrix} 127 \\ 127 \\ \vdots \\ 127 \end{bmatrix}$$

$$\vec{v}\, G_B[i,j] = \vec{b}$$

- No perfect solution, but can be minimized with **ordinary least squares**

UiA

Sum of least squares

$$\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_N \end{bmatrix}$$

$$\vec{v}^T \cdot \vec{v}$$

$$\begin{bmatrix} v_1 & v_2 & \dots & v_N \end{bmatrix} \quad [\cdot]$$

- Can’t isolate G_B directly, as v is a vector.
- Need to multiply with the transpose to make a scalar

$$\vec{v}\, G_B[i,j] = \vec{b}$$

$$(\vec{v}^T\, \vec{v})\, G_B[i,j] = \vec{v}^T\, \vec{b}$$

$$G_B[i,j] = (\vec{v}^T\, \vec{v})^{-1}\, \vec{v}^T\, \vec{b}$$

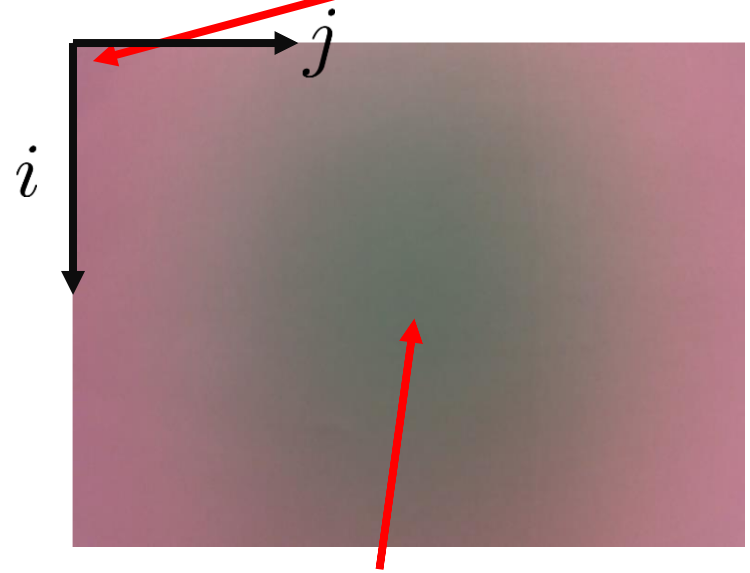
- There is a mathematical proof showing that this minimizes error → se link https://en.wikipedia.org/wiki/Ordinary_least_squares
- Do the same for Green and Red

Gain matrix

- Check gain matrix

```
In [76]: gains[0, 0, :]
Out[76]: array([0.89445495, 1.04327209, 0.71636411])
```

```
In [69]: img[0, 0, :]
Out[69]: array([142, 119, 178], dtype=uint8)
```




```
In [73]: img[270, 360, :]
Out[73]: array([ 98, 110, 100], dtype=uint8)
```

```
In [77]: gains[270, 360, :]
Out[77]: array([1.27477829, 1.1662007 , 1.30426147])
```

Apply gain matrix

- Load gain matrix from calibration (available on Canvas)
  - Hopefully it is similar on all cameras across groups

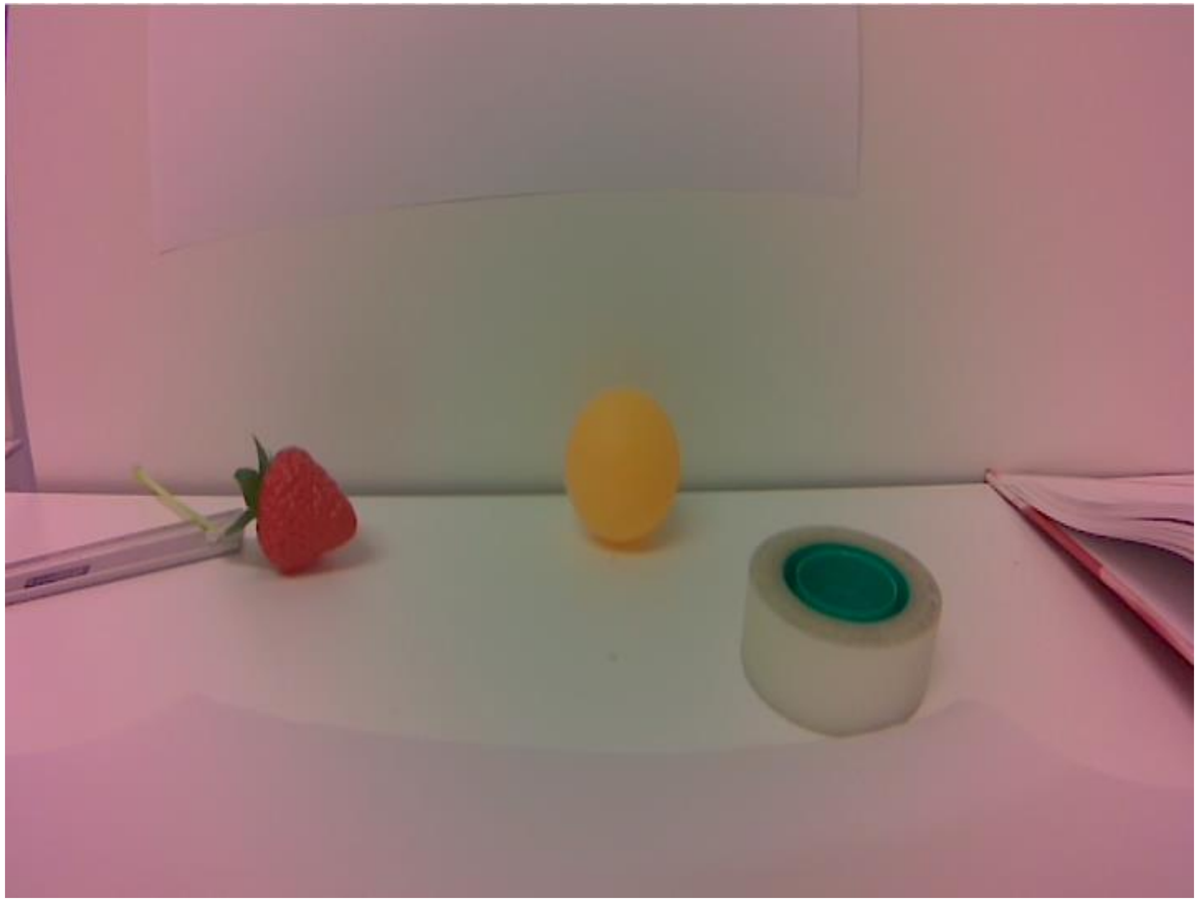
```
# Load color calibration
gainmatrix = np.load('/home/jetbot/colorcalib.npy')
results = np.zeros((540, 720, 3), float)

def calibrateColor(img, gainmatrix, results):
    results[:] = img*gainmatrix
    I = results < 0
    results[I] = 0
    I = results > 255
    results[I] = 255
    img[:] = results
```

