Trabajo práctico 3 - Sensores

Alumnos:

- Carol lugones Ignacio (100073)
- Torresetti Lisandro (99846)

Objetivo

1. Tomar una foto con sus celulares tapando bien el sensor para que no entre nada de luz (se debe obtener una imagen negra, por supuesto....pero no todos los píxeles estarán en cero ya que hay ruido!). Se pide hacer un análisis estadístico del ruido indicando media, desvío y comentando las posibles fuentes del mismo.
2. Tomar una foto también con el celular del patrón de barras generado según el video y obtener el MTF.

In [1]:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from glob import glob
from scipy import signal
%matplotlib inline
img_fnames = glob('./fotos/IMG_*')
print(img_fnames) #Agarramos todas las imagenes en el folder fotos
```

```
['./fotos/IMG_7442.JPG', './fotos/IMG_20210513_202402_BURST9.jpg', './
fotos/IMG_20210513_202402_BURST2.jpg', './fotos/IMG_20210513_202402_BU
RST6.jpg', './fotos/IMG_20210513_202402_BURST4.jpg', './fotos/IMG_743
5.JPG', './fotos/IMG_20210513_202402_BURST8.jpg', './fotos/IMG_2021051
3_202402_BURST10.jpg', './fotos/IMG_7440.JPG', './fotos/IMG_7436.JPG',
'./fotos/IMG_20210513_202402_BURST1.jpg', './fotos/IMG_20210513_202402
_BURST5.jpg', './fotos/IMG_7441.JPG', './fotos/IMG_7438.JPG', './foto
s/IMG_7444.JPG', './fotos/IMG_7443.JPG', './fotos/IMG_20210513_202402_
BURST7.jpg', './fotos/IMG_20210513_202402_BURST11.jpg', './fotos/IMG_7
437.JPG', './fotos/IMG_7439.JPG', './fotos/IMG_20210513_202402_BURST3.
jpg']
```

1) Análisis estadístico del ruido

Para este punto sacamos 10 fotos con cada celular para poder comparar los sensores. Las fotos fueron tomadas con el sensor de la camara completamente tapado. A continuación se detallan los datos de los celulares

Xiaomi Redmi Note 8

- Ancho: 4000 pixeles
- Alto: 3000 pixeles
- Tiempo de exposición: 1/14 seg
- Valor de apertura: 1.67 EV (f/1.8)
- Tasa de velocidad ISO: 9510
- Modo de medida: Promedio ponderado en el centro

- Longitud focal: 4.7 mm

iPhone 7

- Ancho: 4032 pixeles
- Alto: 3024 pixeles
- Tiempo de exposición: 1/4 seg
- Valor de apertura: 1.70 EV (f/1.8)
- Tasa de velocidad ISO: 1600
- Modo de medida: Patron
- Longitud focal: 4.0 mm

In [2]:

```
imgsXiaomi = []
imgsIphone = []
fname = img_fnames[0]
for name in img_fnames:
    img = cv2.imread(name)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    imgsXiaomi.append(img) if 'BURST' in name else imgsIphone.append(img)

print(f"Minima intensidad Xiaomi: {np.min([imgsXiaomi])}")
print(f"Maxima intensidad Xiaomi: {np.max([imgsXiaomi])}\n")

print(f"Minima intensidad Iphone: {np.min([imgsIphone])}")
print(f"Maxima intensidad Iphone: {np.max([imgsIphone])}")
```

```
Minima intensidad Xiaomi: 0
Maxima intensidad Xiaomi: 134

Minima intensidad Iphone: 0
Maxima intensidad Iphone: 47
```

Como se puede ver, en ambos casos la intensidad mínima es cero, pero no sucede lo mismo con la intensidad máxima, lo esperado sería que también sea cero ya que el sensor se encuentra tapado.

In [3]:

```
def statistics(imgStack):
    return  np.mean(imgStack, axis=0),  np.std(imgStack, axis=0)
```

In [4]:

```
stackXiaomi = np.stack(imgsXiaomi)
stackIphone = np.stack(imgsIphone)

# Media y desvio Iphone
img_mediaIphone, img_stdIphone = statistics(stackIphone)

# Media y desvio Xiaomi
img_mediaXiaomi, img_stdXiaomi = statistics(stackXiaomi)
```

In [5]:

```
def dibujar_contorno(mat, title = ''):
    fig = plt.figure()
    X, Y = np.meshgrid(range(len(mat[0])), range(len(mat)))
    Z = mat

    # decimación para no matar la compu calculando contornos!
    dec = 16

    fig = plt.figure(figsize=(16,12))
    cp = plt.contourf(X[::dec], Y[::dec], Z[::dec])
    fig.colorbar(cp)
    plt.title(title, fontsize=18, fontweight='bold')
    plt.show();
    return cp
```

A continuación vamos a realizar los gráficos de la media y el desvío en los distintos canales para ambos celulares utilizando la función brindada por la cátedra.

In [6]:

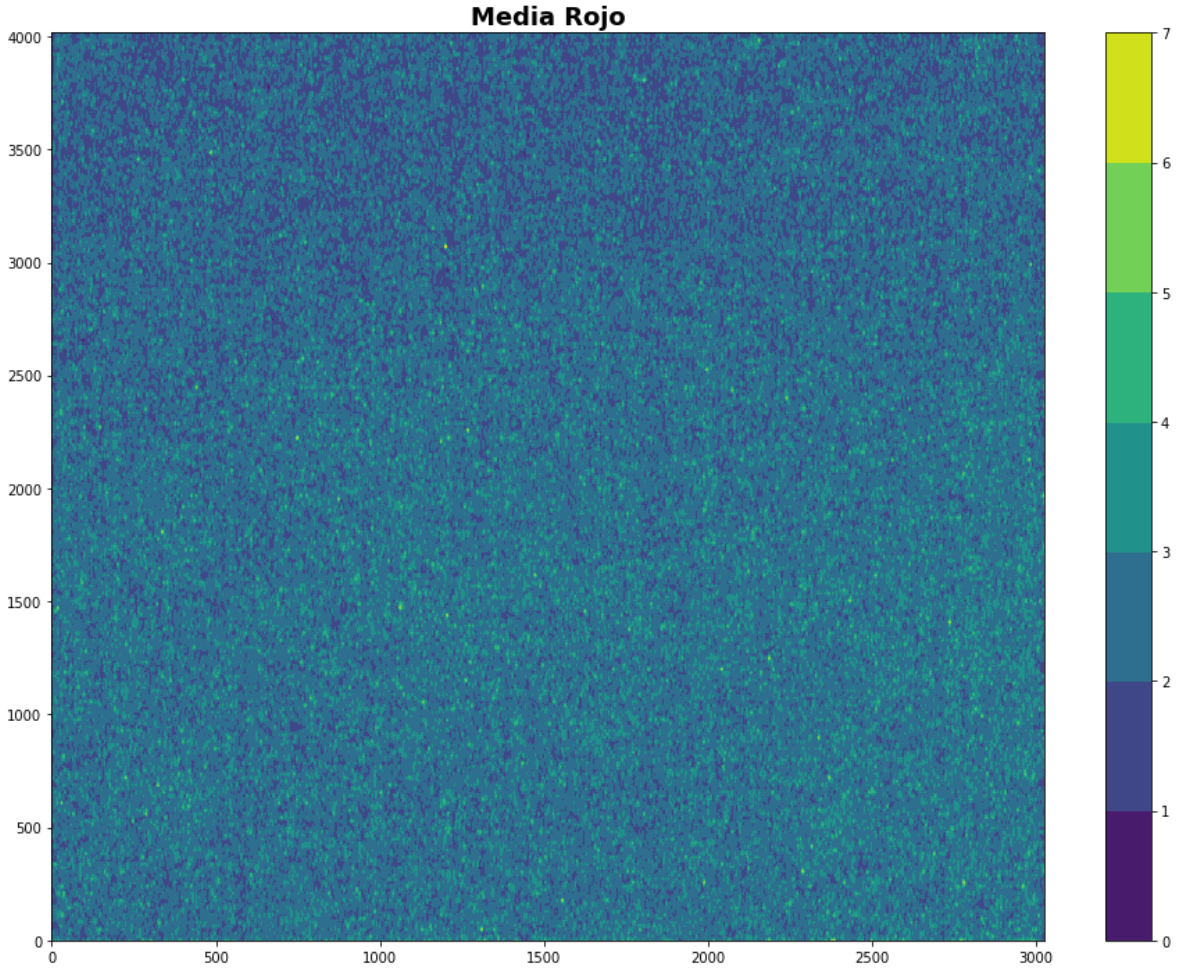
```
CHANNELS = ['Rojo', 'Verde', 'Azul']
def plotMeanAndStdChannels(meanImgs, stdImgs):
    for chNum, ch in enumerate(CHANNELS):
        dibujar_contorno(meanImgs[:,:,chNum], 'Media ' + ch)
        dibujar_contorno(stdImgs[:,:,chNum], 'Std ' + ch)
```

iPhone

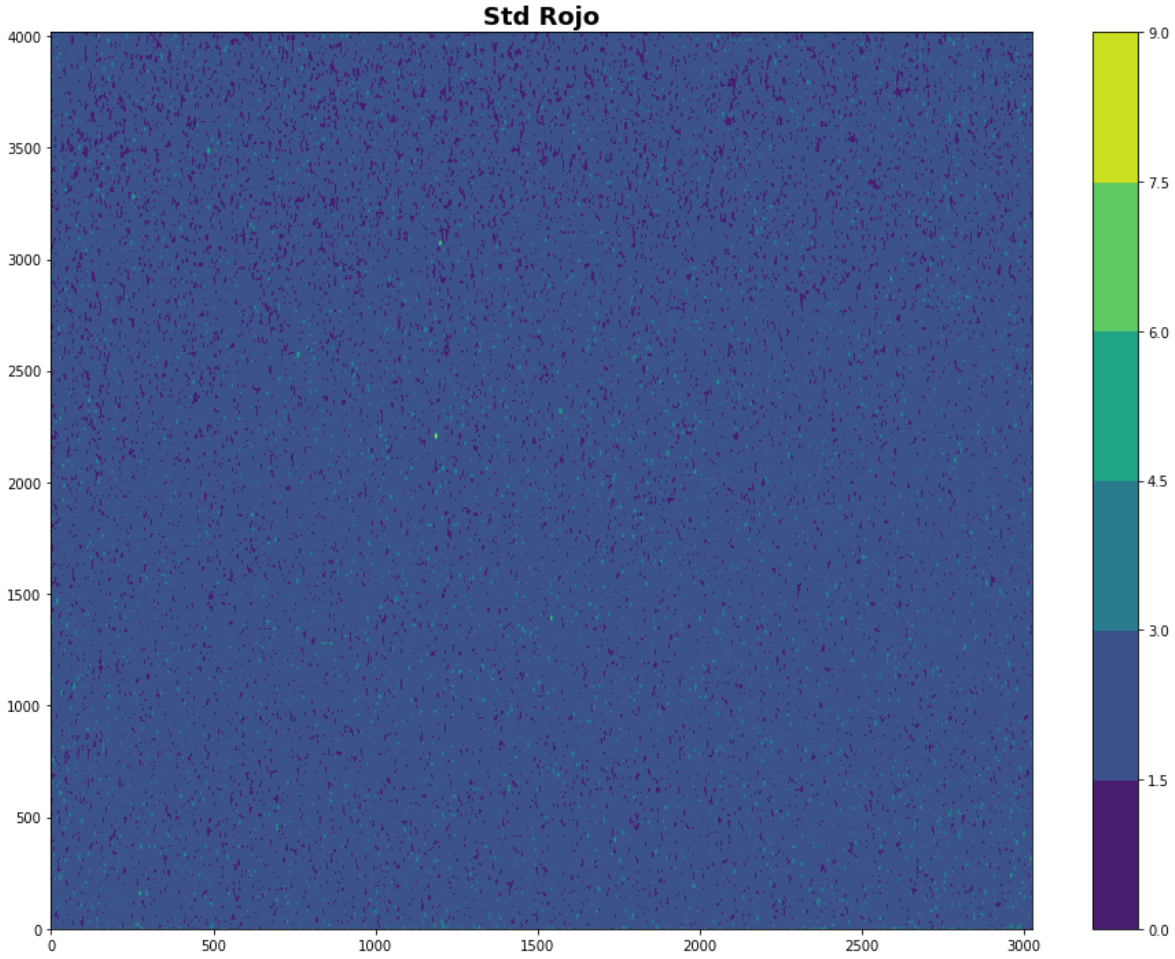
In [7]:

```
plotMeanAndStdChannels(img_mediaIphone, img_stdIphone)
```

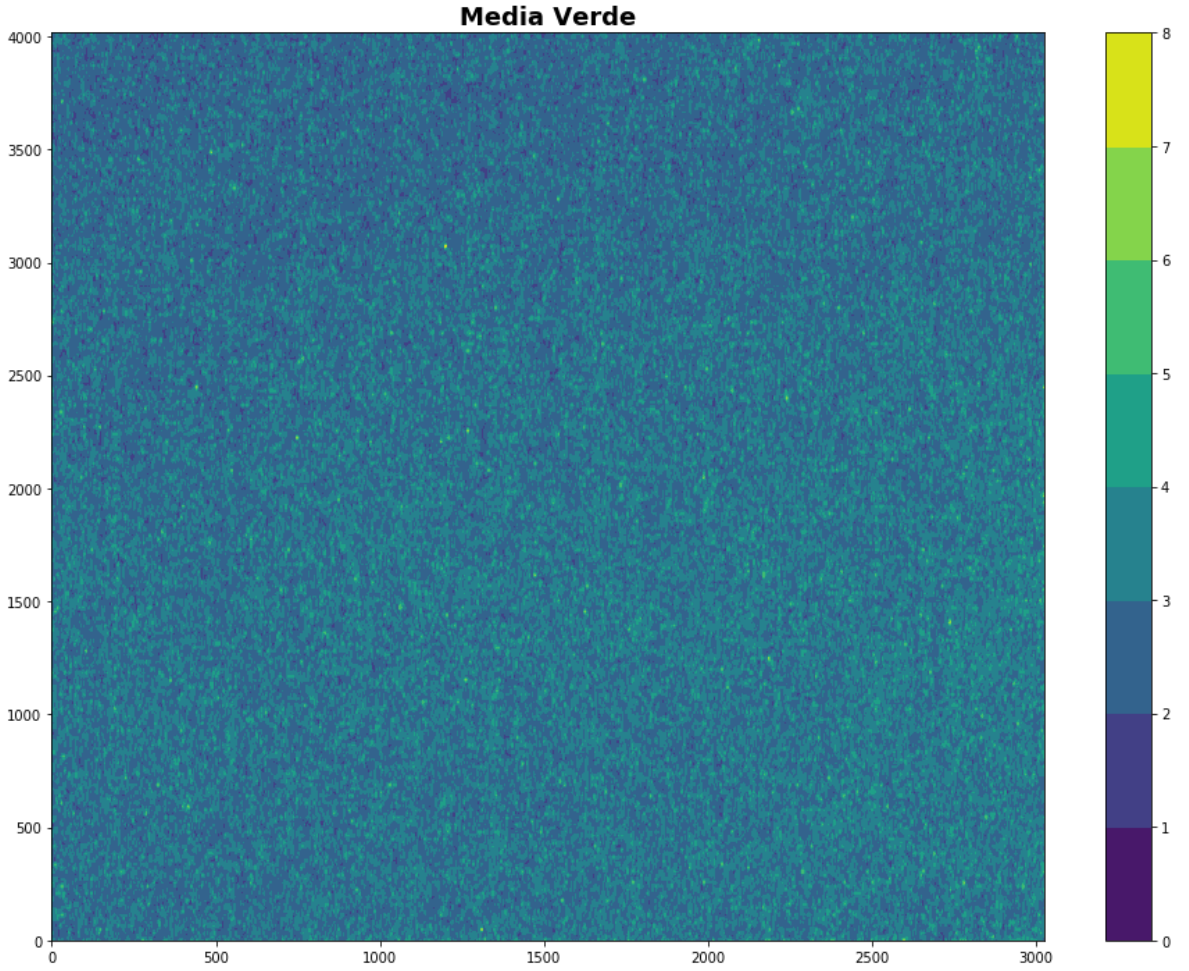
```
<Figure size 432x288 with 0 Axes>
```



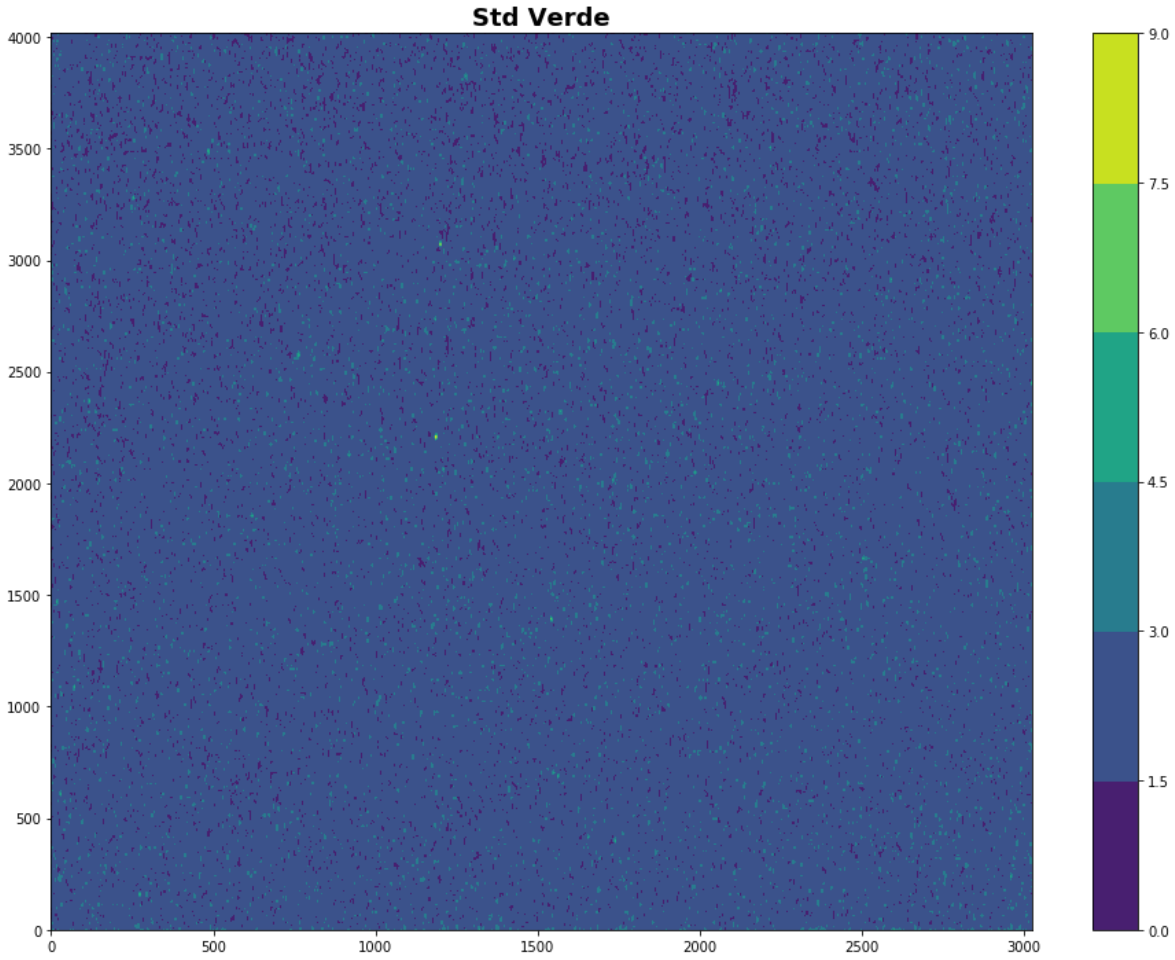
```
<Figure size 432x288 with 0 Axes>
```



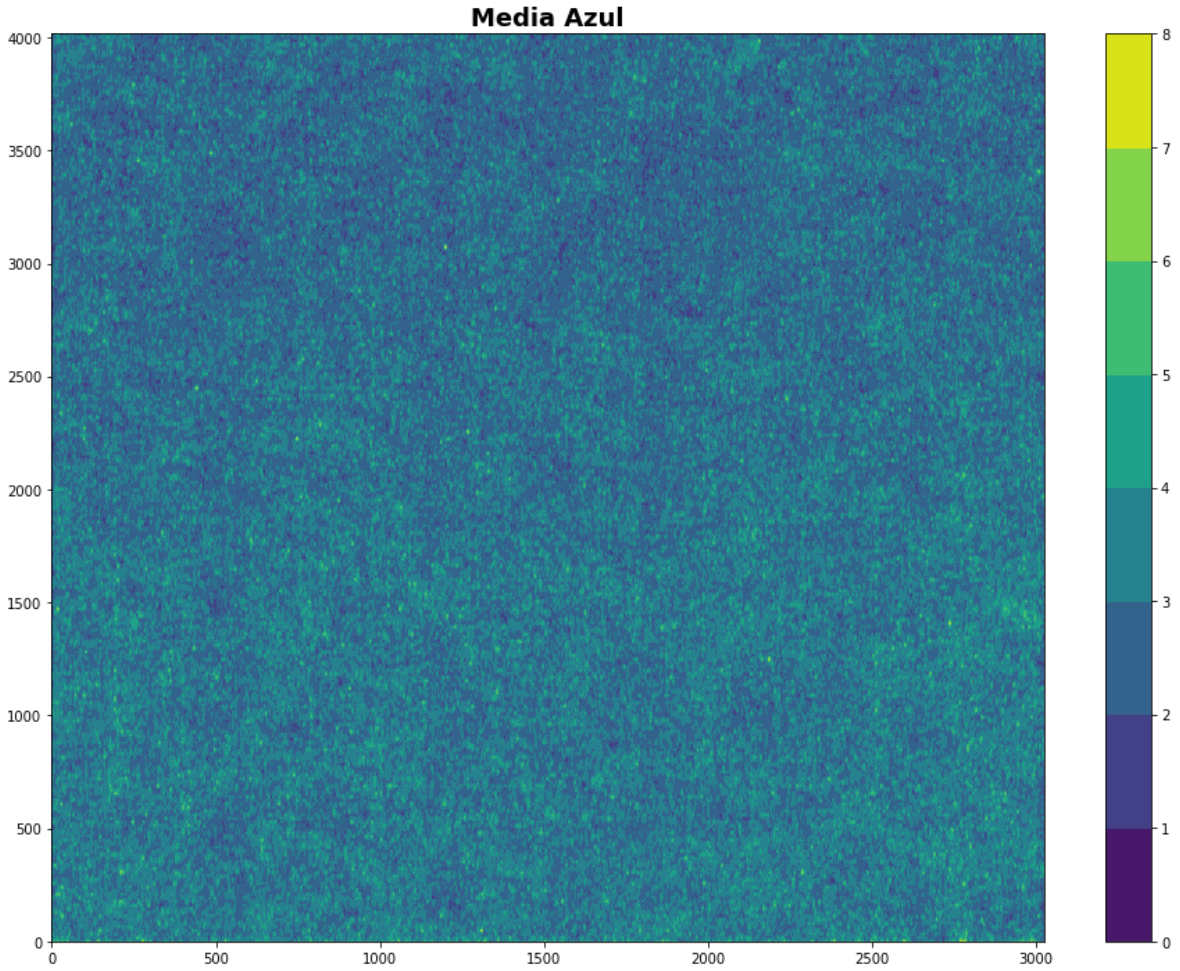
```
<Figure size 432x288 with 0 Axes>
```



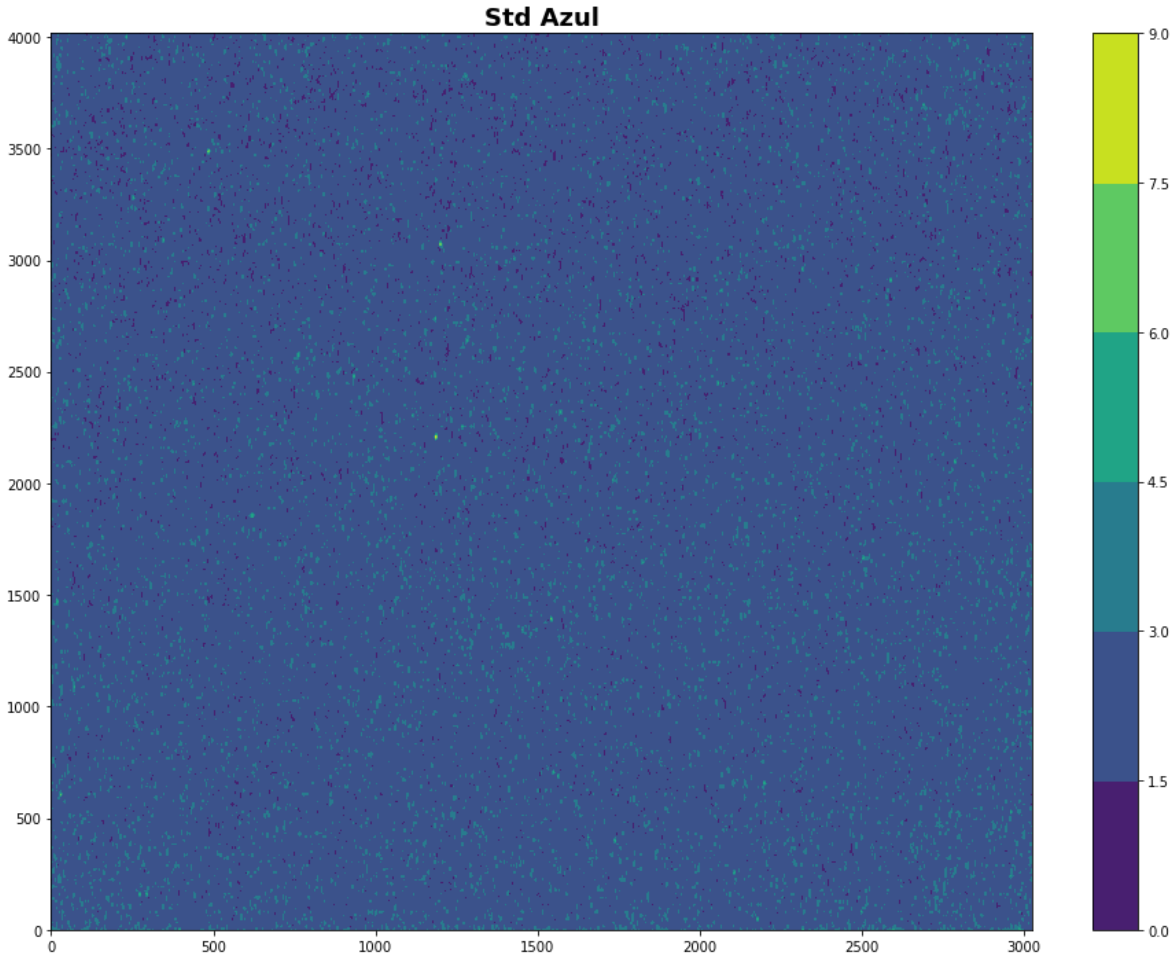

