|  |
| --- |
| **import** **cv2** **import** **numpy** **as** **np** |

Meng-*import* *library* dasar yang dibutuhkan untuk membaca, menampilkan, dan menyimpan gambar dari *Python3*.

|  |
| --- |
| source = cv2.imread('FACE DETECTION.png') |

Membaca gambar yang akan diproses dengan fungsi dasar di *OpenCV*  yaitu *imread*.

|  |
| --- |
| source\_hsv = rgb2hsv(source) |

Memanggil fungsi rgb2hsv yang telah didefinisikan sebelumnya, kemudian memasukkan hasilnya ke variabel *source\_hsv*.

|  |
| --- |
| **def** **rgb2hsv**(source):  row, col, ignore = source.shape  dest = np.zeros((row, col, **3**), np.uint8) |

Mendefinisikan fungsi *rgb2hsv* yang akan digunakan untuk mengubah citra *rgb* ke *hsv*. Pada awalnya variabel *row, col,* dan *ignore* diisi dengan nilai *return* dari fungsi *shape* pada *OpenCV*. *row* diisi dengan baris*, col* diisi dengan kolom*,* sedangkan ignore akan mendapatkan nilai 3 yaitu nilai *Blue, Green,* dan *Red.* Nilai ini bisa di abaikan, karena yang dibutuhkan adalah nilai dari *RGB* langsung. Kemudian mendefinisikan kanvas dengan nama *dest*.

|  |
| --- |
| **for** i **in** range(row):  **for** j **in** range(col):  r\_ = source[i][j][**2**] / **255**  g\_ = source[i][j][**1**] / **255**  b\_ = source[i][j][**0**] / **255** |

Melakukan iterasi pada setiap pixel gambar kemudian mengubah setiap nilai dari *red, green,* dan *blue* ke *range* 0 sampai 1.

|  |
| --- |
| Cmax = max(r\_, g\_, b\_) Cmin = min(r\_, g\_, b\_) delta = Cmax - Cmin |

Mendefinisikan variabel *Cmax* (nilai maksimum dari *Red, Green,* dan *Blue* pada *pixel*) dan *Cmin* (nilai minimumnya). Kemudian mendefinisikan delta yaitu selisih antara nilai maksimum dan nilai minimum citra *RGB* pada suatu *pixel*.

|  |
| --- |
| h = **0** **if** delta == **0**:  h = **0** **elif** Cmax == r\_:  h = **60** \* (((g\_ - b\_) / delta) % **6**)  **elif** Cmax == g\_:  h = **60** \* (((b\_ - r\_) / delta) + **2**) **elif** Cmax == b\_:  h = **60** \* (((r\_ - g\_) / delta) + **4**) |

Menginisiasi variabel *h* dengan nilai 0, kemudian menerapkan kondisi untuk nilai *h* jika delta bernilai 0 maka h akan diisi dengan nilai 0. Selain itu, dicek nilai *Cmax* nya, dengan nilai tertentudidapatkan nilai h tertentu sesuai dengan rumus yang diberikan.

|  |
| --- |
| s = **0** **if** delta == **0** **else** delta / Cmax  v = Cmax |

Mengisi nilai *s* (*saturation*) dengan 0 jika *delta* bernilai 0. Jika tidak, *s* akan diisi dengan nilai *delta / Cmax. Nilai v (value)* diisi dengan nilai *Cmax.*

|  |
| --- |
| h /= **2**  v = v \* **255**  s = s \* **255**  dest.itemset((i, j, **0**), v)  dest.itemset((i, j, **1**), s)  dest.itemset((i, j, **2**), h) **return** dest |

Karena gambar yang digunakan berukuran 8-bit maka nilai *h* dibagi dengan 2*,* sedangkan *v* dan *s* dikali dengan 255. Kemudian nilai *h, s,* dan *v* dimasukkan ke dalam variabel *dest*. Setelah iterasi berakhir program akan mengembalikan kanvas yang telah diisi dengan nilai *hue, saturation,*  dan *value.*

|  |
| --- |
| result = detectFace(source\_hsv) |

Setelah citra gambar diubah ke dalam hsv, dilakukan pendeteksian wajah menggunakan fungsi *detectFace* yang telah didefinisikan terlebih dahulu. Hasil dimasukkan ke variabel *result*.

|  |
| --- |
| **def** **detectFace**(source):  row, col, ignore = source.shape  dest = np.zeros((row, col, **3**), np.uint8) |

Mendefinisikan fungsi *detectFace* yang akan digunakan untuk mendeteksi wajah pada gambar masukan. Pada awalnya variabel *row, col,* dan *ignore* diisi dengan nilai *return* dari fungsi *shape* pada *OpenCV*. *row* diisi dengan baris*, col* diisi dengan kolom*,* sedangkan ignore akan mendapatkan nilai 3 yaitu nilai *Value, Saturation,* dan *Hue.* Nilai ini bisa di abaikan, karena yang dibutuhkan adalah nilai dari *HSV* langsung. Kemudian mendefinisikan kanvas dengan nama *dest*.

