Estimación de MTF

March 5, 2022

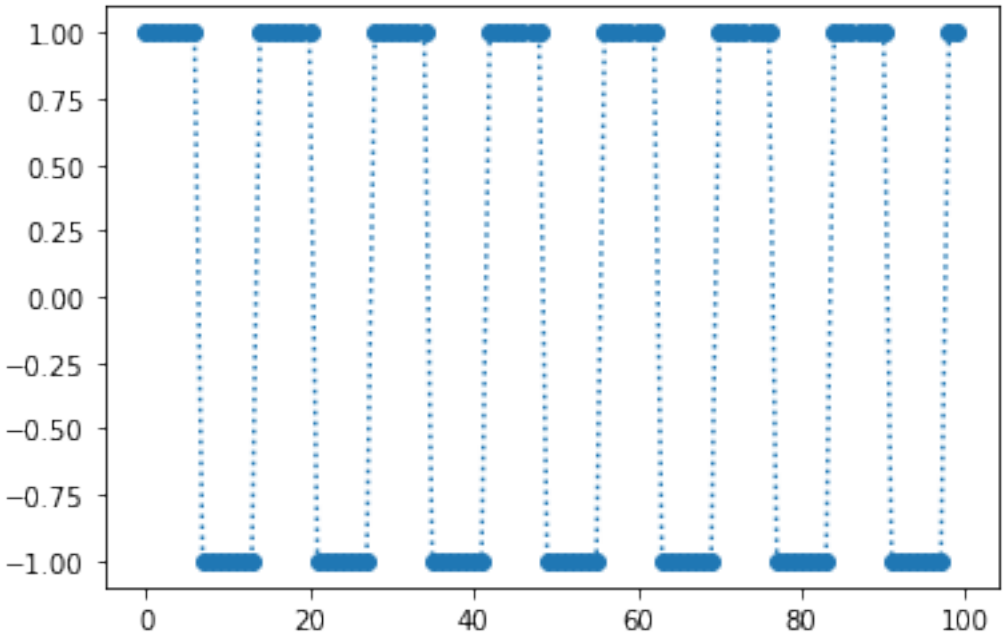
[1]:
```
# Cargamos las bibliotecas que usamos casi siempre
import cv2
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal
%matplotlib inline
print('OpenCV ' + cv2.__version__)
print('numpy ' + np.__version__)
```

```
OpenCV 3.4.2
numpy 1.18.1
```

Creación de función que genere onda cuadrada, con el objetivo de identificar la respuesta en frecuencia ideal.

[113]:
```
def generar_onda_cuadrada(largo, periodo):
    t = np.linspace(0, largo, 100, endpoint=True)
    onda = signal.square(2 * np.pi *1/periodo * t)
    return onda

plt.plot(generar_onda_cuadrada(largo=100, periodo=14), ':o')
```

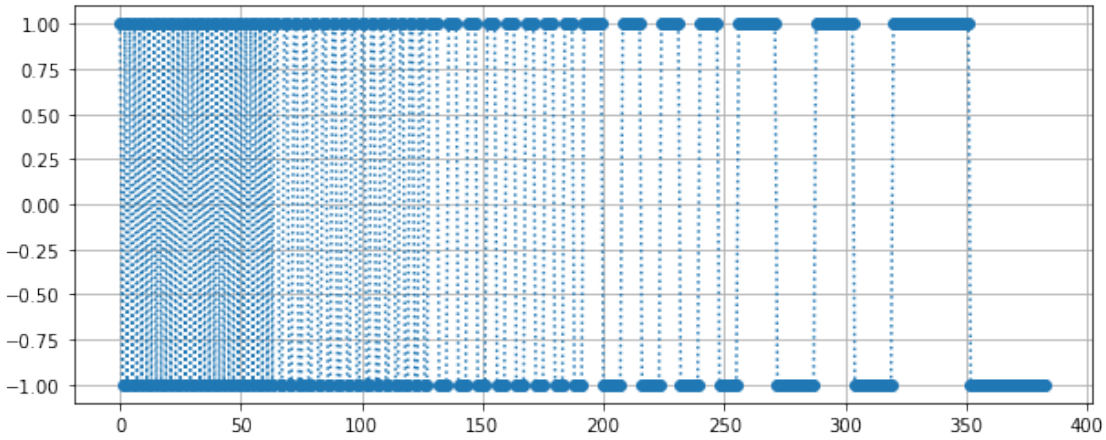
[2]:
```
[<matplotlib.lines.Line2D at 0x88d4b38f48>]
```