```
<Figure size 432x288 with 0 Axes>
```




```
<Figure size 432x288 with 0 Axes>
```




```
<Figure size 432x288 with 0 Axes>
```



Se puede ver que en los tres canales para el caso de la media el ruido se distribuye de manera uniforme, mientras que el desvio es similar en para los tres canales

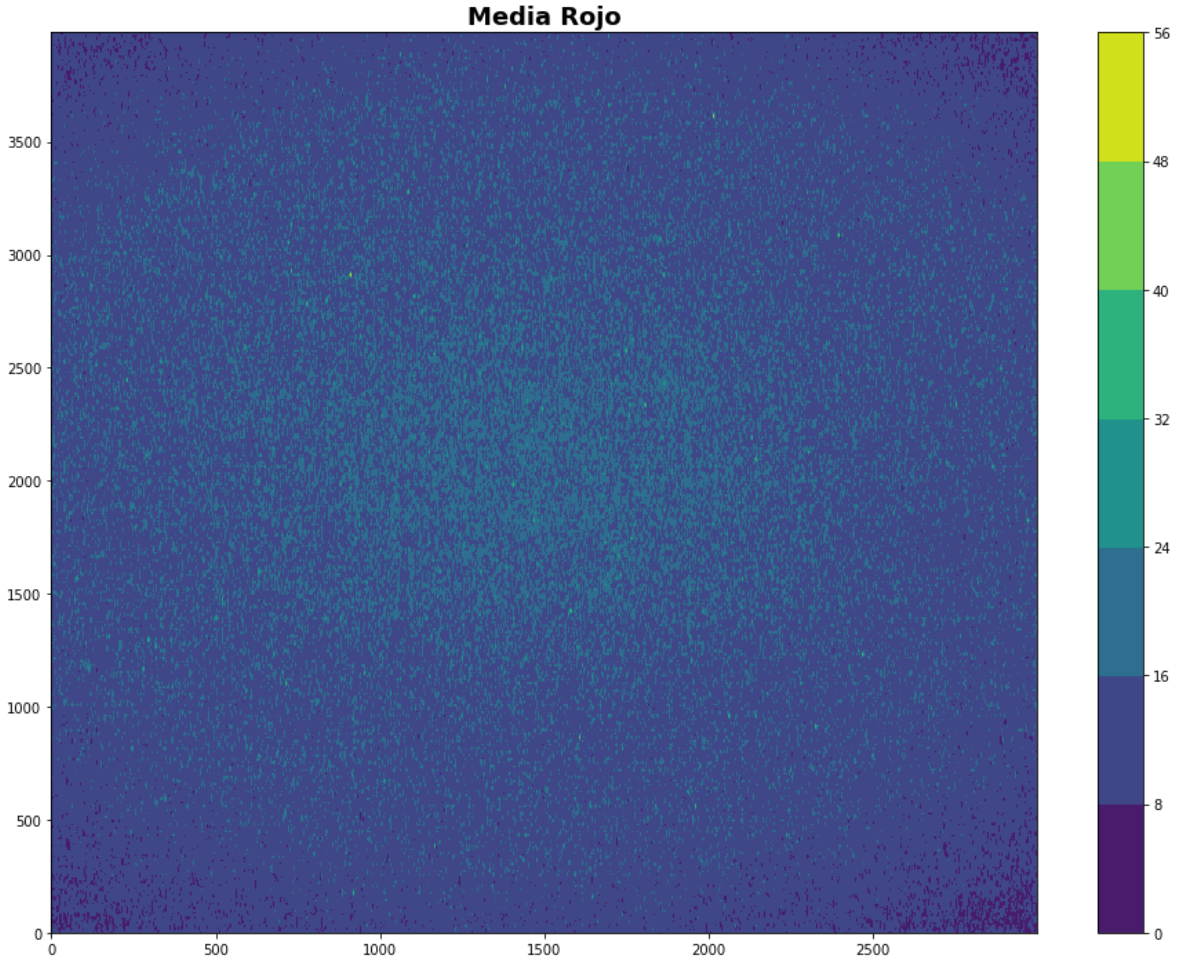
Por que pasa esto? al ser ruido se puedo ver que es uniforme en todo el campo si es geometrico es raro porque se espera que sea uniforme, posibles errores que al tomar la foto entre luz por el borde dado que estaba con la funda protectora , sino efecto de viñetado las cosas mas lejas del eje optico llega menos cantidad de luz ahi

Xiaomi

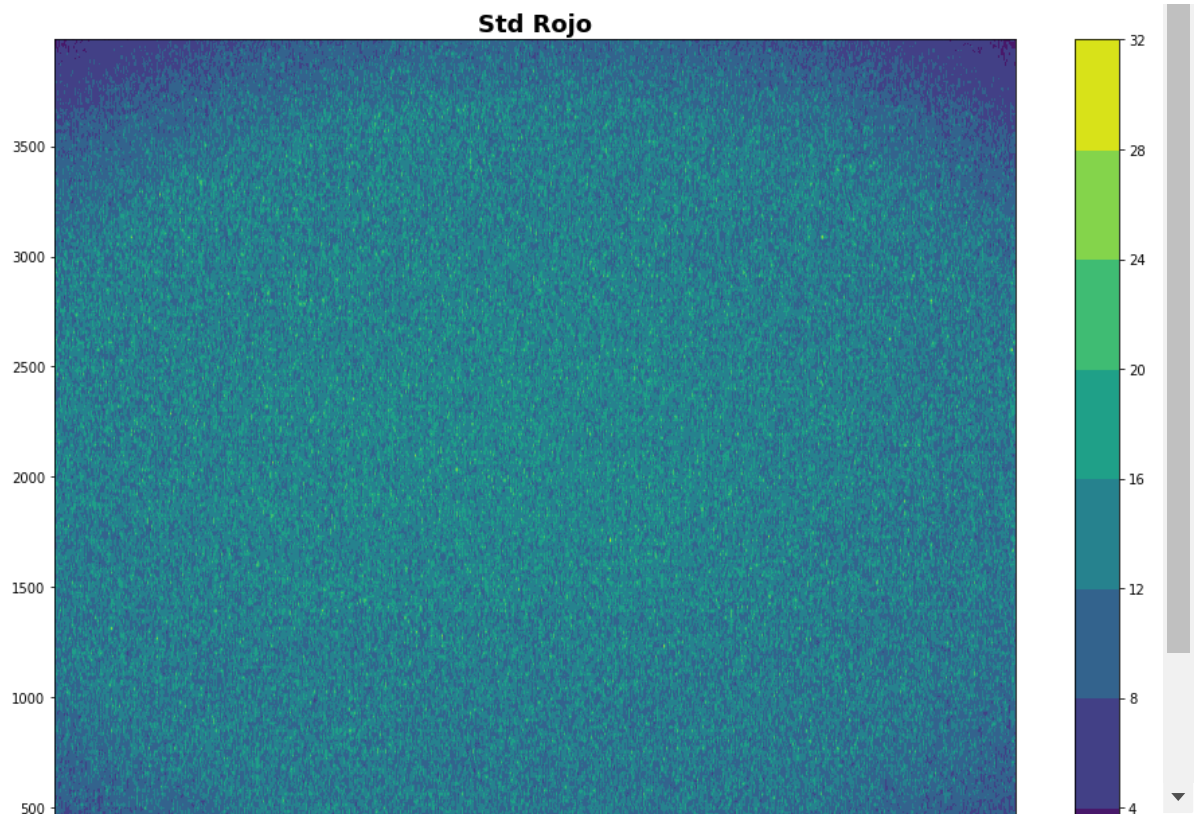
In [8]:

```
plotMeanAndStdChannels(img_mediaXiaomi, img_stdXiaomi)
```

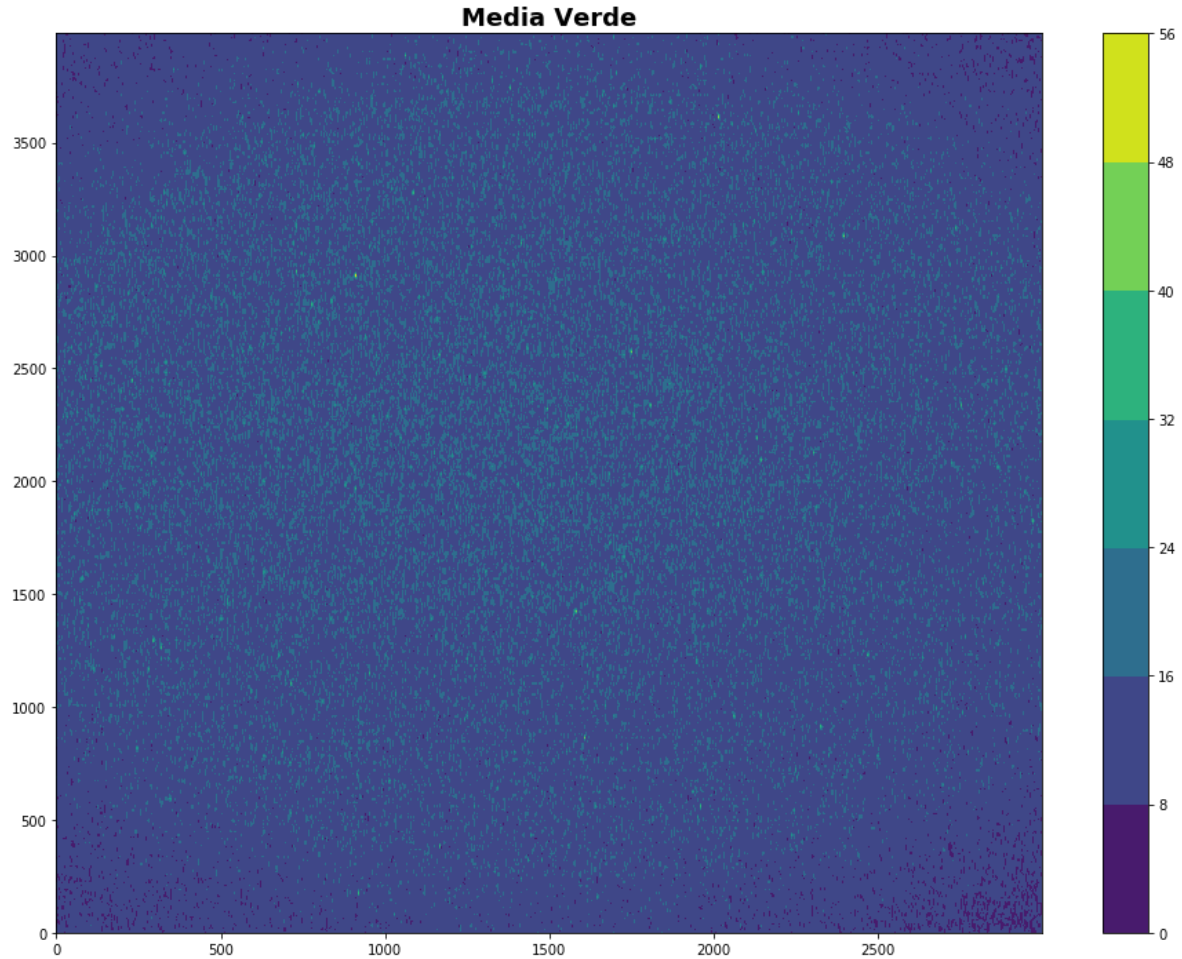
<Figure size 432x288 with 0 Axes>



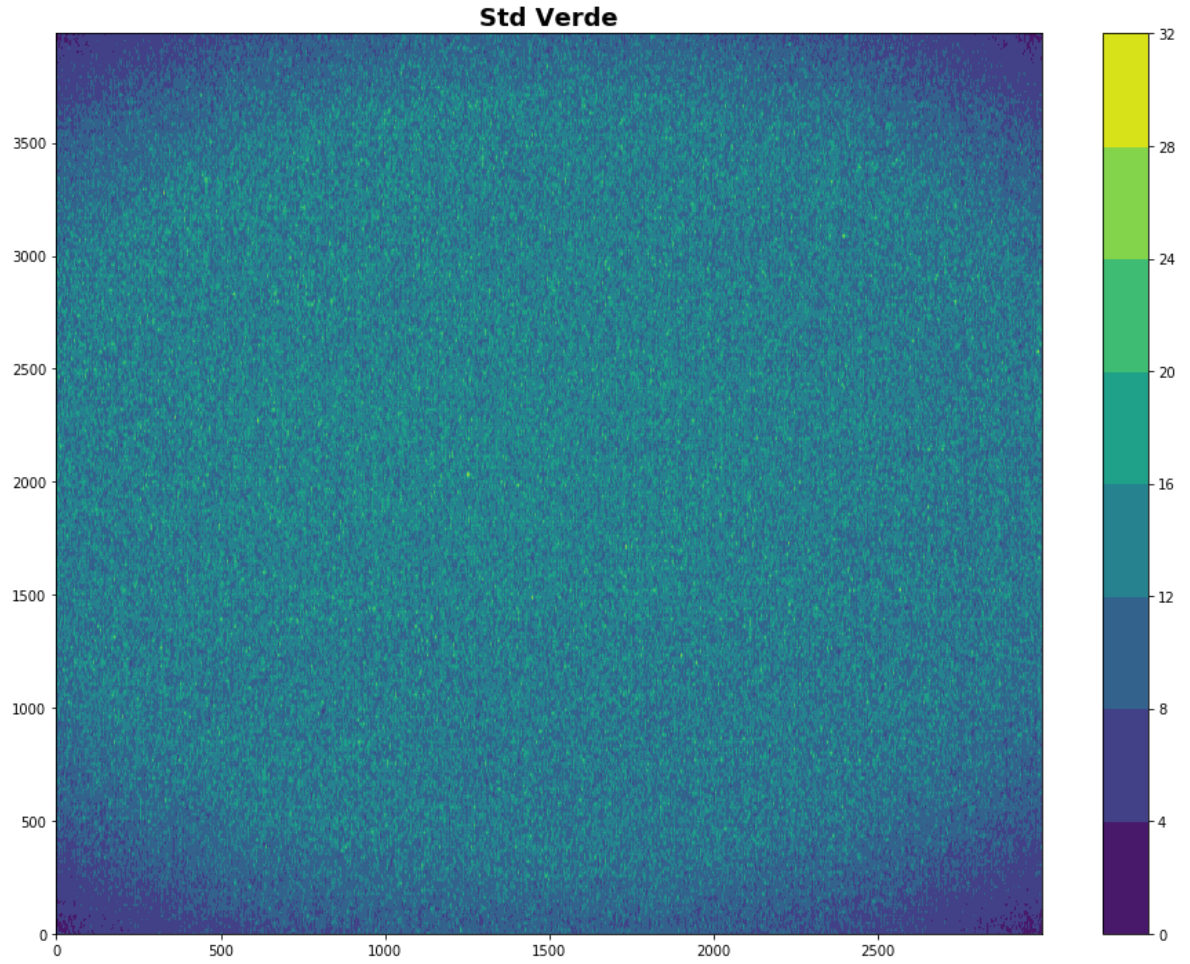
<Figure size 432x288 with 0 Axes>



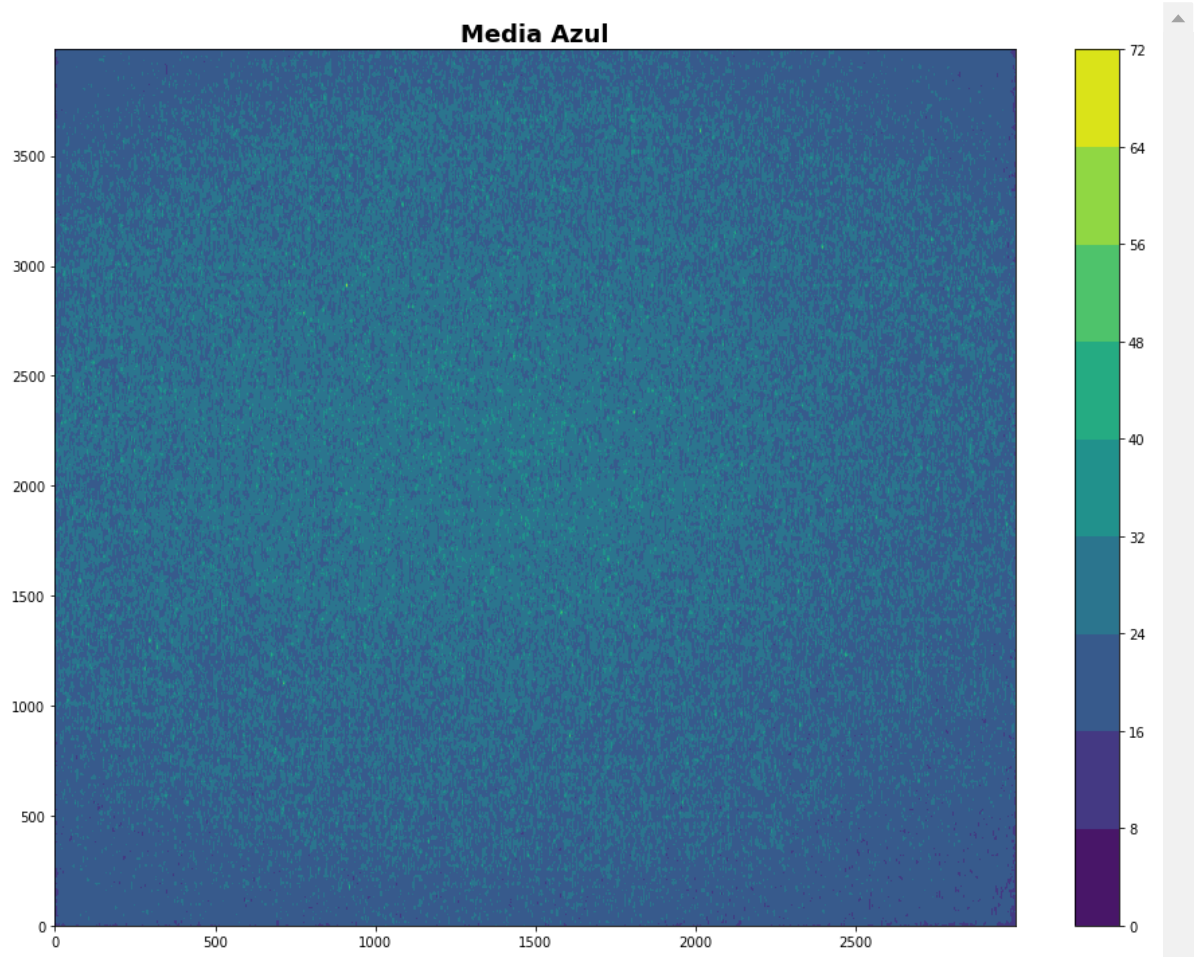