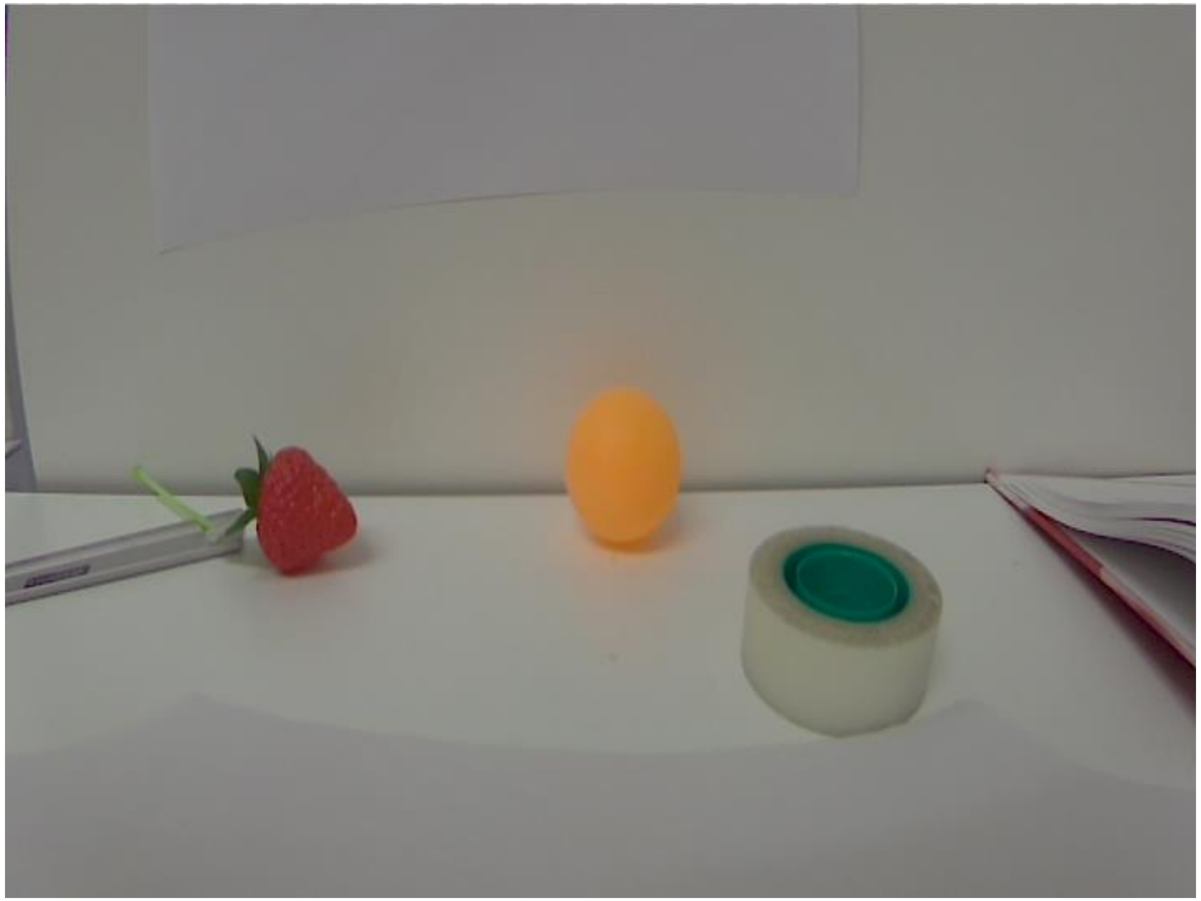
```
ret_val, img = cap.read()
calibrateColor(img, gainmatrix, results)
```

See results

Original



Calibrated



Sum of least squares

- saveImages.py is changed to saveImages_ColorCalibrated.py on Canvas
- The calibration python file is uploaded to Canvas if you want to try, or if the color calibration is bad on your camera
  - I took pictures on the Jetson
  - Transferred images to my own computer using WinSCP
  - Ran calibration on my own computer
  - Saved calibration file to the Jetson using WinSCP