Utilizando la función que genera una onda cuadrada, realizamos un patrón.

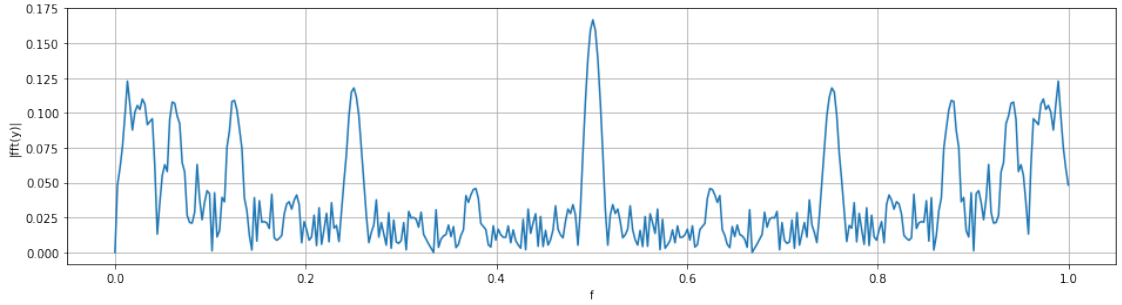
[90]:
```
n_rep = 8
largo_max = 10
ys = []

anchos =       [  2,4,8,16,32,64]
repeticiones = [32,16, 8, 4, 2, 1]
# repeticiones = [ 100, 0, 0, 0, 0, 0]


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(t[:n_show], y[:n_show], ':o')
plt.plot(y, ':o')
```

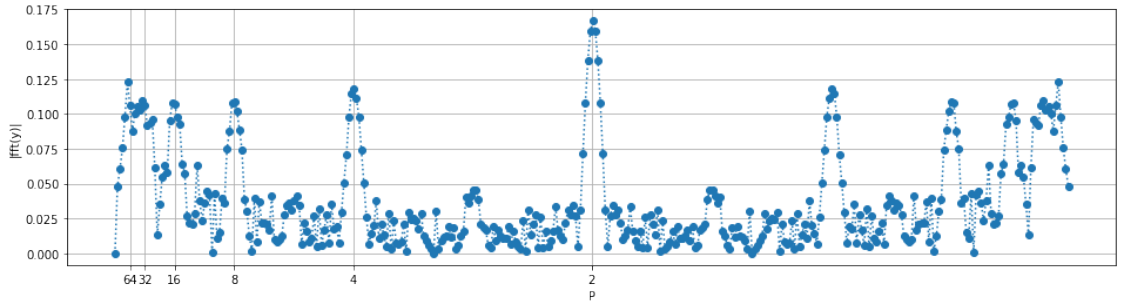
[90]:
```
[<matplotlib.lines.Line2D at 0x88e0d47ac8>]
```



De izquierda a derecha, la frecuancia va disminuyendo. En un principio varía a gran velocidad, y a medida que transcure el tiempo va disminuyendo. Esto se relaciona con que tan rapido se modifica la fase.