```
<Figure size 432x288 with 0 Axes>
```



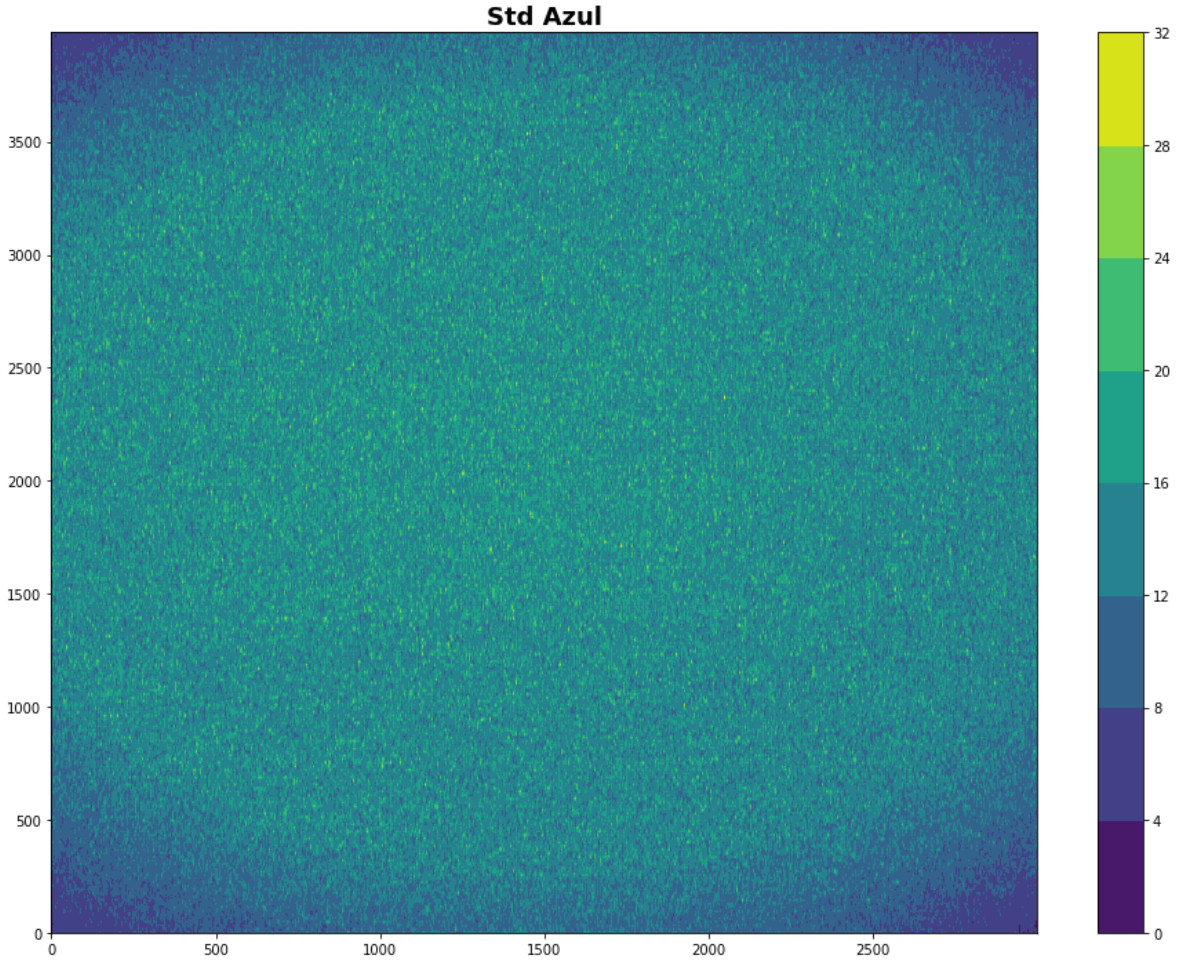

```
<Figure size 432x288 with 0 Axes>
```




```
<Figure size 432x288 with 0 Axes>
```



```
<Figure size 432x288 with 0 Axes>
```



Comparando con las imagenes obtenidas con el iPhone se puede ver que la media para los 3 canales de un celular y de otro no son iguales. Para el caso del Xiaomi las tres medias varian y van variando en valor de canal a canal, siendo el canal rojo el que posee los valores más bajos, luego le sigue el verde y por último el azul. Para el desvío también hay diferencias si comparamos con el iPhone, aquí todas parecen tener un efecto de viñetado en los bordes y además se aprecia más la diferencia entre el desvío estandar en un canal y en otro, cosa que no sucedia con el iPhone.

Estadísticas de ruido: relación entre media y desvío

Para analizar esta relación vamos a realizar histogramas en 2 dimensiones. Como venimos haciendo hasta el momento, lo realizaremos tanto para el iPhone como para el Xiaomi.

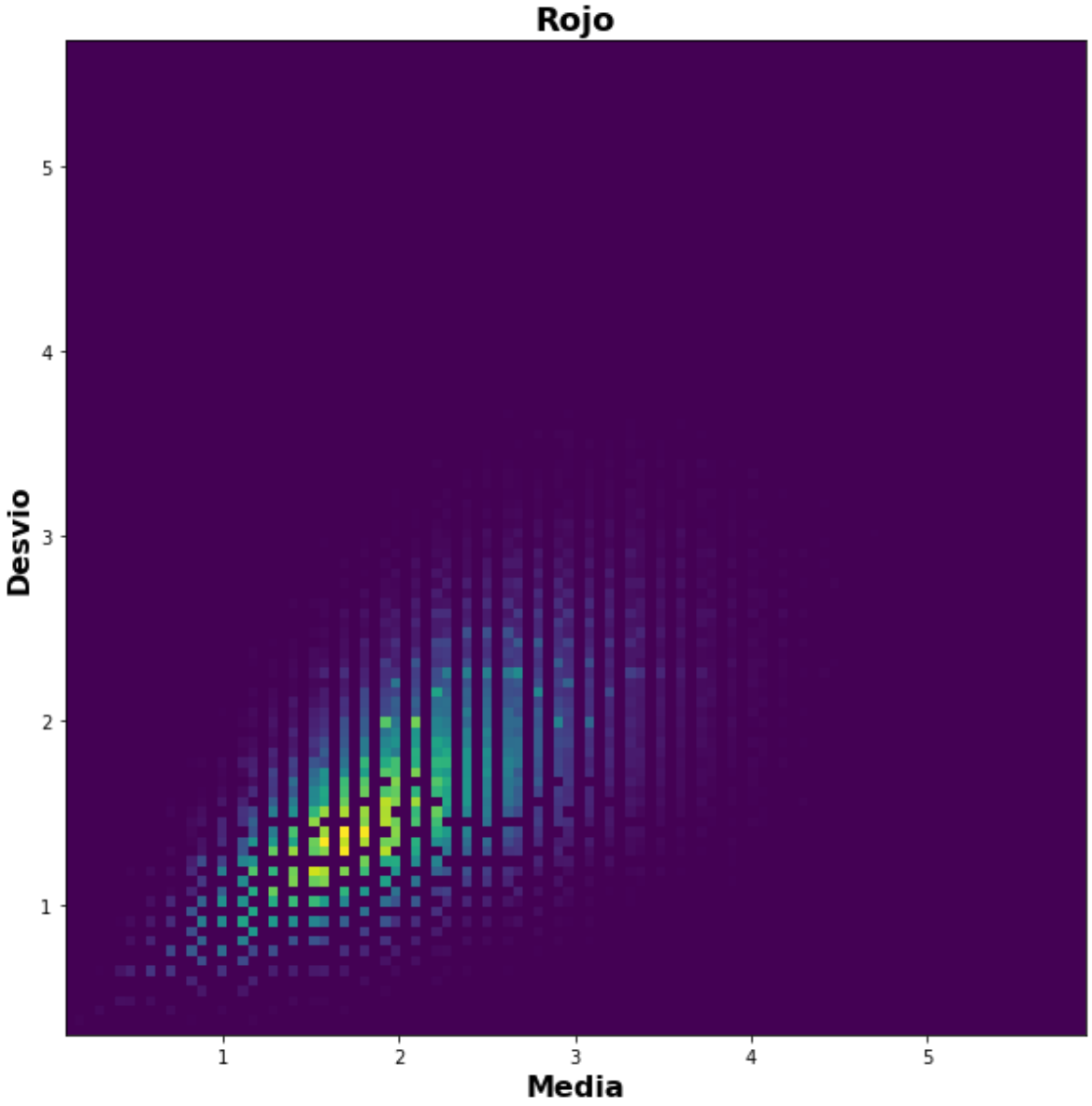
In [9]:

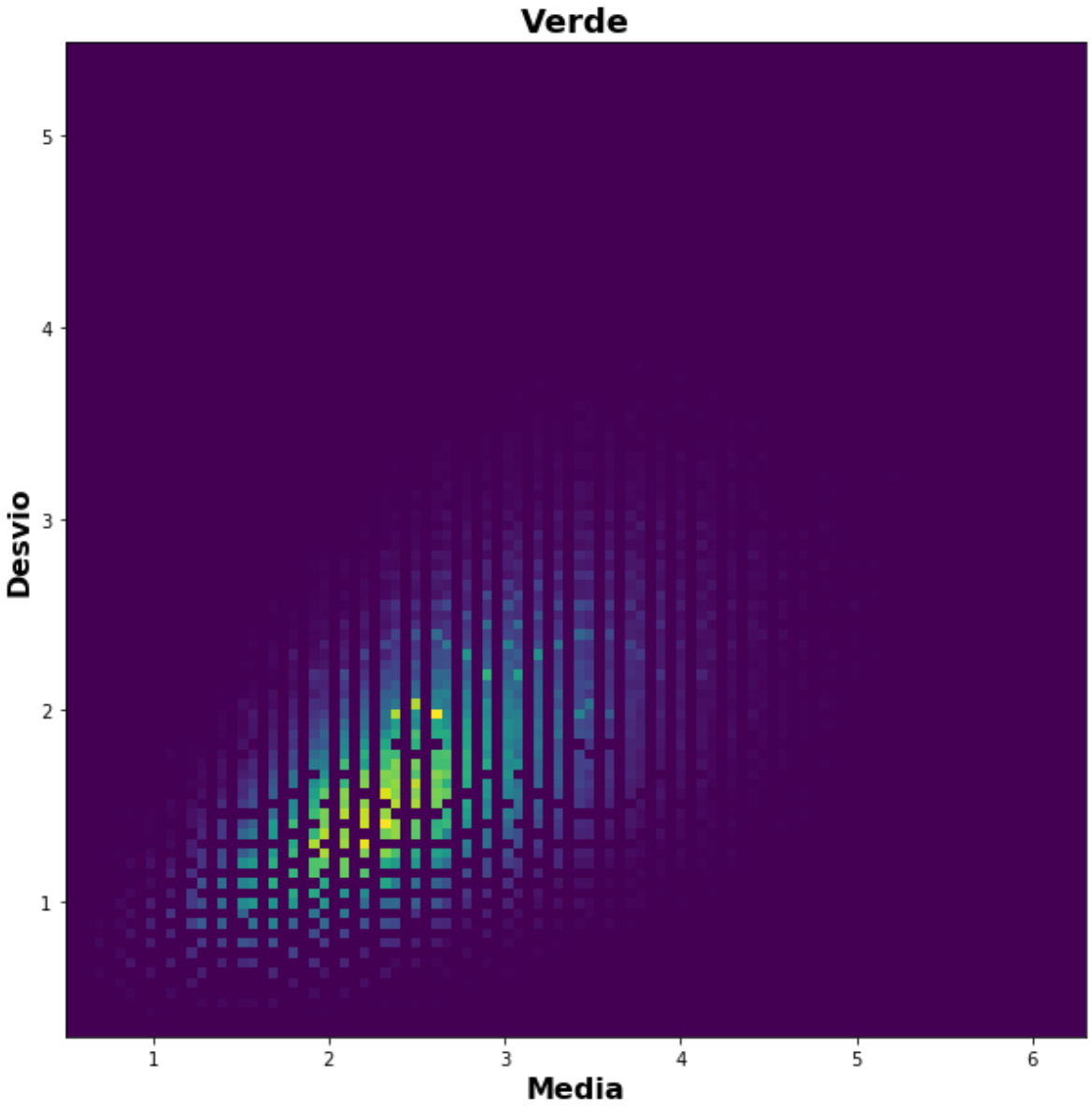
```
def plotHist2d(meanImgs, stdImgs):
    dec = 100
    for chNum, ch in enumerate(CHANNELS):
        channelStd = np.ravel(stdImgs[:,:,chNum])
        channelMean = np.ravel(meanImgs[:,:,chNum])

