

IMAGE GEOMETRY & MORPHOLOGY

FAIQA HADYA HANIFA
4521210058

MENGGAMBAR GEOMETRI



```
# Create a black image and load any image to draw 512x512 dimension and 3 channels color
img_geo = np.zeros((512,512,3), np.uint8)
```

Pertama, gunakan sebuah objek citra dengan dimensi 512x512 dan 3 saluran warna RGB untuk menjadi dasar menggambar bentuk-bentuk geometri.

```
# set blue channel
img_geo[:, :, 0] = 135 # Red channel
img_geo[:, :, 1] = 206 # Green channel
img_geo[:, :, 2] = 235 # Blue channel
```

Menghitung titik koordinat x dan y dari pusat gambar dengan mencari setengah dari lebar dan tinggi dari gambar dasar lalu menyimpannya ke dalam variabel `center_x` dan `center_y` untuk digunakan saat menggambar bentuk lain

Kemudian, mengatur tiga saluran warna tersebut untuk mengubah warna dasar menjadi warna biru.

```
# Calculate the center of the image
center_x = img_geo.shape[1] // 2
center_y = img_geo.shape[0] // 2
```

BASE BOX OUTLINE

```
# Outline the base box  
# Rectangle  
img_geo = cv2.rectangle(img_geo, (510,0), (0,510), (255, 105, 180), 15)
```

Menggambar sebuah persegi sebagai kontur kontak dasar menggunakan fungsi *rectangle* dari library *OpenCV* (cv2).

- (510,0) : posisi awal di titik x dan y
- (0,510) : posis akhir di titik x dan y
- (255, 105, 180) : intensitas saluran untuk warna merah muda
- 15 : ketebalan bentuk persegi

CIRCLE IN THE CENTER

```
# Circle: center coordinate and radius  
img_geo = cv2.circle(img_geo, (center_x, center_y), 63, (128,0,128), -2)  
cv2.circle(img_geo, (center_x, center_y), 70, (255, 105, 180), 3)
```

Menggambar lingkaran di tengah citra dasar dengan menggunakan *center_x* dan *center_y* sebagai sumbu posisi koordinatnya

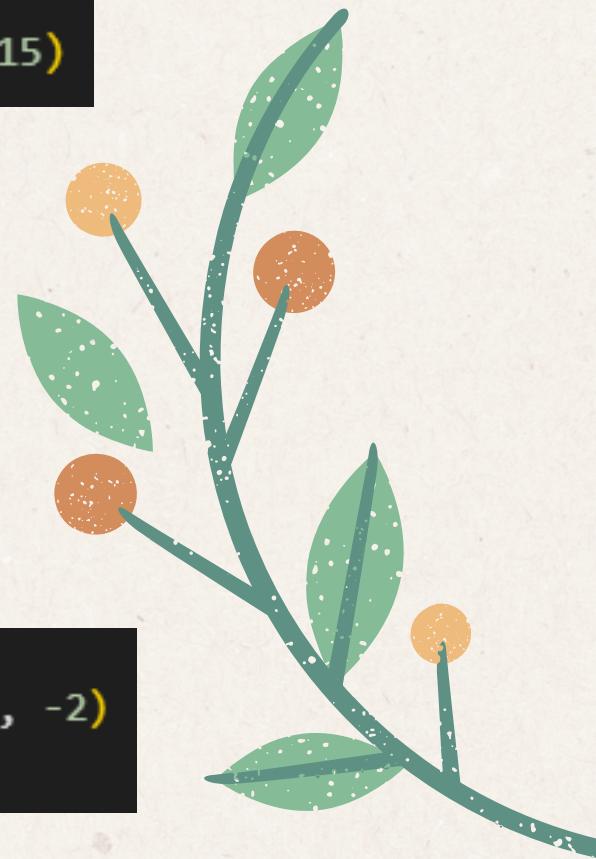
- Atur besar kontur lingkaran lebih besar dari gambar lingkaran dasar 63 dengan nilai 70
- (128,0,128) intensitas saluran untuk warna ungu