[91]:
```
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');
```





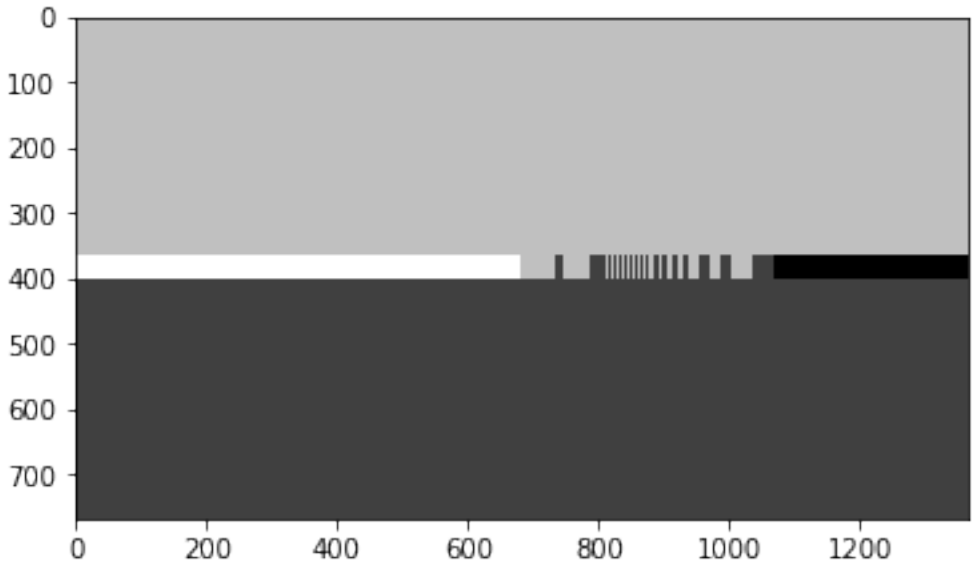
Lo que se pudo ver fueron las componentes en frecuencia de la señal generada. Se hace mediante el uso de la transformada de Fourier. En el grafico discretizado se visibiliza como eje x el período, más útil para el análisis que la frecuencia.

[92]:
```
## 1. Generar patrón
```

[93]:
```
# ajustar según pantalla donde se saca la foto, poner la resolución del monitor


ancho_pantalla = 1366
alto_pantalla = 768



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_pantalla))
centro = np.hstack((centro_izq, np.tile(onda, (alto_franja_central_px, 1)),␣
 ↪centro_der))
abajo = oscuro*np.ones((alto_pantalla//2 - alto_franja_central_px//2,␣
 ↪ancho_pantalla))

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('patron_mtf.png', patron_mtf)
```

