In [1]:
```python
# import libraries
import numpy as np
from matplotlib import pyplot as plt
import cv2

% matplotlib inline
```

In [2]:
```python
# function to display greyscale image
def show(img):
    plt.imshow(img, cmap='gray')
    plt.xticks([])
    plt.yticks([])
```

In [3]:
```python
# importing google drive library andusing the mount function to access drive
from google.colab import drive
drive.mount ('/gdrive')
```

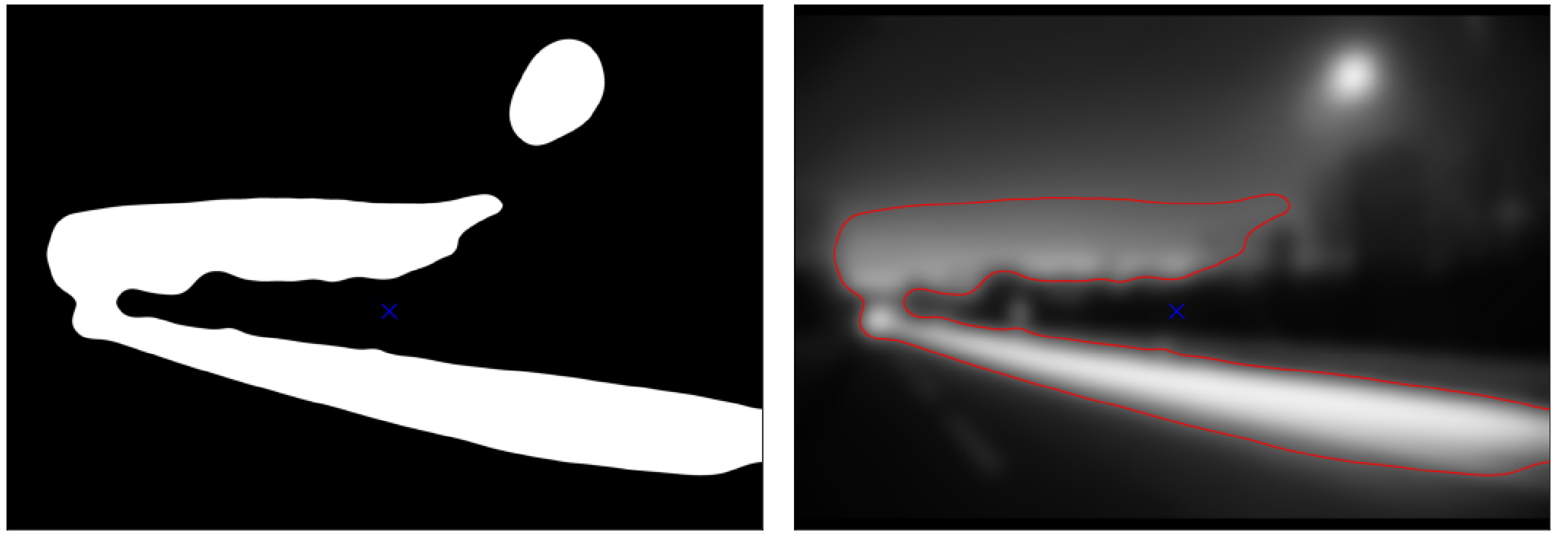
```
Mounted at /gdrive
```

In [4]:
```python
# importing google drive library andusing the mount function to access drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

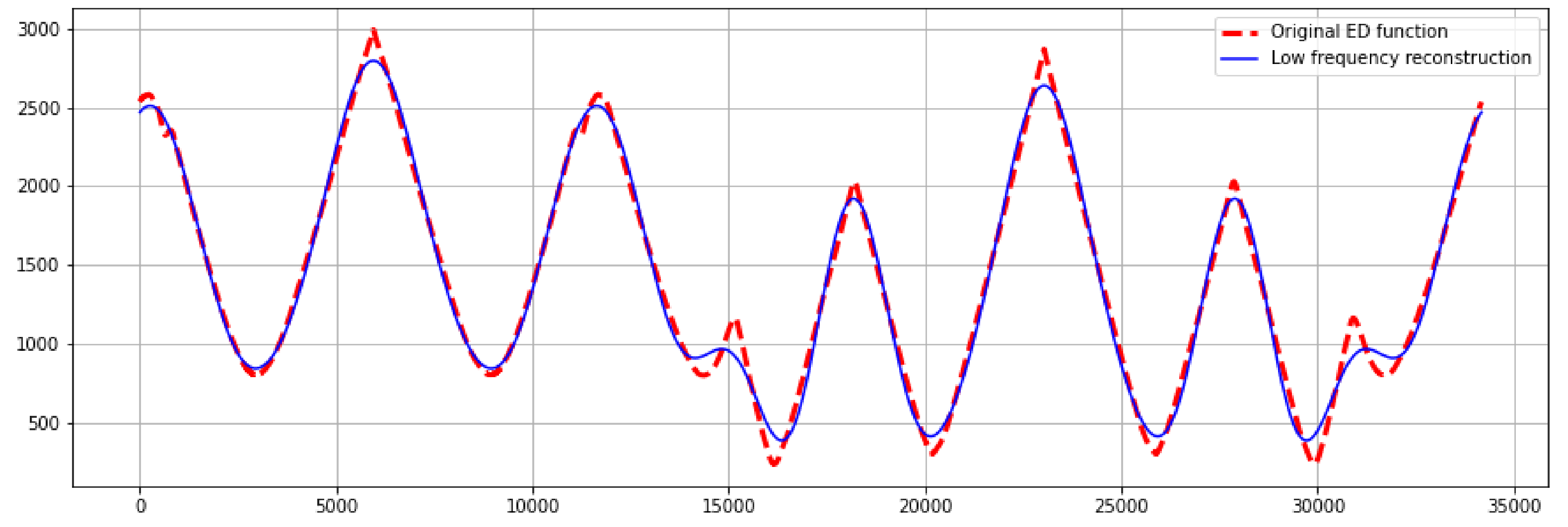
In [5]:
```python
img_original = cv2.imread('/content/drive/MyDrive/Road-Street-Blur-Image.jpg', 0)   # import
ing image
h, w = img_original.shape   # assigning the image shape as height and width

img = np.zeros((h+160,w), np.uint8)   # assigning image shape and defining the data type np.
uint8
img[80:-80,:] = img_original    # assigning the loaded image to the defined dimensions [80:-
80,:]

plt.figure(figsize=(15,5))    # defining diplay figure size on the x and y axis
plt.subplot(131)    # assigning loaded image to subplot row 1, column 1 of 3
show(img)   # display image

blur = cv2.medianBlur(img,(99)) # applying the medianBlur filter to the loaded image.
plt.subplot(132)    # assigning loaded image to subplot row 1, column 2 of 3
show(blur)  # display image

_, th = cv2.threshold(blur,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)   # applying a binary pi
xel threshold for pixel distinction
plt.subplot(133)    # assigning loaded image to subplot row 1, column 3 of 3