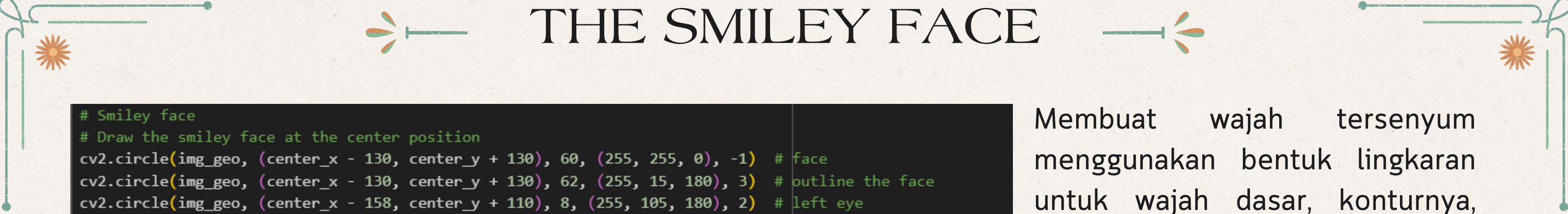
ARROW POINTING FROM THE CIRCLE

```
# Arrow  
img_geo = cv2.arrowedLine(img_geo, (center_x + 72, center_y), (480, center_y), (255, 105, 180), 8)
```

Membuat panah yang mengarah secara horizontal ke sisi ujung kanan dari tepi lingkaran

- (center_x + 72, center_y) : posisi awal mulai ekor panah
- (480, center_y) : posisi akhir ujung panah





THE SMILEY FACE

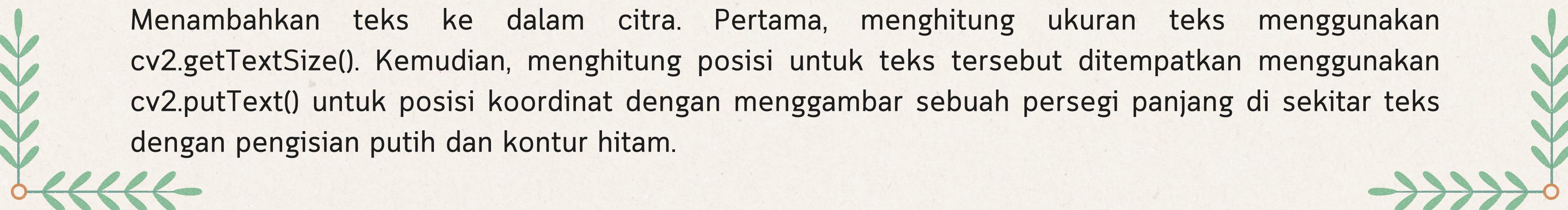
```
# Smiley face
# Draw the smiley face at the center position
cv2.circle(img_geo, (center_x - 130, center_y + 130), 60, (255, 255, 0), -1) # face
cv2.circle(img_geo, (center_x - 130, center_y + 130), 62, (255, 15, 180), 3) # outline the face
cv2.circle(img_geo, (center_x - 158, center_y + 110), 8, (255, 105, 180), 2) # left eye
cv2.circle(img_geo, (center_x - 101, center_y + 110), 8, (255, 105, 180), 2) # right eye
cv2.ellipse(img_geo, (center_x - 130, center_y + 135), (35, 30), 5, 5, 170, (255, 105, 180), 2) # mouth
```

Membuat wajah tersenyum menggunakan bentuk lingkaran untuk wajah dasar, konturnya, kedua mata, dan bentuk elips untuk mulut. Masing-masing bentuk memiliki titik koordinat masing-masing dengan intensitas ketebalan yang berbeda-beda untuk memposisikannya pada pojok kiri pada gambar dasar.