|  |
| --- |
| **for** i **in** range(row):  **for** j **in** range(col):  v\_ = source[i][j][**0**]  s\_ = source[i][j][**1**]  h\_ = source[i][j][**2**] |

Melakukan iterasi pada setiap pixel kemudian mendapatkan tiap nilai dari *value, saturation,* dan *hue.*

|  |
| --- |
| **if** h\_ <= **20** **and** s\_ >= **5**:  dest.itemset((i, j, **0**), v\_)  dest.itemset((i, j, **1**), s\_)  dest.itemset((i, j, **2**), h\_)  **return** dest |

Memasukkan nilai pixel yang memiliki *hue* dibawah 20 *dan saturation* di atas 5. Kenapa memilih *hue* dengan nilai 20, karena saya mengikuti *rule for skin classification* yang menyebutkan bahwa *hue* dengan nilai antara 19 < *hue <* 240 bukan merupakan *hue* dari kulit manusia. Tetapi program ini memberi *tolerance* menjadi 20karena adanya noise pada gambar yang kurang tajam. Selain itu, batas nilai *saturation* juga ditambahkan karena adanya *pixel* dengan *hue* dibawah 20 tetapi bukan kulit manusia. Kemudian setelah iterasi selesai kanvas *dest* dikembalikan nilainya.

|  |
| --- |
| result = hsv2rgb(result) |

Setelah mendapatkan kanvas yang hanya berisi citra dari wajah saja, Kemudian ditambahkan juga fungsi untuk mengubah kembali citra *hsv* ke *rgb* agar didapatkan hasil sesuai contoh. Fungsi telah didefinisikan sebelumnya.

|  |
| --- |
| **def** **hsv2rgb**(source):  row, col, ignore = source.shape  dest = np.zeros((row, col, **3**), np.uint8) |

Mendefinisikan fungsi *hsv2rgb* yang akan digunakan untuk mengubah citra *hsv* ke *rgb*. Pada awalnya variabel *row, col,* dan *ignore* diisi dengan nilai *return* dari fungsi *shape* pada *OpenCV*. *row* diisi dengan baris*, col* diisi dengan kolom*,* sedangkan ignore akan mendapatkan nilai 3 yaitu nilai *Value, Saturation,* dan *Hue.* Nilai ini bisa di abaikan, karena yang dibutuhkan adalah nilai dari *HSV* langsung. Kemudian mendefinisikan kanvas dengan nama *dest*.

|  |
| --- |
| **for** i **in** range(row):  **for** j **in** range(col):  v\_ = source[i][j][**0**]/**255**  s\_ = source[i][j][**1**]/**255**  h\_ = source[i][j][**2**]\***2** |

Melakukan iterasi pada setiap pixel, kemudian memasukkan nilai *HSV* yang telah di normalisasi ke *range* sebelum diubah ke 8 bit.

|  |
| --- |
| c = (v\_) \* (s\_) x = c \* (**1** - abs((h\_ / **60**) % **2** - **1**)) m = v\_ - c |

Mendapatkan nilai c yang merupakan hasil dari *value* dikalikan *saturation*. Kemudian mendapatkan nilai x dan m dengan hasil sesuai rumus pengubahan *HSV* ke *RGB*.

|  |
| --- |
| r\_, g\_, b\_ = **0**, **0**, **0** **if** h\_ < **60**:  r\_, g\_, b\_ = c, x, **0** **elif** h\_ < **120**:  r\_, g\_, b\_ = x, c, **0** **elif** h\_ < **180**:  r\_, g\_, b\_ = **0**, c, x **elif** h\_ < **240**:  r\_, g\_, b\_ = **0**, x, c **elif** h\_ < **300**:  r\_, g\_, b\_ = x, **0**, c **elif** h\_ < **360**:  r\_, g\_, b\_ = c, **0**, x  r, g, b = (r\_ + m) \* **255**, (g\_ + m) \* **255**, (b\_ + m) \* **255** |

Mendapatkan *r’, g’,* dan *b’* sesuai dengan kondisi yang diberikan rumus *HSVtoRGB*.

|  |
| --- |
| dest.itemset((i, j, **0**), b)  dest.itemset((i, j, **1**), g)  dest.itemset((i, j, **2**), r)  **return** dest |

Memasukkan nilai hasil proses ke kanvas, kemudian mengembalikan nilainya ke pemanggil fungsi.

|  |
| --- |
| cv2.imwrite('hasildeteksi.png', result) cv2.waitKey() cv2.destroyAllWindows() |

Menyimpan gambar menggunakan fungsi *imwrite* dari *OpenCV* kemudian memanggil fungsi *waitKey()* dan *destroyAllWindows()* agar gambar dapat terbuka dan menunggu inputan untuk tertutup.

Referensi :

Human Face Detection System Using HSV, Iyad Aldasouqi (<https://pdfs.semanticscholar.org/6d96/132e5d2425855d5968ec71741945e47dc552.pdf>) diakses pada 18 Maret 2018.