[93]: True

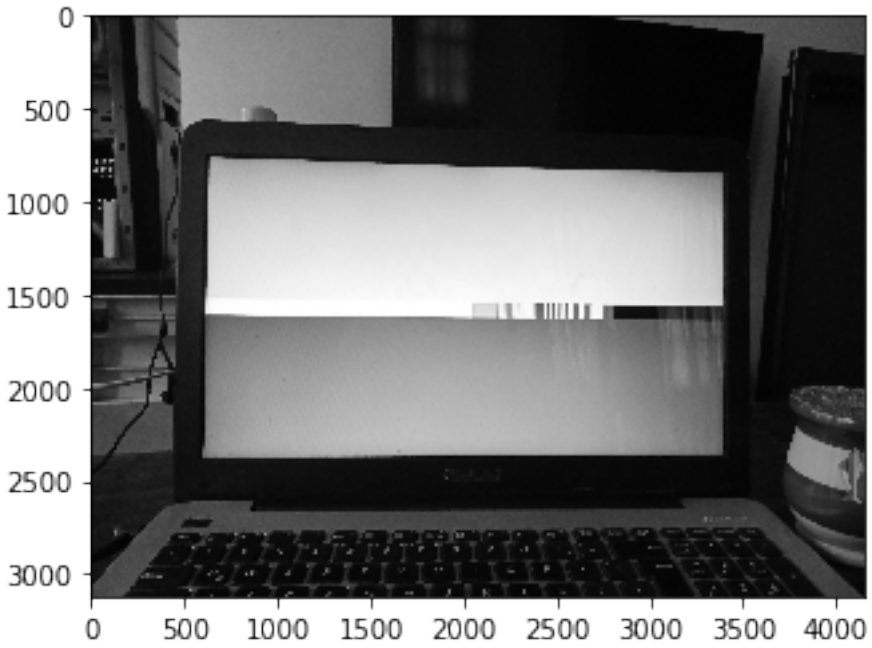


El patrón generado en su centro posee una frecuencia rápida (cambia el color en pocos pixeles) y a medida que se mueve a la derecha la frecuencia disminuye. Esta onda varía únicamente entre 64 y 192.

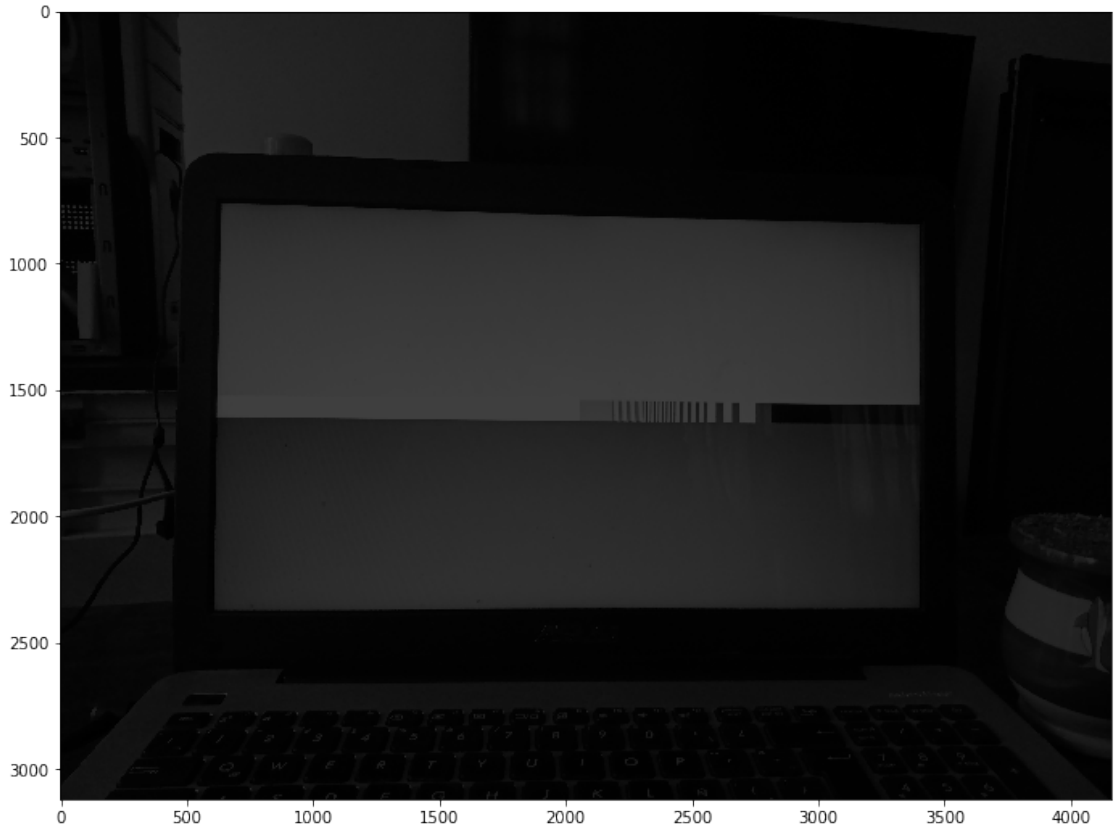
A continuación se captura la imagen, tomando nota de la deistancia de la cámara al display. Idealmente, un pixel en la pantalla coincide con un pixel en el patrón. La toma de ambiente ayuda al teléfono a calibrar el blanco.