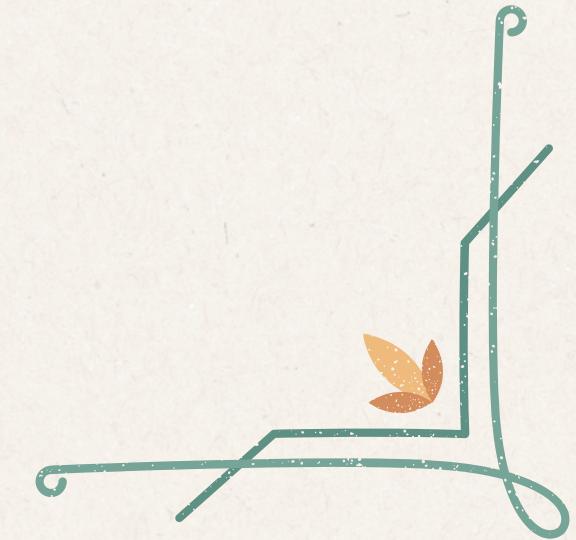
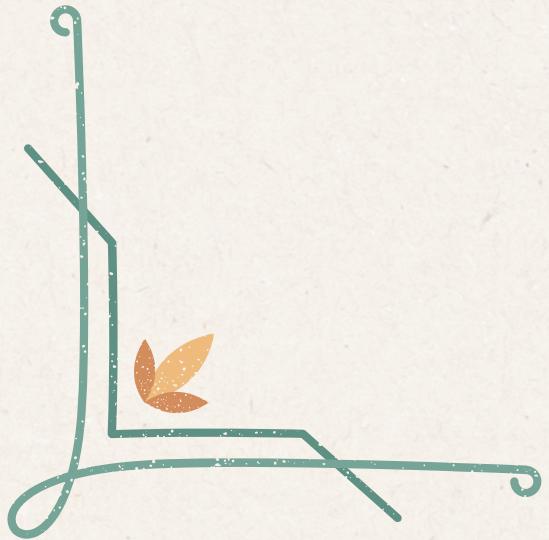
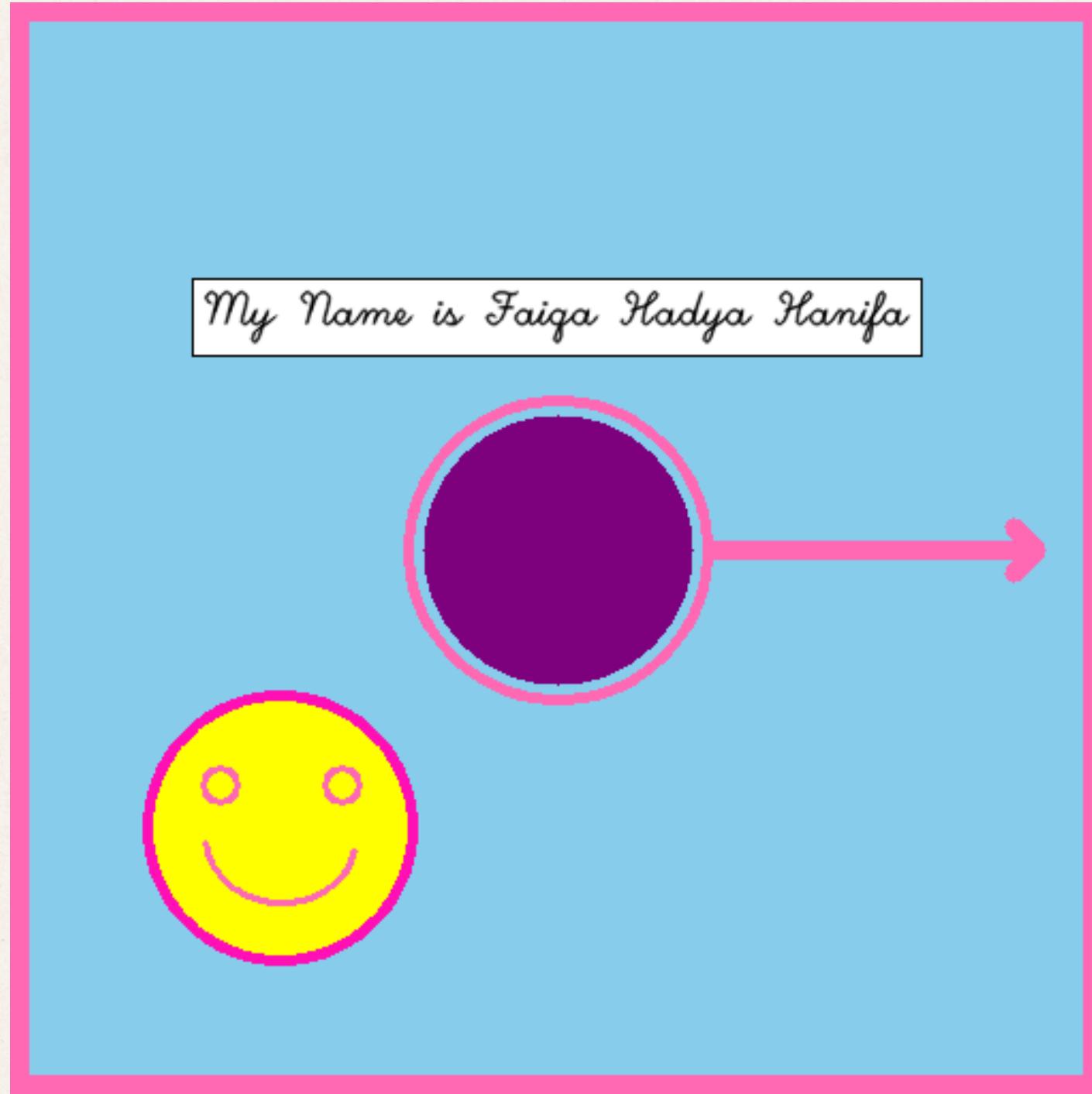
NAME TEXT BOX

```
# Box Text
font = cv2.FONT_HERSHEY_SCRIPT_SIMPLEX
text = 'My Name is Faiqa Hadya Hanifa'
text_size = cv2.getTextSize(text, font, 0.7, 1)[0]
text_width, text_height = text_size[0], text_size[1]
# Calculate the position of the text
text_x = (img_geo.shape[1] - text_width) // 2
text_y = 150
# Draw a rectangle around the text
cv2.rectangle(img_geo, (text_x - 5, text_y - text_height - 5), (text_x + text_width + 5, text_y + 15), (255, 255, 255), -1) # white rectangle
cv2.rectangle(img_geo, (text_x - 5, text_y - text_height - 5), (text_x + text_width + 5, text_y + 15), (0, 0, 0), 1) # black outline
# put text
cv2.putText(img_geo, text, (text_x, text_y), font, 0.7, (0,0,0), 1, cv2.LINE_AA)
```