show(th)    # display image
plt.tight_layout()  # adjust display distance between images
plt.show()    # diplay all images defined in subplot
```

In [6]:
```
M = cv2.moments(th)   # creating moments of our data
h, w = img.shape    # assigning the image shape as height and width

x_c = M['m10'] // M['m00']    # asigning the modular result of the moments to x
y_c = M['m01'] // M['m00']    # asigning the modular result of the moments to y

plt.figure(figsize=(15,5))    # defining diplay figure size on the x and y axis
plt.subplot(121)    # assigning loaded image to subplot row 1, column 1 of 2
show(th)   # display image

plt.plot(x_c, y_c, 'bx', markersize=10)   # plotting x_c and y_c points with a blue marker o
f the x symbol.

kernel = np.array([[0, 1, 0],
                   [1, 1, 1],
                   [0, 1, 0]]).astype(np.uint8)   # creating a 3x3 kernel and assigning a ui
nt8 data type

erosion = cv2.erode(th,kernel,iterations=1)   # computing the minimal value of our threshold
image with a 3x3 kernel to enhance edge definition
boundary = th - erosion   # assigning the boundary values to the variable boundary

cnt, _ = cv2.findContours(boundary, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)   # using the find
Contours function to mark the continoous edges in the boundary variable
img_c = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)   # converting the color space of original ima
ge from RGB to GREYSCALE
cnt = cnt[0]    # replacing the cnt array with element 0 of the cnt in memory
img_cnt = cv2.drawContours(img_c, [cnt], 0, (255,0,0), 9)   # drawing the continoues edges p
reviously specified in the cnt variable to our image.