[94]:
```
mtf_leer = cv2.imread('C:/Users/Images/mtf.jpg')

# pasamos a gris
mtf_leido_gray = cv2.cvtColor(mtf_leer, cv2.COLOR_BGR2GRAY)

plt.imshow(mtf_leido_gray, cmap='gray')
```

[94]:
```
<matplotlib.image.AxesImage at 0x88e0d5c608>
```



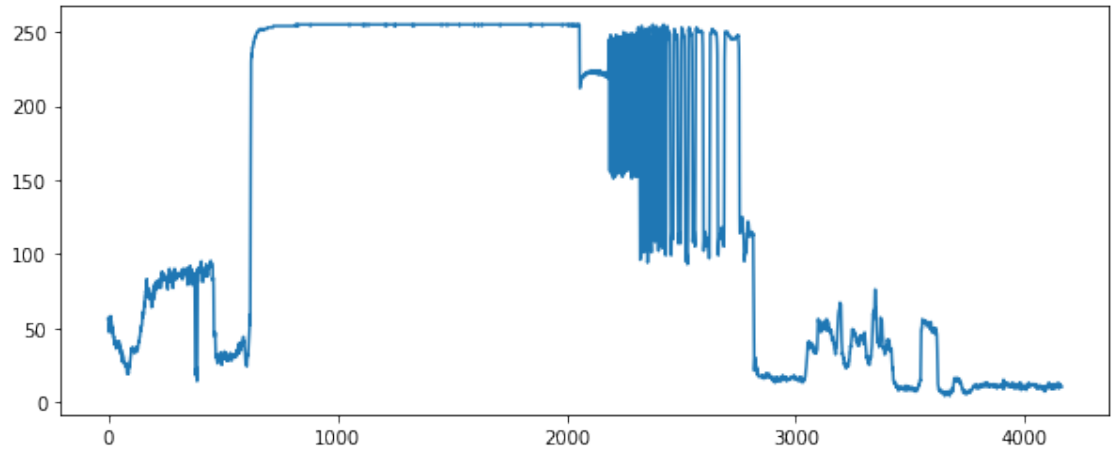
[95]:
```
# Buscar de pegarle con un corte a la zona del mtf
ajustar_linea_mtf = cv2.cvtColor(mtf_leido_gray//4, cv2.COLOR_GRAY2RGB)

ancho_foto = mtf_leido_gray.shape[1]


fila_mtf = 1580
zoom = 1600

# rellenar con principio y fin del patrón mtf en la foto:
columnas_mtf = slice(1240,3050)

ajustar_linea_mtf[fila_mtf, columnas_mtf, 0] =255
plt.figure(figsize=(16,9));
plt.imshow(ajustar_linea_mtf);
```