        #Realizamos el plot del histograma 2d
        plt.figure(figsize=(10,10))
        plt.title(ch, fontweight='bold', fontsize=18)
        plt.xlabel('Media', fontweight='bold', fontsize=16)
        plt.ylabel('Desvio', fontweight='bold', fontsize=16)
        cb = plt.hist2d(channelMean[::dec], channelStd[::dec], bins=100)
```

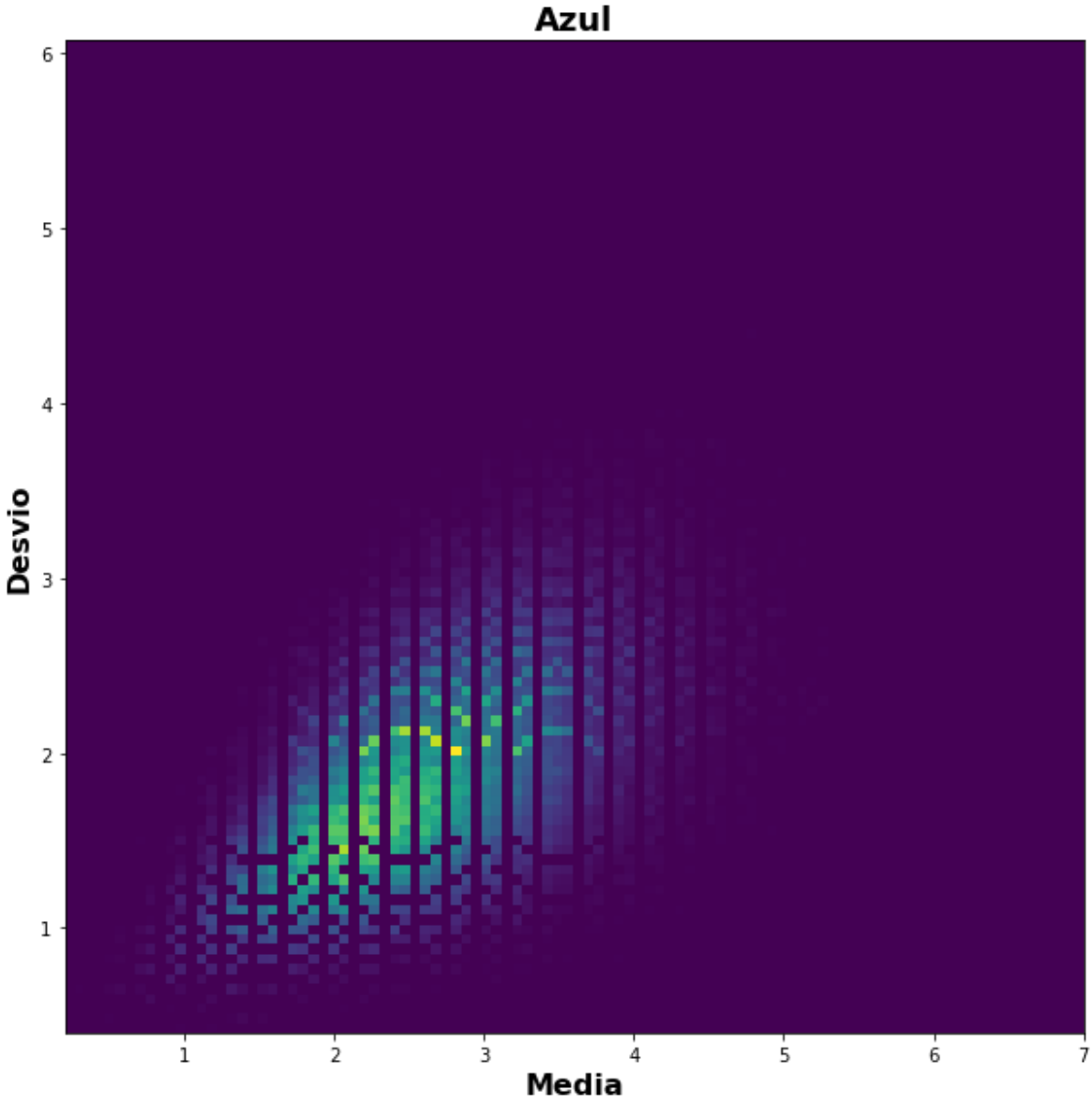
iPhone

In [10]:

```
plotHist2d(img_mediaIphone, img_stdIphone)
```




Verde
Desvio
Media
1
2
3
4
5
6

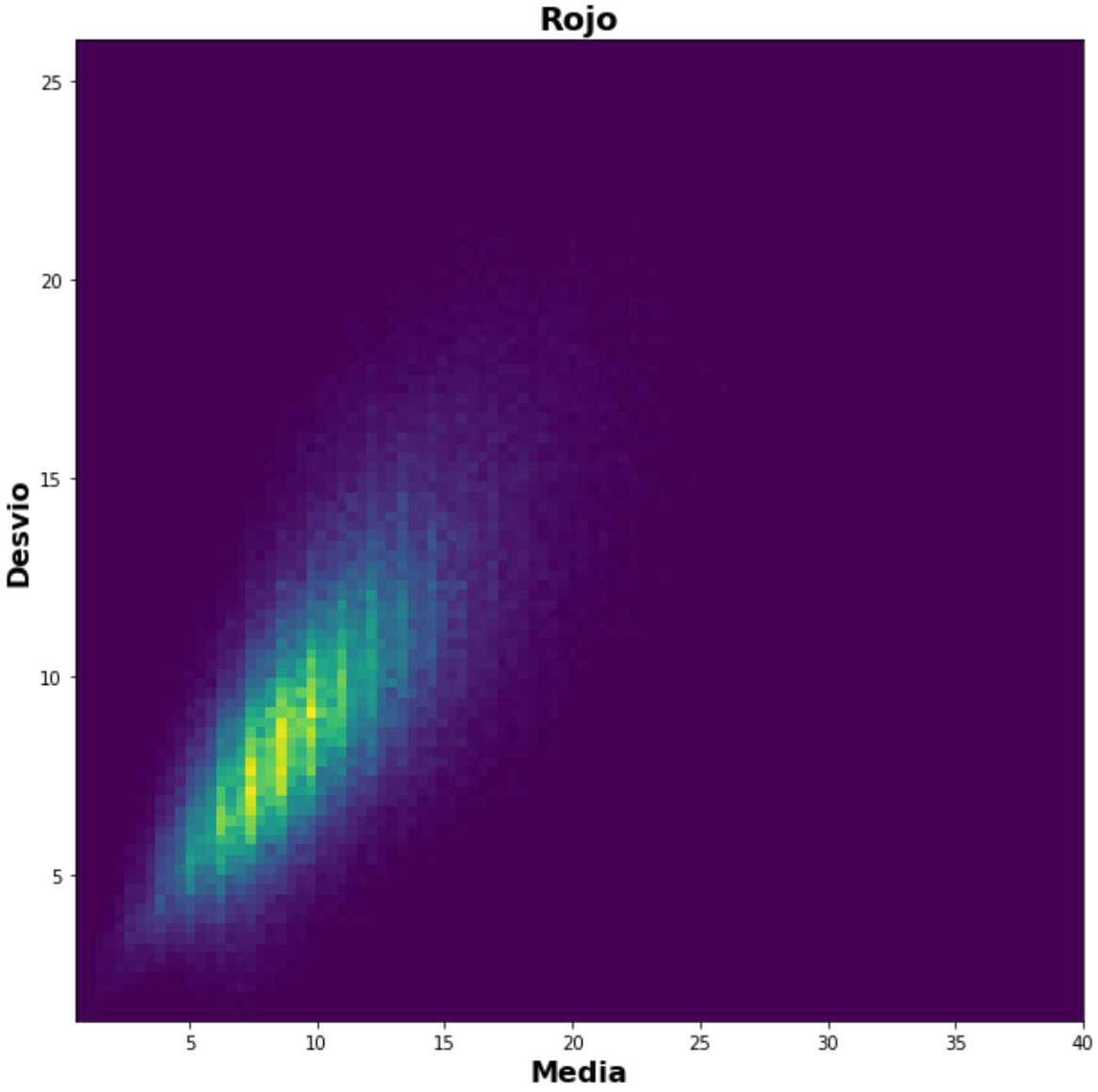

Azul
Desvio
Media
1
2
3
4
5
6
7

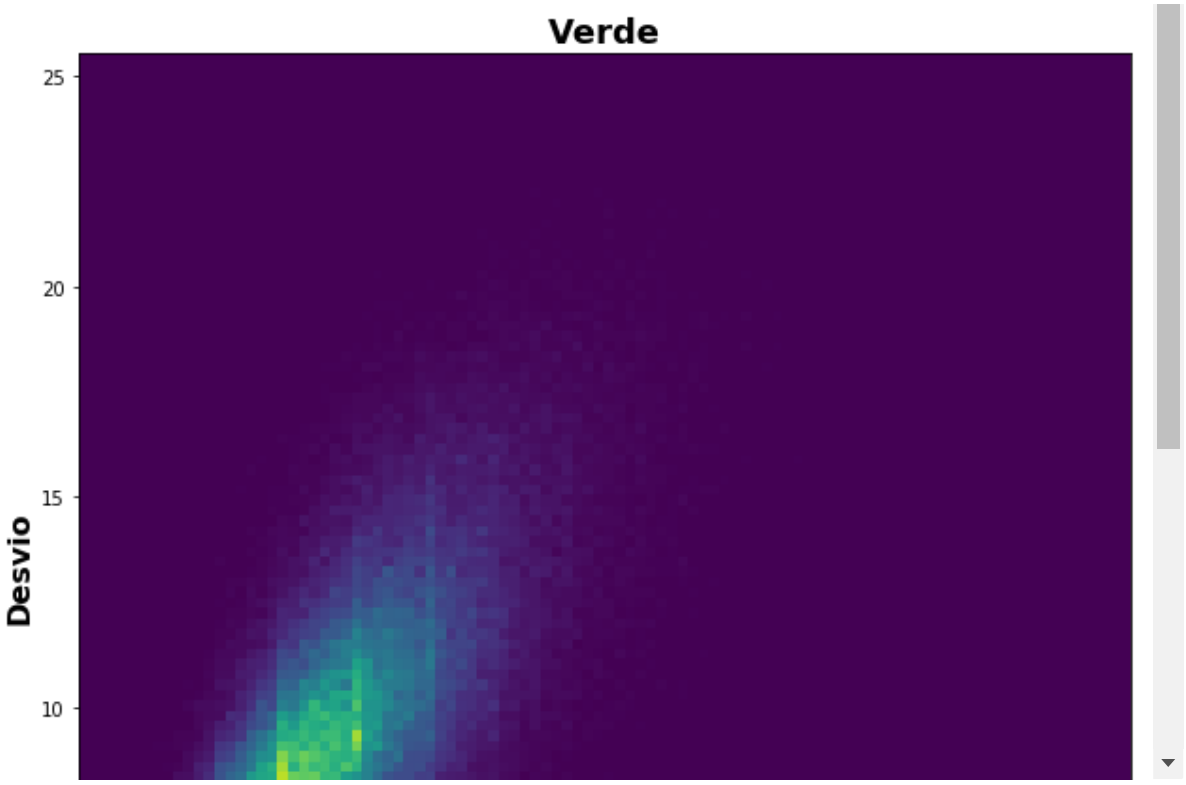
Se puede ver que las tres imagenes poseen rayas las cuales no tienen ningun color, esto se debe a que no hay una relacion para ese valor de media con un valor de desvío estandar.

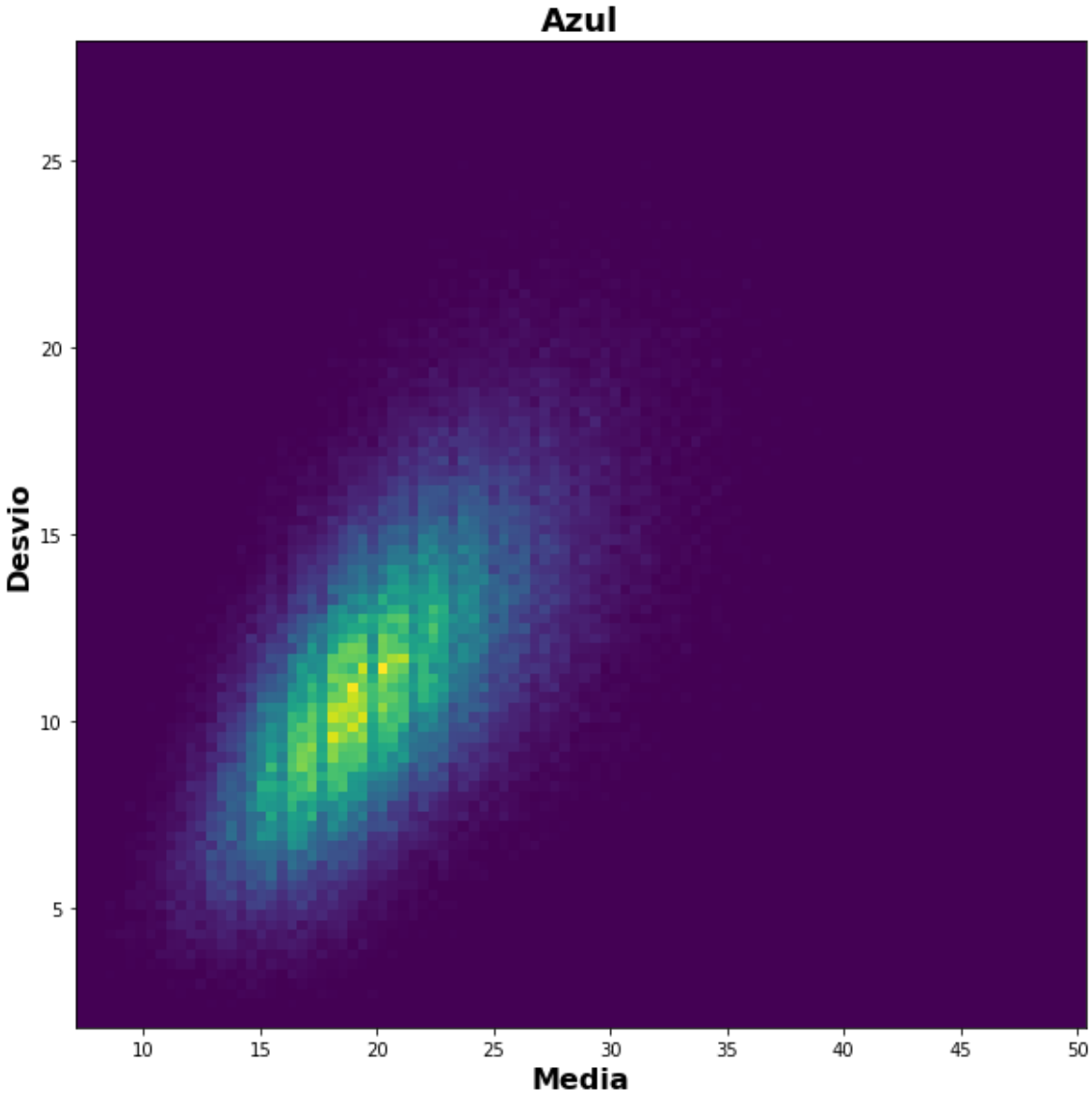
Xiaomi

In [11]:

```
plotHist2d(img_mediaXiaomi, img_stdXiaomi)
```







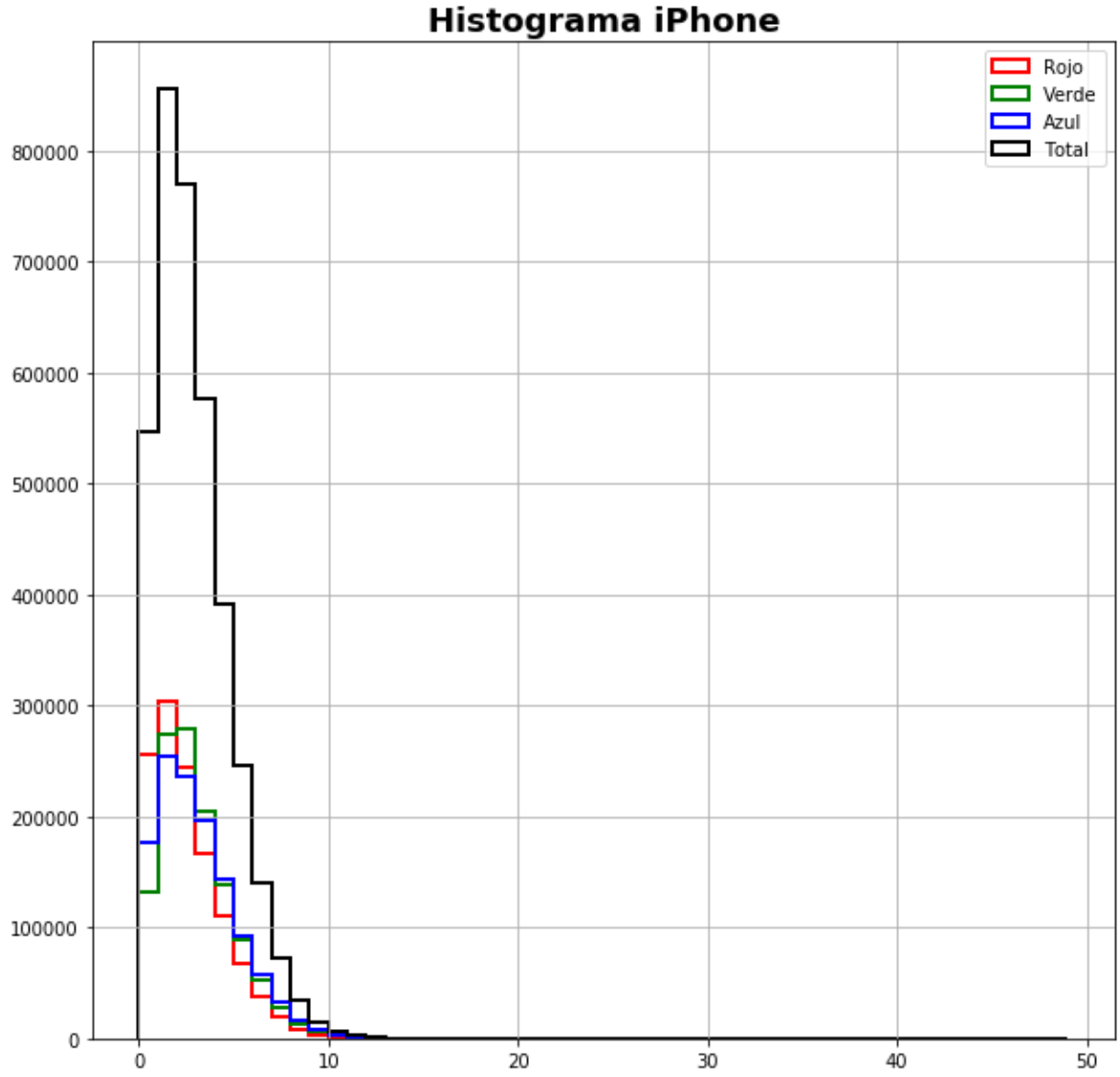
Histogramas

In [12]:

```
def plotHist(stackImgs, title = ''):
    dec = 100
    i_max = 50
    colors = ['red', 'green', 'blue']
    plt.figure(figsize=(10,10))
    plt.title('Histograma ' + title, fontsize=18, fontweight='bold')
    plt.grid()
    for chNum, ch in enumerate(CHANNELS):
        allChannel = np.ravel(stackImgs[:,:,:,chNum])
        _ = plt.hist(allChannel[::dec], bins=range(i_max), color=colors[chNum],hist


    _ = plt.hist(np.ravel(stackImgs)[::dec], bins=range(i_max), color='black', hist
    plt.legend(['Rojo', 'Verde', 'Azul', 'Total'])
```

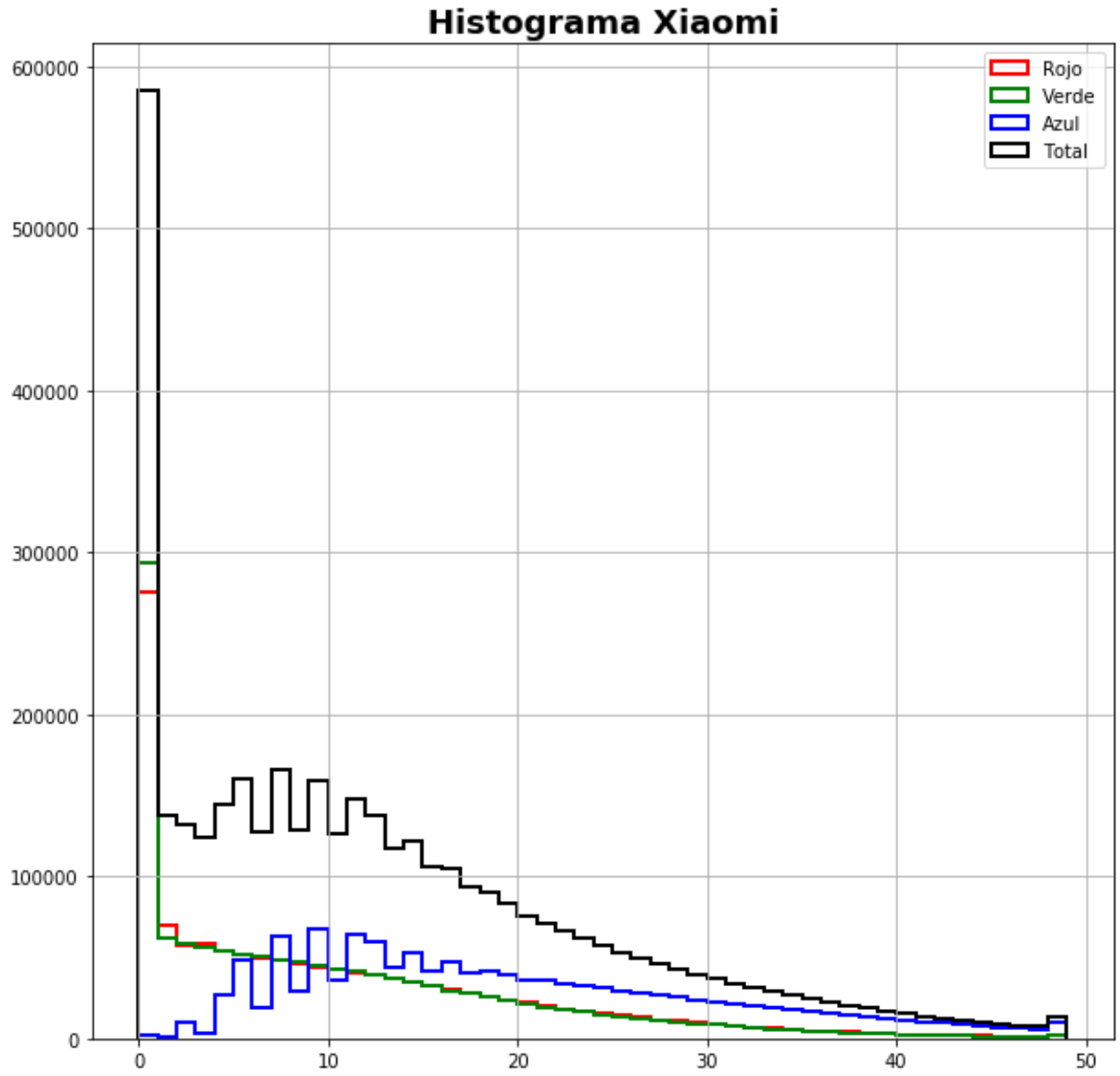
In [13]:

```
plotHist(stackIphone, 'iPhone')
```



In [14]:

```
plotHist(stackXiaomi, 'Xiaomi')
```



Conclusiones 1

Se puede apreciar como con dos celulares con sensores distintos al tomar fotos con el sensor tapado se obtienen diferentes resultados.

La solución ideal sería que el histograma tenga un pico en el cero y luego no posea más valores ya que el sensor se encuentra tapado. En ambos casos se puede ver que no es así.

El que posee el pico más elevado en cero es el Xiaomi, pero para este sensor la cantidad de pixeles con valores mayores a cero son muchos más (en la figura se puede apreciar que recién en 50 la curva del histograma se aproxima a cero). En cambio en el iPhone es distinto, si bien no posee un pico tan elevado en cero como el Xiaomi, la curva se hace cero mucho más rápido, es decir no se ven pixeles con valores mayores a 10, aproximadamente.

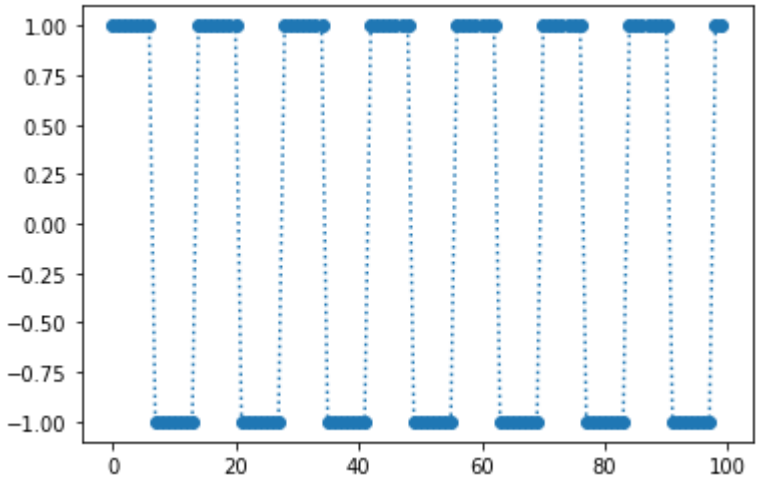
Otra diferencia que se puede apreciar es que para el iPhone, todos los canales poseen una estadística similar, o sea las curvas son parecidas. Esto no sucede con el Xiaomi, la curva roja y verde se asemejan pero la azul posee una forma totalmente distinta, es mas achatada y alargada.

2) MTF

El siguiente punto solo se realizo con el iPhone

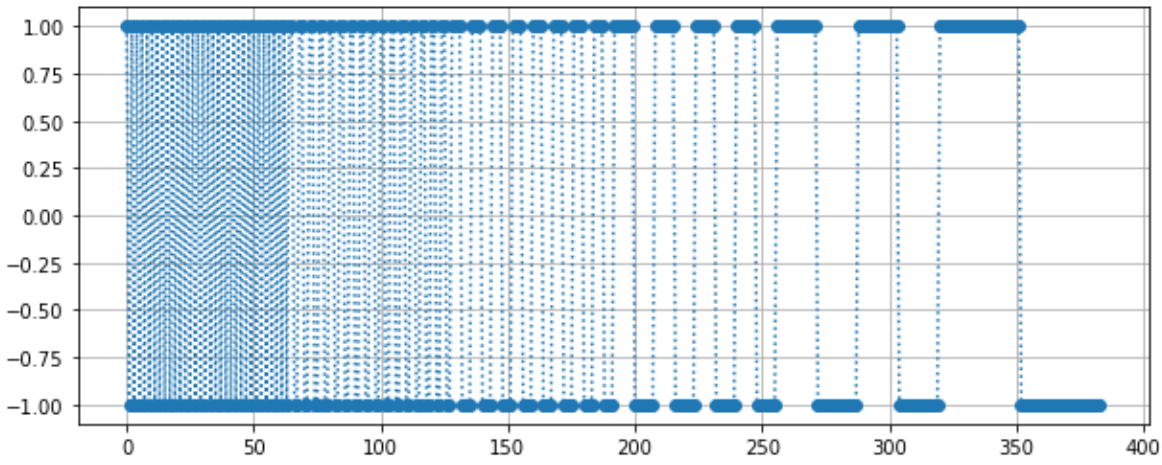
In [15]:

```
def generar_onda_cuadrada(largo, periodo):
    onda_cuadrada=np.zeros(largo)
    for i in range (0,largo):
        if np.mod(i,periodo)<periodo/2:
            onda_cuadrada[i]=1
        if np.mod(i,periodo)>=periodo/2:
            onda_cuadrada[i]=-1
    return onda_cuadrada
plt.plot(generar_onda_cuadrada(largo=100, periodo=14), ':o');
```



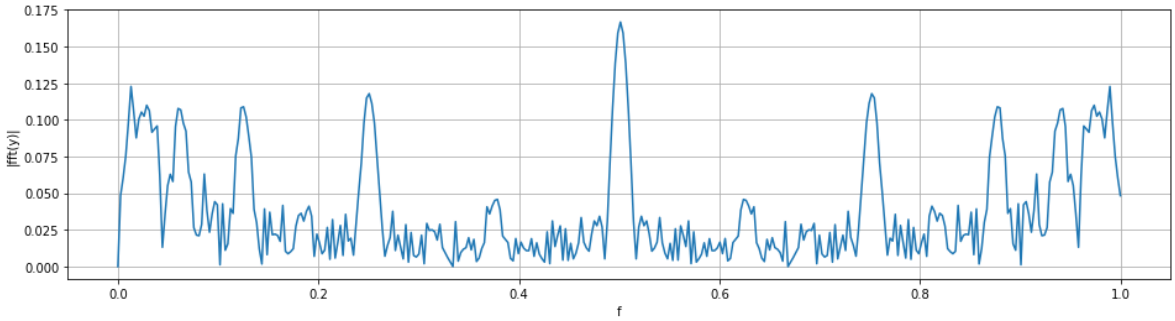
In [16]:

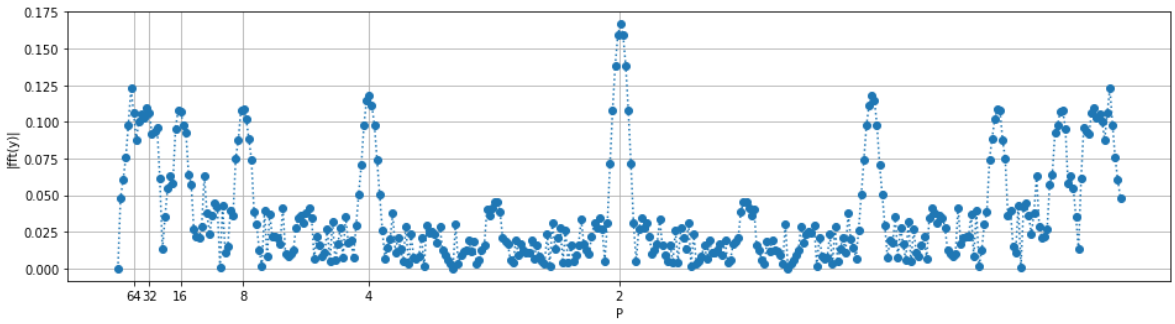
```
n_rep = 8
largo_max = 10
ys = []

anchos = [ 2, 4, 8,16,32,64]
repeticiones = [32,16, 8, 4, 2, 1]

for a, r in zip(anchos, repeticiones):
    ys.append(generar_onda_cuadrada(a*r, a))

y = np.hstack(ys)
n_samples = len(y)

t = np.linspace(0, n_samples, n_samples)


n_show = n_samples // 10
plt.figure(figsize=(10,4))
plt.grid()
plt.plot(y, ':o');
```



In [17]:

```
y_f = np.fft.fft(y)

ancho_pantalla = 1920
f = np.linspace(0, 1, n_samples)

plt.figure(figsize=(16,4))
plt.grid()

plt.ylabel('|fft(y)|')
plt.plot(f, np.abs(y_f)/n_samples)
plt.xlabel('f');

x = [1/(a) for a in anchos]
labels = [a for a in anchos]
plt.figure(figsize=(16,4))
plt.ylabel('|fft(y)|')
plt.grid()
n_show = n_samples
plt.plot(f[:n_show], np.abs(y_f[:n_show])/n_samples , ':o')
plt.xticks(x, labels);
plt.xlabel('P');
```





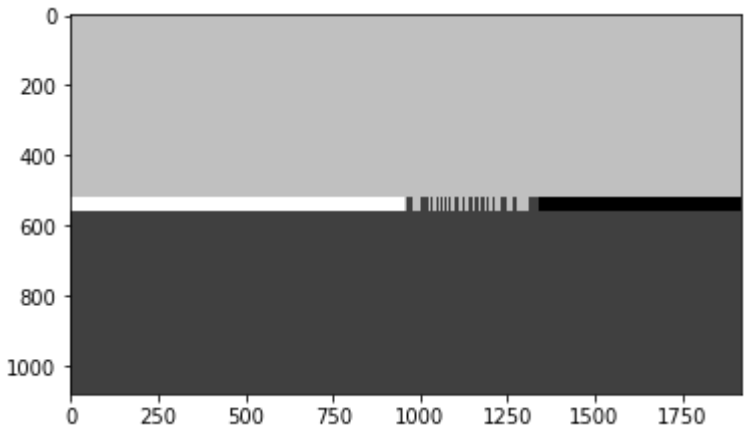
A continuación generamos el patrón al que le tomaremos la foto con el celular

In [18]:

```
# ajustar según pantalla donde se saca la foto, poner la resolución del monitor

# full hd (1080p)
# ancho_pantalla = 1920
# alto_pantalla = 1080

# # típico notebooks:
# ancho_pantalla = 1368
# alto_pantalla = 768

# alguien tiene pantalla "retina" o 4k?
ancho_pantalla = 1920
alto_pantalla = 1080

alto_franja_central_px = 40

oscuro = 64
claro = 192

medio = (claro + oscuro) / 2
amplitud = (claro - oscuro) / 2

onda = medio + amplitud * y

centro_izq = 255 * np.ones((alto_franja_central_px, ancho_pantalla // 2))
centro_der = 0 * np.ones((alto_franja_central_px, ancho_pantalla // 2 - len(onda)))

arriba = claro * np.ones((alto_pantalla // 2 - alto_franja_central_px // 2, ancho_p
centro = np.hstack((centro_izq, np.tile(onda, (alto_franja_central_px, 1)), centro_
abajo = oscuro * np.ones((alto_pantalla // 2 - alto_franja_central_px // 2, ancho_p

patron_mtf = np.vstack((arriba, centro, abajo)).astype(np.uint8)

patron_mtf = cv2.cvtColor(patron_mtf, cv2.COLOR_GRAY2RGB)

plt.imshow(patron_mtf)
cv2.imwrite('./fotos/patron_mtf.png', patron_mtf)
```

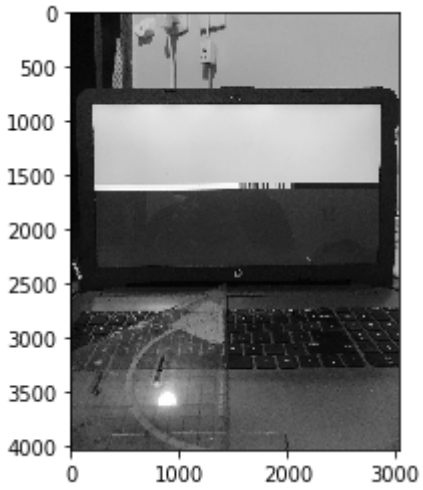
Out[18]:

True



In [19]:

```
mtf_leer = cv2.imread('./fotos/mtf1.JPG')
mtf_leido_gray = cv2.cvtColor(mtf_leer, cv2.COLOR_BGR2GRAY)
plt.imshow(mtf_leido_gray, cmap='gray');
```



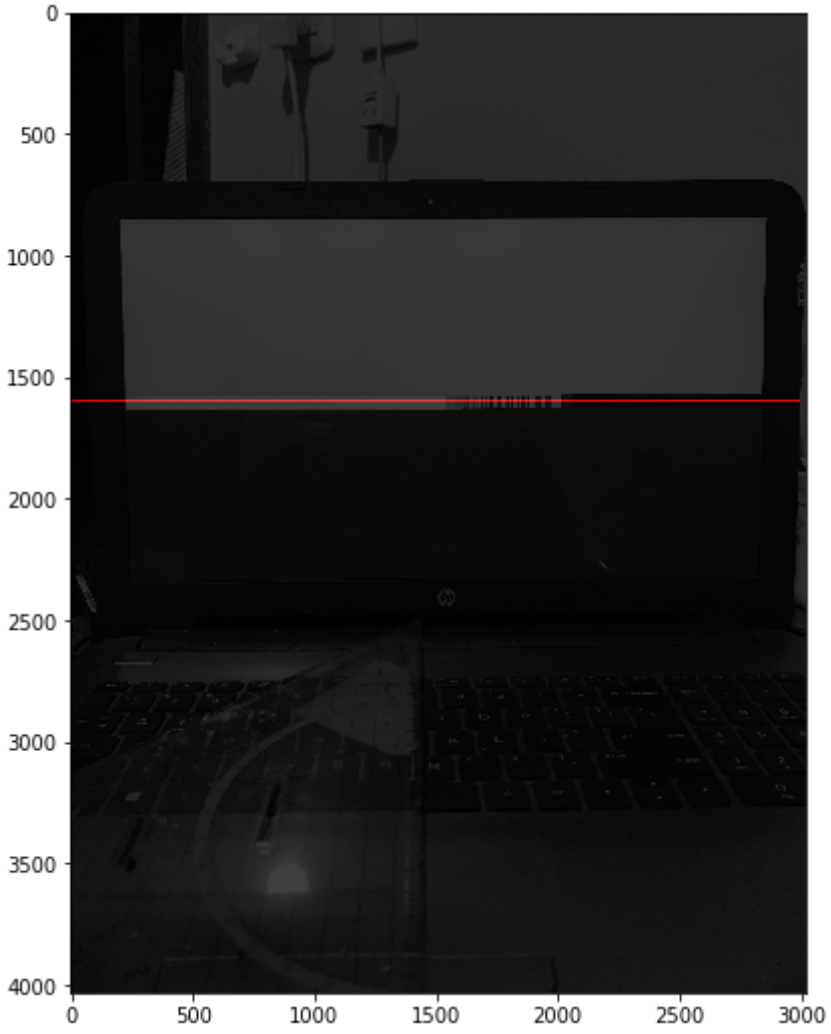
Mediciones

Realizando las mediciones que dijeron en el video sacamos que la distancia desde donde se saco la foto hasta la pantalla era de 50cm, y el ancho del patrón de 7 cm aproximadamente. En estos calculos puede haber cierto error debido a que se midió con una regla.

A continuación trazamos una linea roja para encontrar las coordenadas en las que se encuentra el patrón de la foto que tomamos

In [20]:

```
# Buscar de pegarle con un corte a la zona del mtf
ajustar_linea_mtf = cv2.cvtColor(mtf_leido_gray // 4, cv2.COLOR_GRAY2RGB)

ancho_foto = mtf_leido_gray.shape[1]

fila_mtf = 1600
zoom = 1600

# rellenar con principio y fin del patrón mtf en la foto:
columnas_mtf = slice(0, 3000)

ajustar_linea_mtf[fila_mtf, columnas_mtf, 0] = 255
plt.figure(figsize=(16,9));
#cv2.line(ajustar_linea_mtf,(0,1625),(2500,1625),(255,0,0),2,cv2.LINE_AA) #Trazamos
plt.imshow(ajustar_linea_mtf);
```

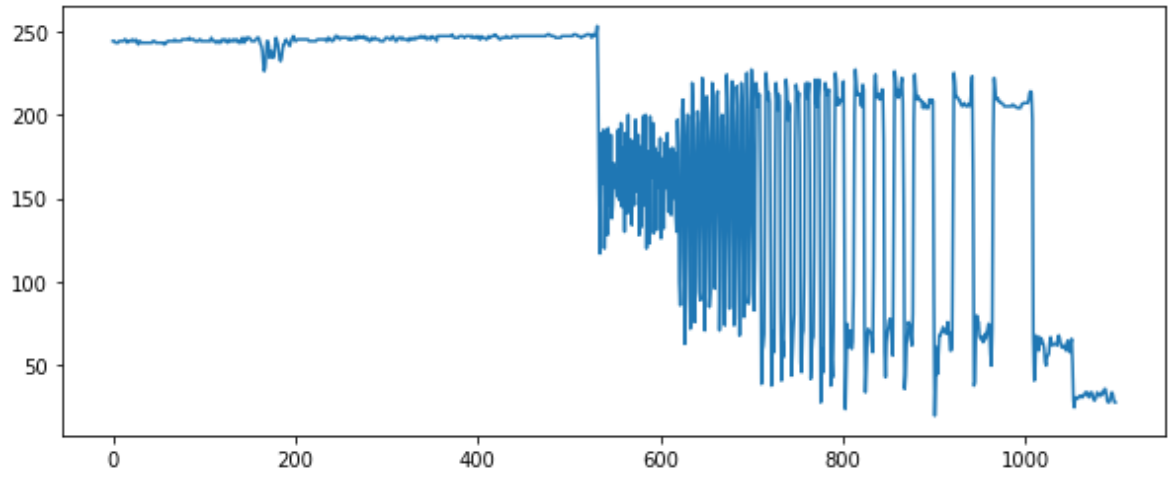


In [21]:

```
# refinamos el corte
zoom = 1000
fila_mtf = 1600

columnas_mtf = slice((ancho_foto // 2 - zoom // 2), (ancho_foto // 2 + zoom // 2) +
```

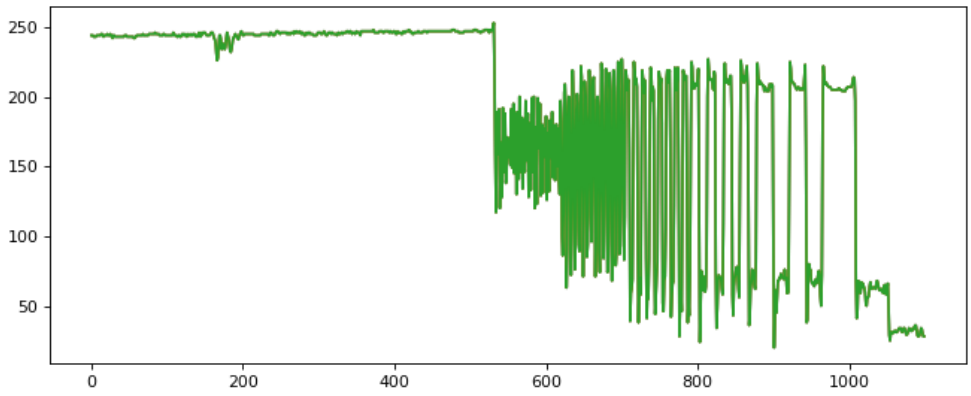
In [22]:

```
# dibujamos con más detalle el corte
y_est = mtf_leido_gray[fila_mtf, columnas_mtf]

plt.figure(figsize=(10,4))
plt.plot(y_est);
```



In [33]:

```
# Generamos gráfico para leer donde cortar para recuperar la estimación de la y de

# por ahí si no quedó perfectamente centrado necesitan otro valor diferente de +/-
claro_est = mtf_leido_gray[fila_mtf, columnas_mtf]
oscuro_est = mtf_leido_gray[fila_mtf, columnas_mtf]

%matplotlib notebook
plt.figure(figsize=(10,4))
plt.plot(y_est)
plt.plot(claro_est)
plt.plot(oscuro_est)
#En caso de no ver la imagen correr varias veces esta celda, por lo general corrien
```



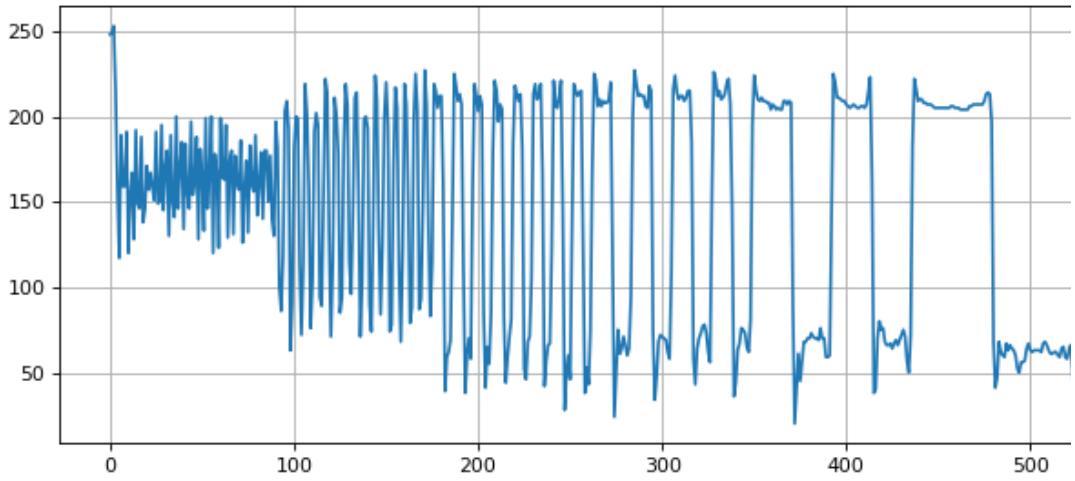
Out[33]:

```
[<matplotlib.lines.Line2D at 0x7fc58ce3e590>]
```

In [34]:

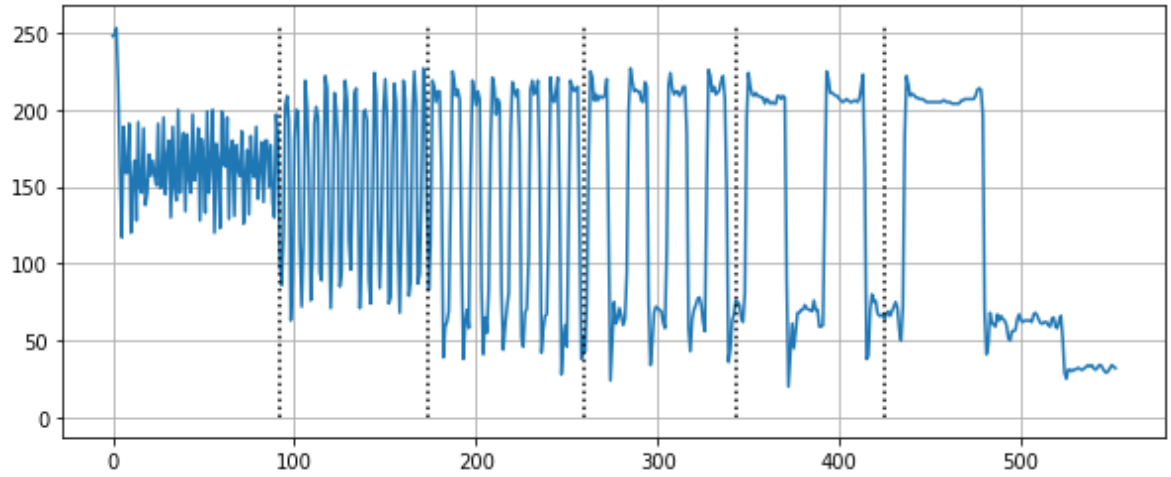
```
# Leer del gráfico de arriba

k_start = 529
k_end = 1082

y_est_r = y_est[k_start:k_end]

%matplotlib notebook
plt.figure(figsize=(10,4))
plt.grid()
plt.plot(y_est_r);
#Lo mismo que arriba, correr mas de una vez si no se observa nada
```

Figure 1



In [25]:

```
# llenar con bordes medidos del gráfico:
bordes= [92,174,259,343, 425]

%matplotlib inline
plt.figure(figsize=(10,4))
plt.grid()
plt.plot(y_est_r)

for b in bordes:
    plt.plot([b, b], [0, 255], ':k')
```

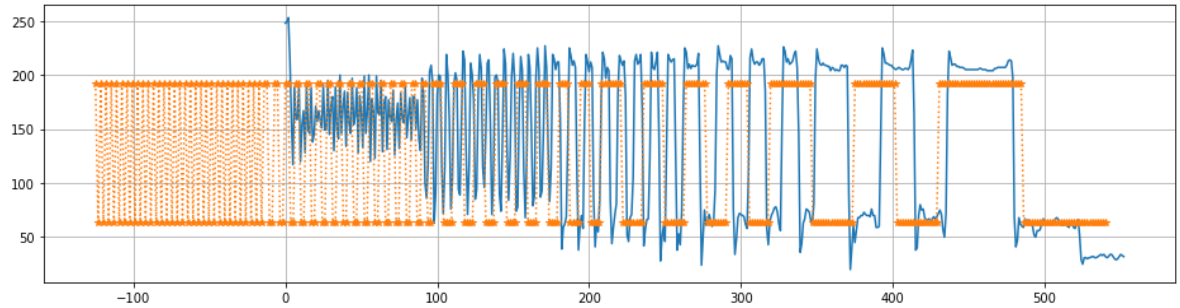


In [26]:

```
# comparamos con la función que le "metimos de entrada" al sistema
%matplotlib inline
plt.figure(figsize=(16,4))
plt.grid()
plt.plot(y_est_r)

# -9,455 son offsets para que quede bien, cambiarlo según lo que obtuvieron
offset_l = -125
offset_r = 540

x_est=np.linspace(offset_l,offset_r,len(onda))
plt.plot(x_est, onda, ':*')
```

Out[26]:

```
[<matplotlib.lines.Line2D at 0x7fc58cdce410>]
```

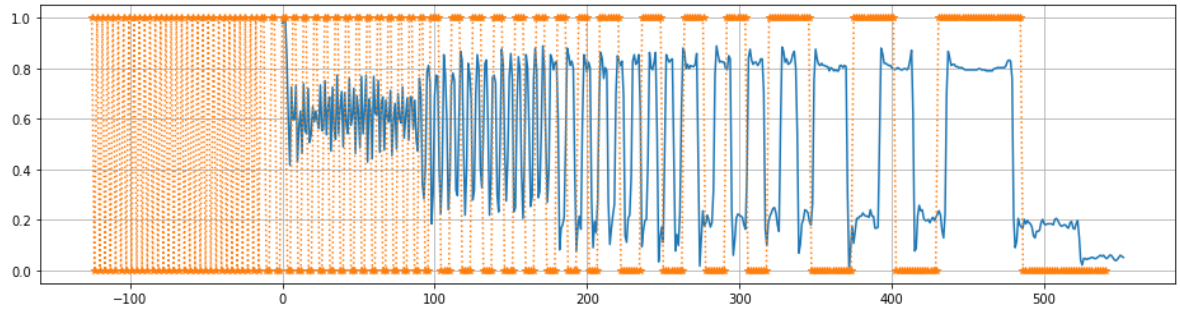


In [27]:

```
# idem normalizado
%matplotlib inline
plt.figure(figsize=(16,4))
plt.grid()
plt.plot((y_est_r-np.min(y_est_r))/(np.max(y_est_r)-np.min(y_est_r)))
x_est=np.linspace(offset_l,offset_r,len(onda))
plt.plot(x_est, (onda-np.min(onda))/(np.max(onda)-np.min(onda)), ':*')
```

Out[27]:

```
[<matplotlib.lines.Line2D at 0x7fc58d003b90>]
```



In [28]:

```
# Sacar fotos y tomar nota de la geometría de cómo tomaron la foto en particular di
# y ángulo subtendido por zona usada para calcular mtf:
distancia_monitor_mm = 500
ancho_zona_mtf_mm = 70

# Calcular ángulo que genera la zona usada para medir mtf (cateto menor en la panta
# y cateto mayor distancia entre cámara y pantalla)
angulo_zona_mtf_deg = np.arctan2(ancho_zona_mtf_mm, distancia_monitor_mm)*180/np.pi
ancho_zona_mtf_px = len(y_est_r)
```

In [29]:

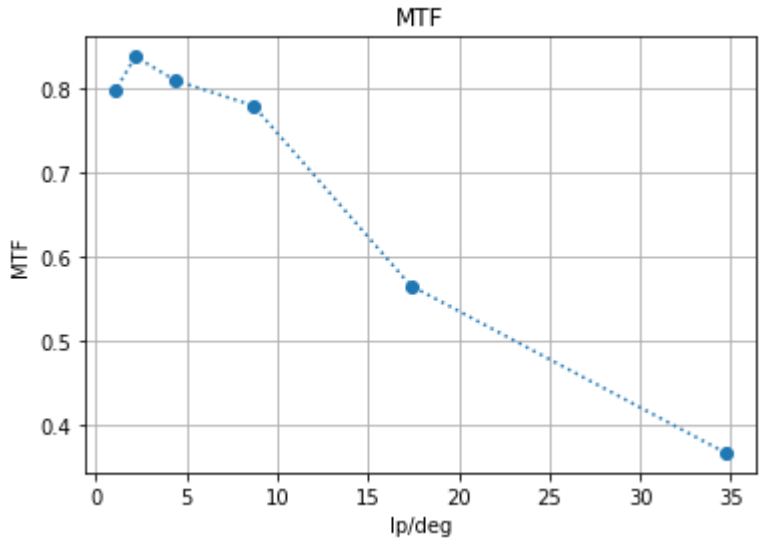
```
# Calculamos grados por pixel para usar en el gráfico
deg_per_px = ancho_zona_mtf_px / angulo_zona_mtf_deg
```

$$\mathrm{MTF}_{\mathrm{Local}} = \frac{i_{max} - i_{min}}{i_{max} + i_{min}}$$

In [30]:

```
# medimos el contraste en cada tramo y lo graficamos
tramos = np.split(y_est_r, bordes)

mtfs = []

for tramo in tramos:
    i_max = np.max(tramo).astype(np.float32)
    i_min = np.min(tramo).astype(np.float32)
    mtf = (i_max - i_min) / (i_max + i_min)
    mtfs.append(mtf)
    print(mtf)

plt.grid(); plt.title('MTF')
plt.plot(1/np.array(anchos)*deg_per_px, mtfs, ':o')
plt.ylabel('MTF');
plt.xlabel('lp/deg');
```

```
0.36756757
0.56551725
0.7786561
0.80876493
0.8367347
0.7975708
```



Conclusiones 2

En el eje x podemos ver los pares de líneas por grado, mientras que en el eje y tenemos el valor del MTF. Se puede ver que a medida que aumenta la frecuencia espacial (aumenta X) va disminuyendo la respuesta del sistema óptico lo cual se puede deber a la lente, la distancia a la que estamos o por como esta enfocado. El valor MTF50, que es cuando la amplitud es el 50% de la señal de entrada, nos da alrededor de 23 pares de líneas por grado.