Menambahkan teks ke dalam citra. Pertama, menghitung ukuran teks menggunakan `cv2.getTextSize()`. Kemudian, menghitung posisi untuk teks tersebut ditempatkan menggunakan `cv2.putText()` untuk posisi koordinat dengan menggambar sebuah persegi panjang di sekitar teks dengan pengisian putih dan kontur hitam.

← HASIL AKHIR →



FILTER BLUR

```
# (a.1) Filter Blurring Kernel 5

# define kernel and normalization
kernel_5 = np.ones((5,5), np.float32) / 25
# apply convolution
img_kernel_5 = cv2.filter2D(img, -1, kernel_5)
# concatenate image horizontally
list_img_5 = np.hstack([img, img_kernel_5])
# display the concatenated image
cv2.imshow(list_img_5)
```

2D convolution pada *image filtering* dilakukan dengan cara *low pass filter* untuk *image blurring* dan mengurangi *noise*. *Image convolution* menggunakan fungsi cv2.filter2D mengatur kernel berupa matriks untuk melakukan filtering. Akan diterapkan kernel 5, 7, dan 10 pada gambar burung. Fungsi ini membutuhkan tiga argumen: gambar masukan, kedalaman gambar keluaran (-1 disini akan memberikan keluaran gambar yang sama dengan gambar masukan), dan kernel yang digunakan, dimana disini disimpan dalam variabel kernel_5 untuk mengembalikan matriks 5x5. Fungsi np.ones() akan membuat sebuah array. Melalui script ini ukuran kernel dapat diubah-ubah sesuai kebutuhan pemrosesan gambar.

```
# (a.1) Filter Blurring Kernel 5  
  
# define kernel and normalization  
kernel_5 = np.ones((5,5), np.float32) / 25
```

```
# (a.2) Filter Blurring Kernel 7  
  
# define kernel and normalization  
kernel_7 = np.ones((7,7), np.float32) / 50
```

```
# (a.3) Filter Blurring Kernel 10  
# define kernel and normalization  
kernel_10 = np.ones((10,10), np.float32) / 80
```



Namun, untuk setiap meningkatnya ukuran kernel, nilai normalisasi perlu diatur untuk mencapai intensitas yang sesuai dan menghindari kehilangan fitur-fitur pada gambar. Dapat dilihat bahwa angka pembagi semakin besar jika ukuran kernel meningkat.