plt.subplot(122)    # assigning loaded image to subplot row 1, column 2 of 2
plt.plot(x_c, y_c, 'bx', markersize=10)   # plotting x_c and y_c points with a blue marker o
f the x symbol.
show(img_cnt)   # display image

plt.tight_layout(0)   # adjust display distance between images
plt.show()    # diplay all images defined in subplot

cnt = cnt.reshape(-1,2)   # reshaping cnt to a 2 dimentional array
left_id = np.argmin(cnt.sum(-1))    # getting the minimum result from element totals of the
 cnt array.
cnt = np.concatenate([cnt[left_id:,:], cnt[:left_id,:]])    # concatenating the left_id into
the cnt reshaped array.
```



In [7]:
```
dist_c = np.sqrt(np.square(cnt-[x_c, y_c]).sum(-1))   # creating a distance of our boundary.
f = np.fft.rfft(dist_c)   # culculating the distance
cutoff = 15   # assining a cutoff value of 15
```

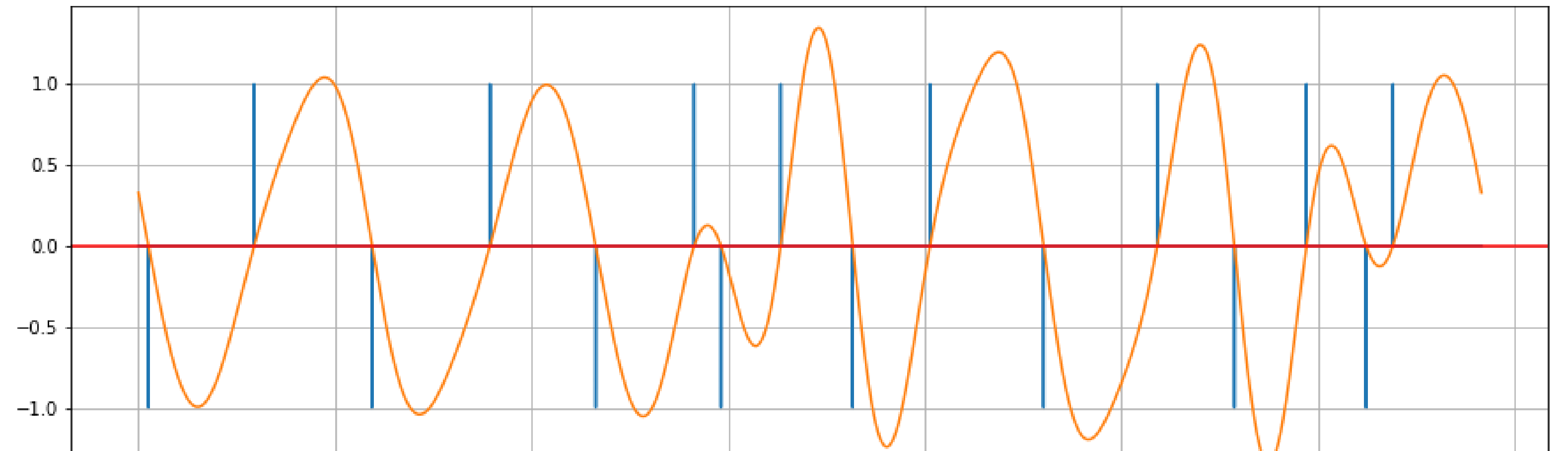
```
f_new = np.concatenate([f[:cutoff],0*f[cutoff:]])   # creating f_new by applying the cutoff
 value
dist_c_1 = np.fft.irfft(f_new)    # culculating the distance

plt.figure(figsize=(15,5))    # defining diplay figure size on the x and y axis
plt.grid()    # applying a grid reference to our plot
plt.plot(dist_c, label='Original ED function', color='r', linewidth='3', linestyle='--')
# ploting values of dist_c
plt.plot(dist_c_1, label='Low frequency reconstruction', color='b', linestyle='-')    # plot
ing values of dist_c_1
plt.legend()    # applying a legend in our plot
plt.show()
```



In [8]:
```
eta = np.square(np.abs(f_new)).sum()/np.square(np.abs(f)).sum()   # squaring f_new and devid
ing it by f
print('Power Retained: {:.4f}{}'.format(eta*100,'%'))
```

```
Power Retained: 99.8891%
```

In [9]:
```
derivative = np.diff(dist_c_1)    # culculating the differrences of dist_c_1
sign_change = np.diff(np.sign(derivative))/2    # culculating the differences of derivative
 and deviding the result by 2