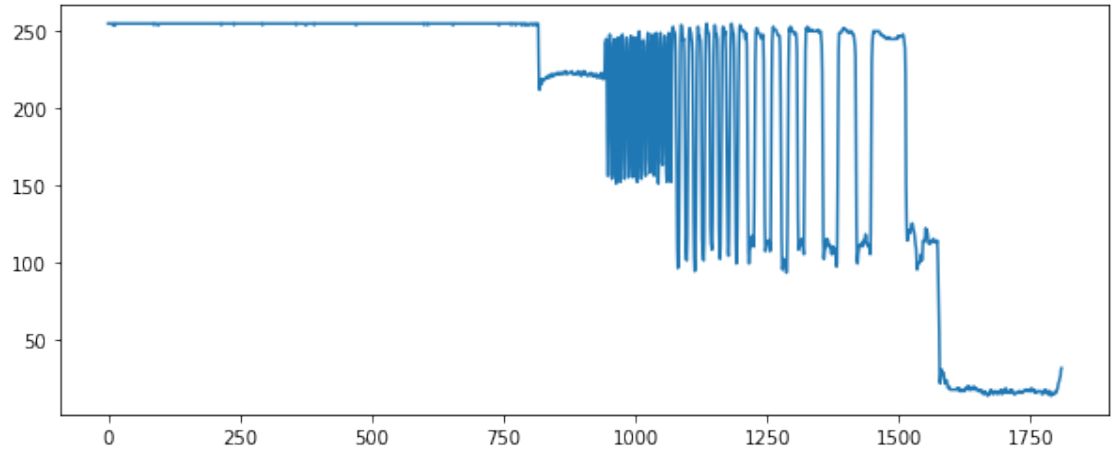


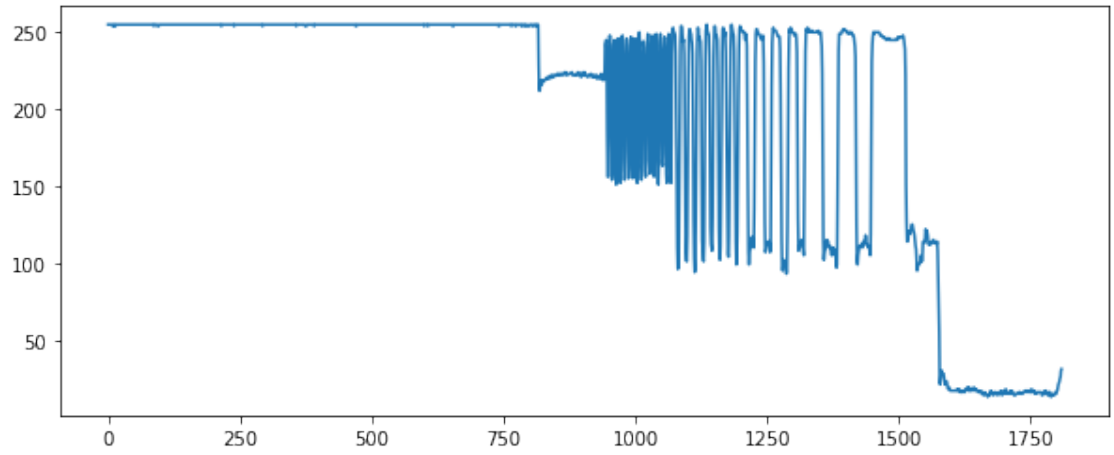
Debajo se realizará un corte en la imagen con el objetivo de ver que ocurre únicamente en el patrón en cuanto a intensidad de pixel. Una vez obtenida esa respuesta, se hará foco en la parte que interesa, donde la frecuencia es más alta.

[96]:
```
# generar algunos trazados para ver qué hay en esa línea
plt.figure(figsize=(10,4))
plt.plot(mtf_leido_gray[fila_mtf,:])

zoom = 1000
plt.figure(figsize=(10,4))
plt.plot(mtf_leido_gray[fila_mtf, columnas_mtf])

plt.figure(figsize=(10,4))
zoom = 100
plt.plot(mtf_leido_gray[fila_mtf, columnas_mtf])
```