Perbedaan utama penggunaan kernel 5, 7, dan 10 terletak pada ukuran dan pola bobot yang diterapkan pada pixel citra selama proses *blurring*. Dimana dapat dilihat dari gambar-gambar disamping, semakin besar ukuran kernel, semakin besar area yang dipertimbangkan sehingga menghasilkan pemburaman yang lebih kuat dan merata, juga mengaburkan detail yang lebih halus. Intensitas pemburaman dan penghalusan akan lebih terlihat ketika menggunakan gambar *grayscale* yang telah dikonversi menjadi saluran warna tunggal hitam dan putih.

KERNEL 5



KERNEL 7



KERNEL 10



DIRECTIONAL BLURRING

```
# horizontal
kernel_hor = np.array([[0,0,0], [1,1,1], [0,0,0]], np.float32) / 3.0
```



```
# horizontal
kernel_hor = np.array([[0,1,-1,0], [1,3,-3,1], [1,3,-3,1], [0,1,-1,0]], np.float32) / 4
```



Langkah lain dari penghalusan dan pemburaman dari fungsi cv2.filter2D adalah menggunakan *directional blurring* atau pemburaman secara horizontal dengan mengatur ukuran matriks yang diinginkan sebagai filter untuk gambar masukan. Cara ini akan mereduksi detail dalam arah tertentu. Sebelumnya, digunakan matriks 3x3 dan ukuran kernel ([0,0,0], [1,1,1], [0,0,0]) pada gambar burung. Walaupun, perbedaan tidak terlalu terlihat dengan kernel tersebut, sedikit penghalusan terjadi pada bagian tepi dari objek pada gambar. Jika, digunakan matriks 4x4 dan kernel yang berbeda; ([0,1,-1,0], [1,3,-3,1], [1,3,-3,1], [0,1,-1,0]) pada gambar lain yang memiliki lebih banyak *noise*, dapat dilihat pemburaman dan penghalusan hasil filter pada gambar sebelah kanan dibandingkan dengan gambar masukan asli.