#
plt.figure(figsize=(15,5))
plt.plot(sign_change)
plt.plot(derivative)
plt.axhline(y=0, color='r')
plt.grid()
plt.show()
```

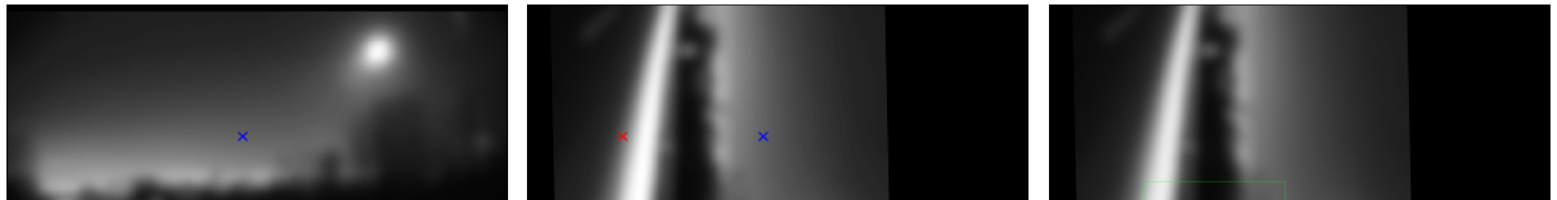


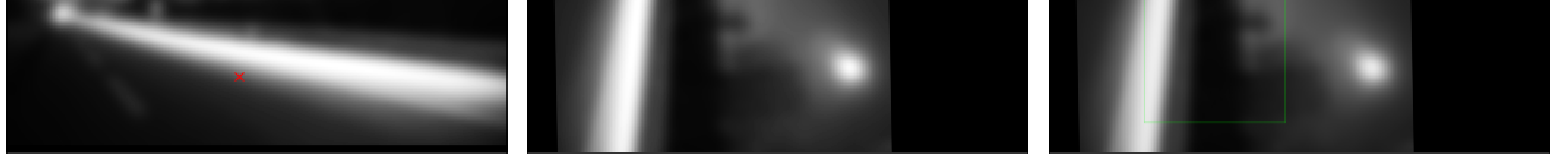
In [10]:

```
minimas = cnt[np.where(sign_change > 0)[0]]   # creating a condition to add elements when co
ndition is met.
v1, v2 = minimas[0], minimas[2]   # assigning the first and third element values of minimas
 to v1, v2

plt.figure(figsize=(15,5))
plt.subplot(131)
show(img)

plt.plot(v1[0], v1[1],'rx')   # ploting all elements of v1 with a red x
plt.plot(v2[0], v2[1],'bx')   # ploting all elements of v2 with a blue x
plt.subplot(132)

theta = np.arctan2((v2-v1)[1], (v2-v1)[0])*180/np.pi    # using arctan2 for determining the
 best value
print('The rotation of ROI is {:.02f}\u00b0'.format(theta))

R = cv2.getRotationMatrix2D(tuple(v2),theta,1)    # culculating the image rotation
img_r = cv2.warpAffine(img,R,(w,h))   # using the warpAffine to change the image state
v1 = (R[:,:2] @ v1 + R[:,-1]).astype(np.int)    # creating new v1 by appending R
v2 = (R[:,:2] @ v2 + R[:,-1]).astype(np.int)    # creating new v1 by appending R

plt.plot(v1[0], v1[1],'rx')   # ploting all elements of v1 with a red x
plt.plot(v2[0], v2[1],'bx')   # ploting all elements of v2 with a blue x
show(img_r)

ux = v1[0]    # assigning v1[0] value to ux
uy = v1[1] + (v2-v1)[0]//3    # assigning (v1[1] plus the (v2-v1)[0]//3)  value to uy
lx = v2[0]    # assigning v2[0] value to lx
ly = v2[1] + 4*(v2-v1)[0]//3    # assigning (v1[1] plus the 4*(v2-v1)[0]//3)  value to ly

img_c = cv2.cvtColor(img_r, cv2.COLOR_GRAY2BGR)   # converting image to COLOR_GRAY2BGR
cv2.rectangle(img_c, (lx,ly),(ux,uy),(0,255,0),2)   # drawing a bordering rectange of our im
ages
plt.subplot(133)
show(img_c)