[96]:
```
[<matplotlib.lines.Line2D at 0x88e0cbb9c8>]
```


250
200
150
100
50
0
0
1000
2000
3000
4000

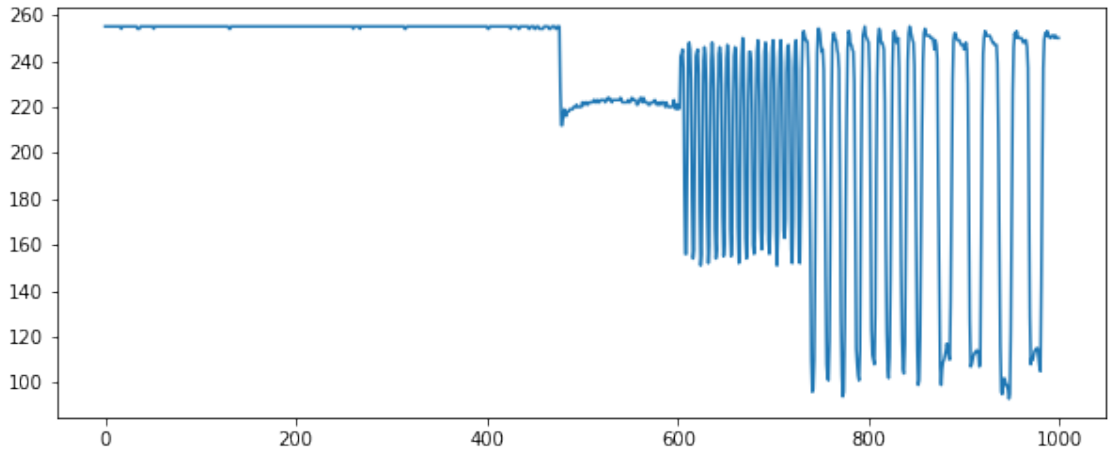

250
200
150
100
50
0
250
500
750
1000
1250
1500
1750


250
200
150
100
50
0
250
500
750
1000
1250
1500
1750

[97]:
```
# refinamos el corte
zoom = 1000
fila_mtf = 1580

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

[98]:
```
# dibujamos con más detalle el corte

y_est = mtf_leido_gray[fila_mtf, columnas_mtf]

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

[98]:
```
[<matplotlib.lines.Line2D at 0x88e0c5eb48>]
```



[99]:
```
# Generamos gráfico para leer donde cortar para recuperar la estimación de la y␣
 ↪de arriba

# por ahí si no quedó perfectamente centrado necesitan otro valor diferente de +/
 ↪- 35:
claro_est = mtf_leido_gray[fila_mtf-90, columnas_mtf]
oscuro_est = mtf_leido_gray[fila_mtf+90, columnas_mtf]

%matplotlib notebook
plt.figure(figsize=(10,4))
plt.plot(y_est)
plt.plot(claro_est)
plt.plot(oscuro_est)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

[99]:
```
[<matplotlib.lines.Line2D at 0x88df765fc8>]
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

[100]:
```
# Leer del gráfico de arriba

k_start =  480
k_end = 1000

y_est_r = y_est[k_start:k_end]

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

```
<IPython.core.display.Javascript object>
```

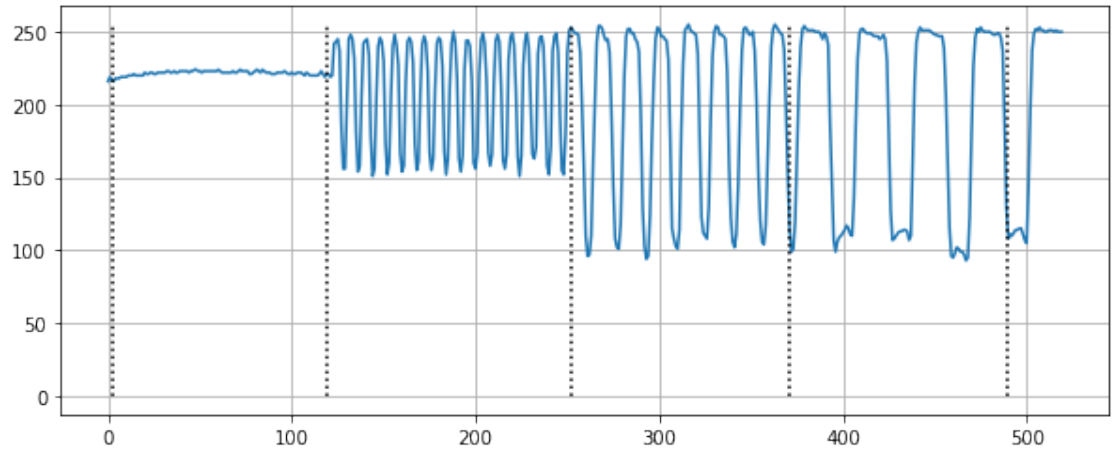
```
<IPython.core.display.HTML object>
```

[100]:
```
[<matplotlib.lines.Line2D at 0x88e0baad48>]
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