AVERAGE BLUR () DENGAN KERNEL 5, 7, DAN 10

```
## BLUR
# Create a figure to contain the subplots
fig = plt.figure(figsize=(12, 8))

# Display the original image
ax = fig.add_subplot(2, 2, 1)
ax.set_title("Original Image")
plt.imshow(img_bgr)

kernelSizes = [(5, 5), (7, 7), (10, 10)]

# Loop over the kernel sizes and display the blurred images
for i, (kX, kY) in enumerate(kernelSizes):
    # Apply an "average" blur to the image using the current kernel size
    blurred = cv2.blur(img_bgr, (kX, kY))

    # Add a subplot for the blurred image
    ax = fig.add_subplot(2, 2, i + 2)
    ax.set_title("Blurred Image ({}, {})".format(kX, kY))
    plt.imshow(blurred)

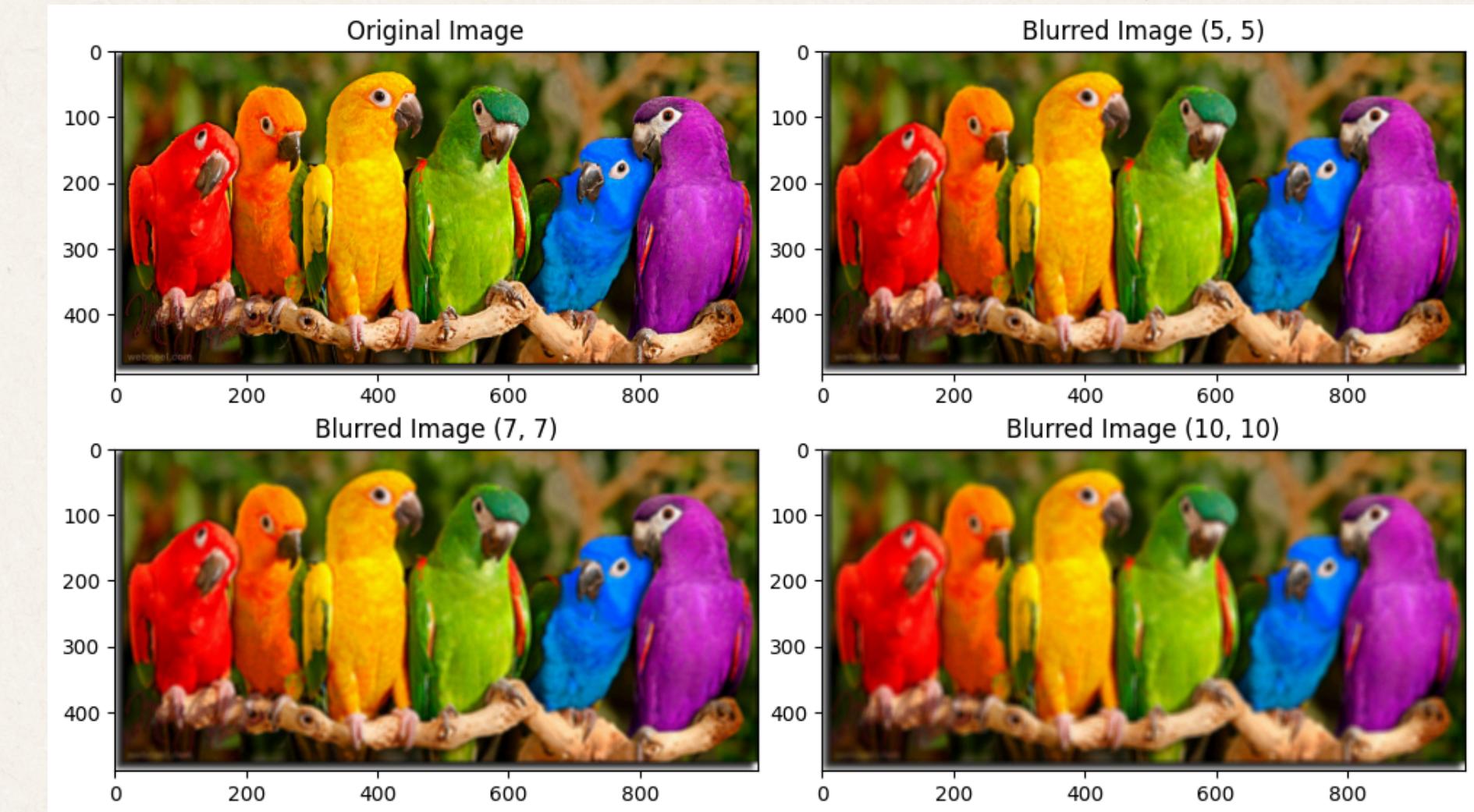
# Adjust layout and display the plot
plt.subplots_adjust(wspace=0.1, hspace=-0.2)
plt.show()
```

Fungsi `cv2.blur` melakukan *average blurring* dengan mengambil rata-rata dari area sekitar pixel pusat dan mengganti dengan nilai rata-rata tersebut. Proses ini mengurangi *noise* dan tingkat detail dari suatu citra. Untuk menggunakan fungsi ini, gambar akan digabungkan dengan filter yang ternormalisasi dengan sebuah kernel $m \times n$. Fungsi `cv2.blur` membutuhkan dua argumen: gambar masukan dan ukuran kernel. Dalam hal ini, diaplikasikan kernel 5x5, 7x7, dan 10x10 dan didefinisikan dalam daftar `kernelSizes`.