plt.tight_layout()
plt.show()
```

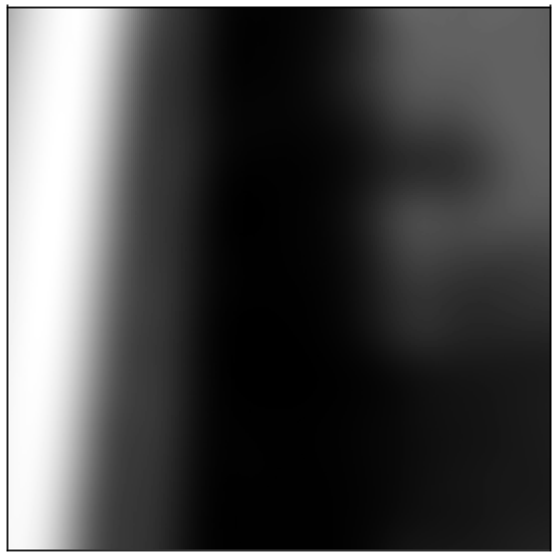
```
The rotation of ROI is -88.77°
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:17: DeprecationWarning: `np.int`
is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Do
ing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to u
se e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your current
use, check the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.
20.0-notes.html#deprecations
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:18: DeprecationWarning: `np.int`
is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Do
ing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to u
se e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your current
use, check the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.
20.0-notes.html#deprecations
```

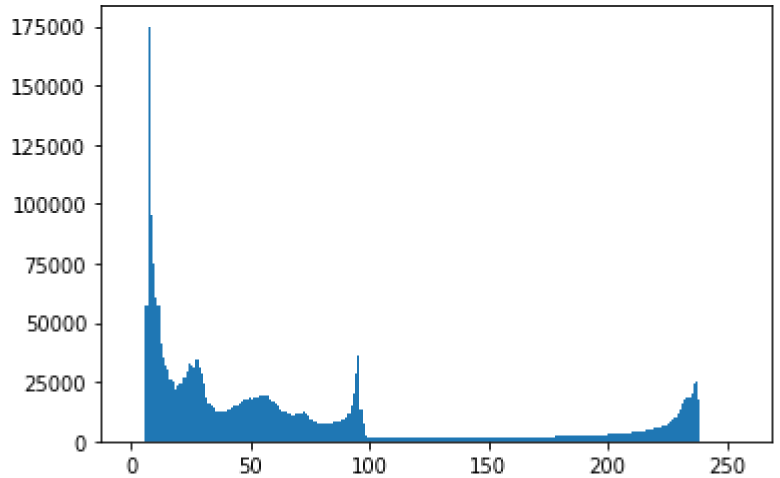




In [11]:
```
roi = img_r[uy:ly,ux:lx]     # creating an image from the above green highlighted matrix
plt.figure(figsize=(5,5))
show(roi)   # displaying image
```



In [12]:
```
plt.hist(roi.ravel(),256,[0,256]); plt.show()   # Flattening array to produce a linear displ
ay
```



In [13]:
```
from PIL import Image   # import Image library from PIL
```

In [14]:
```
from google.colab.patches import cv2_imshow   # importing cv2_imshow from google.colab.patch
es
```

In [15]:
```
!pip install sewar
```

```
Requirement already satisfied: sewar in /usr/local/lib/python3.7/dist-packages (0.4.5)
Requirement already satisfied: Pillow in /usr/local/lib/python3.7/dist-packages (from sewar)
(7.1.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from sewar)
(1.21.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from sewar)
(1.4.1)
```

In [16]:
```
from sewar import full_ref    # importing libraries
from skimage import measure, metrics    # importing libraries
```

GAUSSIAN FILTER

EVALUATION

3x3 kernel

In [17]:
```
orig = cv2.imread('/content/drive/MyDrive/Road-Street-Blur-Image.jpg', 1)
```

In [18]:
```
gaussian_kernel = np.ones((3,3),np.float32)/25    # defining parameters of a gaussian kernel
```

In [19]:
```
conv_gaussian = cv2.filter2D(orig, -1, gaussian_kernel, borderType = cv2.BORDER_CONSTANT)
# convolving our imported image though our gaussian kernel previously defined.

plt.figure(figsize=(10,5))
show(conv_gaussian)
```

In [20]:
```
gaussian = conv_gaussian
```

In [21]:
```
rmse_skimg = metrics.normalized_root_mse(orig, gaussian)    # culculating the random mean sc
ore error between the original image and the filtered one
print("RMSE: based on scikit-image = ", rmse_skimg)
```

```
RMSE: based on scikit-image =  0.6400470851141961
```

In [22]:
```
mse_skimg = metrics.mean_squared_error(orig, gaussian)    # culculating the mean score error
between the original image and the filtered one
print("MSE: based on scikit-image = ", mse_skimg)
```