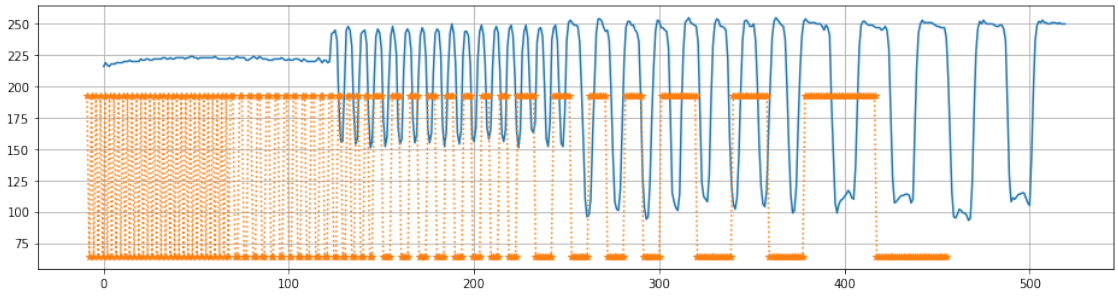
[107]:
```
# llenar con bordes medidos del gráfico:
bordes= [3, 119,252,371,490]

%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')
```



0.1 4. Extraer MTF

[102]:
```
# 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 = -9
offset_r = 455

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

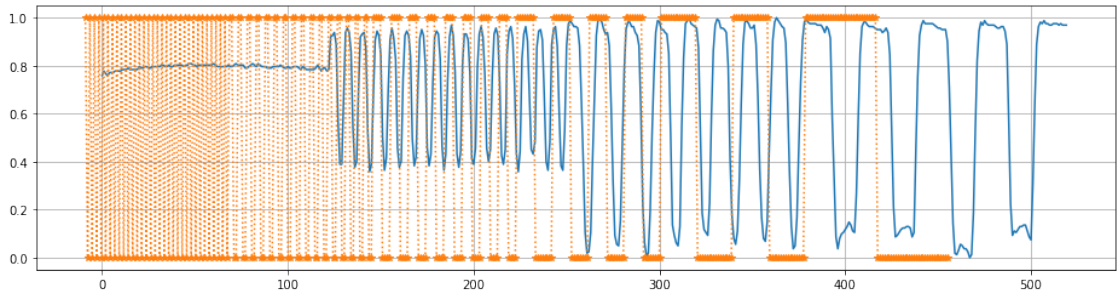
[102]:
```
[<matplotlib.lines.Line2D at 0x88d9633888>]
```



[103]:
```
# 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)), ':*')
```

[103]:
```
[<matplotlib.lines.Line2D at 0x88e107ee48>]
```



En el gráfico se observa y compara la respuesta ideal con la obtenida.

[104]:
```
# Sacar fotos y tomar nota de la geometría de cómo tomaron la foto en particular␣
 ↪distancia al monitor
# y ángulo subtendido por zona usada para calcular mtf:
distancia_monitor_mm = 380
ancho_zona_mtf_mm = 1

# Calcular ángulo que genera la zona usada para medir mtf (cateto menor en la␣
 ↪pantalla
# 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)
```

[105]:
```
# Calculamos grados por pixel para usar en el gráfico
deg_per_px = ancho_zona_mtf_px/angulo_zona_mtf_deg
print(deg_per_px)
```

```
3448.778563077067
```

$ MTF_{Local} = $\frac{i_{max} - i_{min}}{i_{max} + i_{min}}$$

[109]:
```
# 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');
```

```
0.0068965517
0.018181818
0.24875621
0.46131805
0.46397695
0.41340783
```