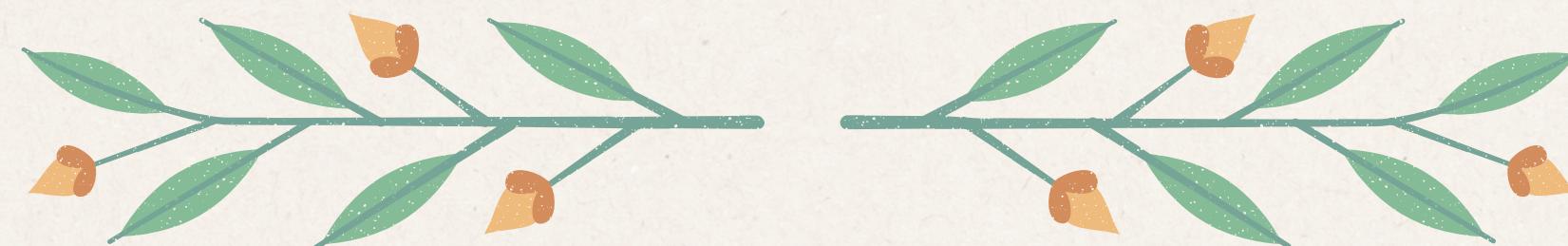


AVERAGE BLUR DENGAN KERNEL 5, 7, DAN 10

Seperti gambar hasil *running script* sebelumnya disamping ini, gambar terkaburkan dan dengan bertambahnya ukuran kernel citra akan semakin buram. Dapat dibandingkan dengan gambar masukan asli di kiri atas. Gambar menjadi sedikit lebih buram dan detail pada bagian bulu burung-burung



dan tangkai pohon mengalami penghalusan saat menggunakan kernel 5x5 dan 7x7. Namun, saat ukuran kernel mencapai 10x10 citra menjadi buram secara signifikan. Ketika menggunakan fungsi *average blurring*, setiap pixel dalam area kernel akan mendapatkan bobot yang sama dan sangat mudah untuk terjadi *overblurring* dan kehilangan bagian penting pada gambar.



GAUSSIAN BLUR DENGAN KERNEL 5, 7, 15

Berbeda dengan fungsi sebelumnya, cv2.GaussianBlur tidak dapat menerima ukuran kernel dengan angka genap. Fungsi ini menggunakan *weighted mean* dengan pixel tetangga yang lebih dekat ke pixel pusat menyumbang lebih banyak bobot pada rata-rata. Gaussian digunakan untuk menghilangkan *noise* mengikuti distribusi gaussian.