```
MSE: based on scikit-image =  2764.658229694894
```

In [23]:
```
psnr_skimg = metrics.peak_signal_noise_ratio(orig, gaussian, data_range=None)   # culculatin
g the peak signal-to-noise ratio between the original image and the filtered one
print("PSNR: based on scikit-image = ", psnr_skimg)
```

```
PSNR: based on scikit-image =  13.714389099041151
```

In [24]:
```
from skimage.metrics import structural_similarity as ssim
ssim_skimg = ssim(orig, gaussian, data_range = img.max() - img.min(), multichannel = True)
# culculating the structural similarity index meassure of the original image and the filtere
d one
print("SSIM: based on scikit-image = ", ssim_skimg)
```

```
SSIM: based on scikit-image =  0.6470805596987184
```

5x5 kernel

```
In [25]: orig = cv2.imread('/content/drive/MyDrive/Road-Street-Blur-Image.jpg', 1)
```

```
In [26]: gaussian_kernel = np.ones((5,5),np.float32)/25    # defining parameters of a gaussian kernel
```

```
In [27]: conv_gaussian = cv2.filter2D(orig, -1, gaussian_kernel, borderType = cv2.BORDER_CONSTANT)
         # convolving our imported image though our gaussian kernel previously defined.

         plt.figure(figsize=(10,5))
         show(conv_gaussian)
```

```
In [28]: gaussian = conv_gaussian
```

```
In [29]: rmse_skimg = metrics.normalized_root_mse(orig, gaussian)    # culculating the random mean sc
         ore error between the original image and the filtered one
         print("RMSE: based on scikit-image = ", rmse_skimg)

RMSE: based on scikit-image =  0.007502946453236304
```

```
In [30]: mse_skimg = metrics.mean_squared_error(orig, gaussian)    # culculating the mean score error
         between the original image and the filtered one
         print("MSE: based on scikit-image = ", mse_skimg)

MSE: based on scikit-image =  0.3799105000415987
```

```
In [31]: psnr_skimg = metrics.peak_signal_noise_ratio(orig, gaussian, data_range=None)   # culculatin
         g the peak signal-to-noise ratio between the original image and the filtered one
         print("PSNR: based on scikit-image = ", psnr_skimg)

PSNR: based on scikit-image =  52.333990640304606
```

```
In [32]: from skimage.metrics import structural_similarity as ssim
         ssim_skimg = ssim(orig, gaussian, data_range = img.max() - img.min(), multichannel = True)
         # culculating the structural similarity index meassure of the original image and the filtere
         d one
         print("SSIM: based on scikit-image = ", ssim_skimg)

SSIM: based on scikit-image =  0.9971254693402405
```

7x7 kernel

In [33]:
```
orig = cv2.imread('/content/drive/MyDrive/Road-Street-Blur-Image.jpg', 1)
```

In [34]:
```
gaussian_kernel = np.ones((7,7),np.float32)/25    # defining parameters of a gaussian kernel
```

In [35]:
```
conv_gaussian = cv2.filter2D(orig, -1, gaussian_kernel, borderType = cv2.BORDER_CONSTANT)
# convolving our imported image though our gaussian kernel previously defined.

plt.figure(figsize=(10,5))
show(conv_gaussian)
```

In [36]:
```
gaussian = conv_gaussian
```

In [37]:
```
rmse_skimg = metrics.normalized_root_mse(orig, gaussian)    # culculating the random mean sc
ore error between the original image and the filtered one
print("RMSE: based on scikit-image = ", rmse_skimg)
```

```
RMSE: based on scikit-image =  0.7345632052533311
```

In [38]:
```
mse_skimg = metrics.mean_squared_error(orig, gaussian)    # culculating the mean score error
between the original image and the filtered one
print("MSE: based on scikit-image = ", mse_skimg)
```

```
MSE: based on scikit-image =  3641.463353838443
```

In [39]:
```
psnr_skimg = metrics.peak_signal_noise_ratio(orig, gaussian, data_range=None)   # culculatin
g the peak signal-to-noise ratio between the original image and the filtered one
print("PSNR: based on scikit-image = ", psnr_skimg)
```

```
PSNR: based on scikit-image =  12.518044171133425
```

In [40]:
```
from skimage.metrics import structural_similarity as ssim
ssim_skimg = ssim(orig, gaussian, data_range = img.max() - img.min(), multichannel = True)
# culculating the structural similarity index meassure of the original image and the filtere
d one
print("SSIM: based on scikit-image = ", ssim_skimg)
```

```
SSIM: based on scikit-image =  0.8182867008228009
```