Menggunakan fungsi *cv2.GaussianBlur* ini membutuhkan tiga argumen: gambar masukan, ukuran kernel yang disimpan dalam *kernelSizes*, dan parameter simpangan baku. Dengan mengatur nilai tersebut menjadi 0, standar deviasi akan dihitung secara otomatis berdasarkan ukuran kernel.

```
# GAUSS BLUR
fig = plt.figure(figsize=(12, 8))
fig.suptitle("Gaussian Blur Comparison", fontsize=20)

# Display the original image
ax = fig.add_subplot(2, 2, 1)
ax.set_title("Original Image")
plt.imshow(img_bgr)

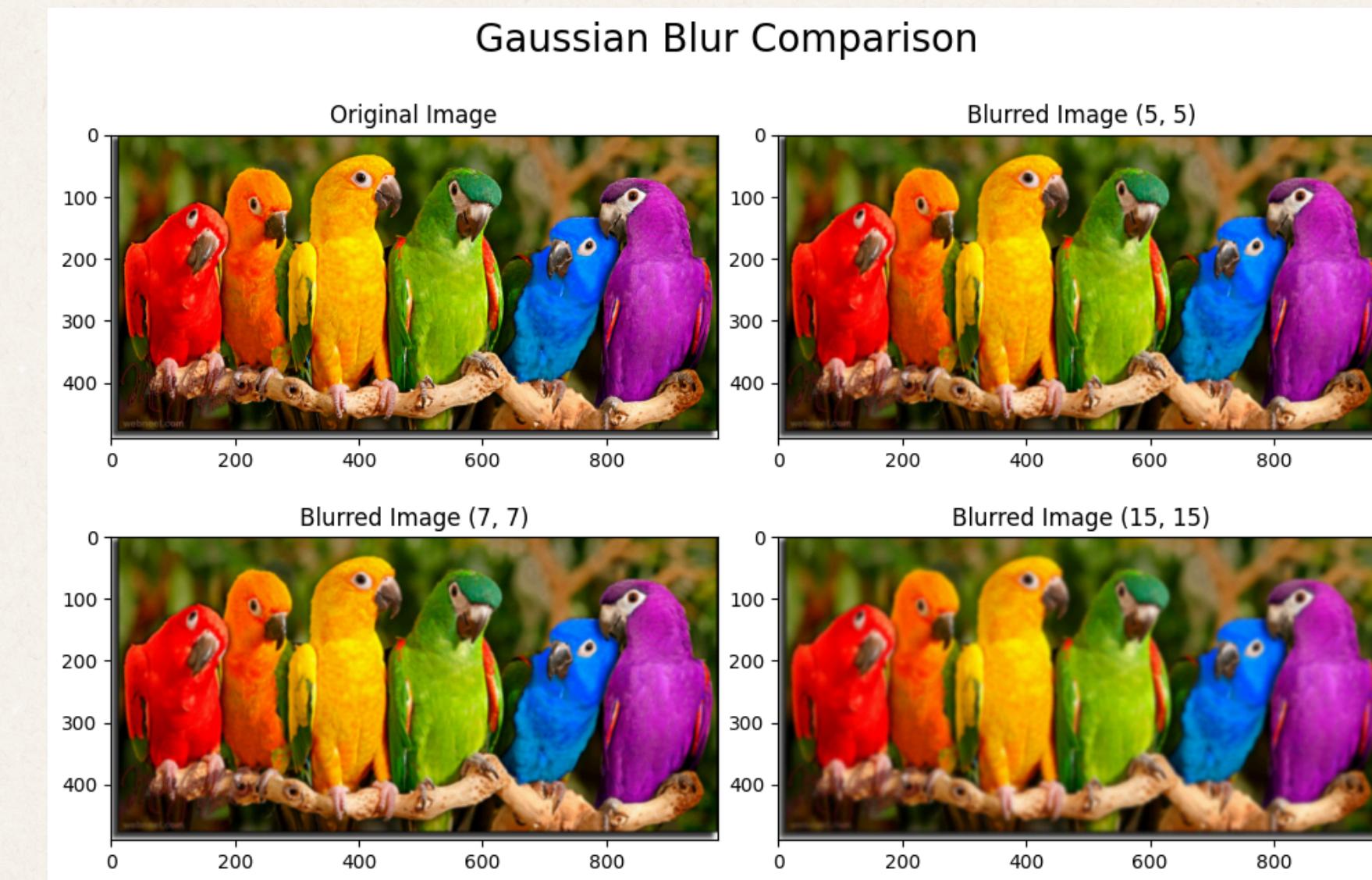
kernelSizes = [(5, 5), (7, 7), (15, 15)]

# Loop over the kernel sizes and display the blurred images
for i, (kX, kY) in enumerate(kernelSizes):
    # Apply Gaussian blur to the image using the current kernel size
    img_gaussblur = cv2.GaussianBlur(img_bgr, (kX, kY), 0)

    # Add a subplot for the blurred image
    ax = fig.add_subplot(2, 2, i + 2)
    ax.set_title("Blurred Image ({}, {})".format(kX, kY))
    plt.imshow(img_gaussblur)

# Adjust layout and display the plot
plt.subplots_adjust(wspace=0.1, hspace=-0.3, top=1)
plt.show()
```

GAUSSIAN BLUR DENGAN KERNEL 5, 7, 15



- * Hasil akhir adalah gambar yang tidak terlalu buram dan lebih buram secara natural dibandingkan menggunakan *average blur* sebelumnya dan dapat mempertahankan lebih banyak bagian tepi pada gambar karena perhitungan rata-rata yang tertimbang. Gaussian blur memberikan hasil pemburaman yang lebih bagus.

ACCESS THE FULL NOTEBOOK :

<https://colab.research.google.com/drive/1cxNxAZQscEpzoUMblQpRizFR741aaz8y?usp=sharing>

