Emprovador virtual: Aplicació per a canviar els colors de la roba en imatges

Santi Estrada i Fité

Resum— Aquest projecte utilitza una de les tasques més comunes en el processament d'imatges, l'alteració de color, per a crear una aplicació que permeti modificar el color de qualsevol peça de roba en una imatge, mantenint-ne les seves propietats d'il·luminació i textura originals. Per a aconseguir-ho ens servirem de l'entorn OpenCV, el qual fa possible el tractament d'imatges en temps real. L'usuari serà capaç de seleccionar qualsevol element de la imatge, podent tractar-se tant d'una peça de roba sencera com d'elements o parts que la constitueixin, i escollir qualsevol color que desitgi aplicar-hi. En el cas que es seleccioni més d'un element o es modifiqui el color d'algun d'ells, es comprovarà l'harmonia dels colors actuals d'aquests, tant si són els originals com si han estat modificats.

Paraules clau—OpenCV, processament d'imatges, segmentació, floodfill, transferència de color, harmonia de color.

Abstract—The aim of this project is to use one of the most common tasks in image processing, color alteration, in order to build an application which allows modifying the color of any piece of clothing in an image, maintaining the original lightning and texture. To achieve this we will make use of the OpenCV environment, which focuses on real-time image processing. The user will be able to select any element of the image, being either an item of clothing or distinct elements or parts of it, and pick the desired color to apply. In the case where more than one element is selected or one of their colors is modified, the harmony of their current colors will be checked, whether they are the originals or they have been modified.

Index Terms—OpenCV, image processing, segmentation, floodfill, color transfer, color harmony.

1 Introducció

L'ALTERACIÓ de color és una de les tècniques més practicades en el tractament d'imatges, generalment dedicada a la correcció de color per tal d'eliminar tonalitats no desitjades en una imatge o millorar-ne les propietats d'il·luminació. Aquestes pràctiques poden estar involucrades tant en el món de la fotografia, adreçades a l'augment de qualitat i retocs d'imatges, com en el món del cinema, en tasques com el post-processament de pel·lícules.

Un dels mètodes més peculiars per a dur a terme un tipus de correcció de color més general es basa en aplicar la distribució de colors d'una imatge origen a una altra imatge objectiu, el que es coneix com a transferència de color. En aquest projecte s'utilitzarà aquesta tècnica per a modificar el color de parts concretes i diferenciades d'una imatge, concretament peces de roba. D'aquesta manera, aconseguirem tenir una aplicació que permeti observar en temps real com quedarien les peces de roba en diferents colors i decidir quins d'ells són els més adequats per a cada imatge, el que vindria a ser un emprovador virtual.

- *E-mail de contacte: santiago.estrada@e-campus.uab.cat*

- *Menció realitzada: Computació*
- *Treball tutoritzat per: Robert Benavente i Vidal (Computació)*
- *Curs 2014/15*

2 Estat de l'art

Actualment existeixen varis algorismes de transferència de color, els quals poden tenir en compte diferents paràmetres estadístics de la imatge relacionats amb la distribució dels colors. Un exemple n'és l'algorisme proposat per Reinhard et al. [1], on es té en compte la mitjana i desviació estàndard dels píxels per a aconseguir mapejar la distribució de colors d'una imatge font a una imatge objectiu.

Per altra banda, existeix un algorisme que, a partir d'un píxel origen de la imatge, busca píxels veïns de color semblant i els modifica de color, però no conserva la il·luminació original d'aquests píxels. Tot i així, pot servir com a algorisme de segmentació d'elements de la imatge. Aquest algorisme s'anomena *floodfill* [2], i ha estat utilitzat en projectes com el proposat per Samarth Brahmbhatt [3] per a detectar diferents objectes en una imatge a partir d'un sol clic sobre un punt interior d'aquests.

Finalment, es poden trobar mètodes per a millorar l'harmonia de colors presents en una imatge, per tal de fer-la més visualment atractiva. Un d'ells es tracta de l'algorisme proposat per D. Cohen-Or et al. [8], el qual determina l'harmonia de color d'una imatge determinada a partir d'una sèrie de regles i models i corregir la distribució de colors en el cas que no sigui harmònica.

3 OBJECTIUS

L'objectiu principal del projecte consisteix en aconseguir una aplicació que permeti modificar el color d'una o vàries peces de roba o parts d'aquestes seleccionades per l'usuari en una imatge. D'aquest objectiu principal se'n deriven quatre objectius més, els quals corresponen als quatre grans blocs en els que es divideix el projecte:

- Implementar un mètode de segmentació que permeti seleccionar un element de la imatge a través d'un únic clic sobre un punt interior d'aquest.
- Implementar un mètode de transferència de color que permeti modificar el color de l'element seleccionat.
- Comprovar l'harmonia dels colors dels elements seleccionats i/o modificats entre ells, informant si la combinació de colors és harmònica o no.
- Dissenyar una interfície gràfica amigable i intuïtiva a ulls de l'usuari que ofereixi les funcionalitats i possibilitats adequades.

4 METODOLOGIA

En aquest projecte s'utilitzarà la tècnica de transferència de color per a modificar el color de parts concretes i diferenciades d'una imatge, concretament peces de roba, però en lloc d'obtenir la distribució de colors d'una imatge origen utilitzarem un sol color, el que es desitgi aplicar, i la imatge objectiu serà únicament una part de la imatge original, corresponent a la peça de vestir o part d'aquesta que s'hagi seleccionat. Per tant, en primer lloc necessitarem trobar un mètode de segmentació d'imatges per a obtenir la part de roba determinada i diferenciar-la de la resta d'elements de la imatge. Un cop es tingui l'element de la imatge seleccionat i un color desitjat, ja serà possible aplicar l'algorisme de transferència de color i substituir el color actual de l'element pel nou color escollit, mantenint la textura i la il·luminació originals d'aquest, com poden ser arrugues, ombres, zones amb més lluminositat, etc.

OpenCV és una gran llibreria centrada principalment en el processament d'imatges en temps real, de manera que ens serà de gran ajuda al llarg de tot el procés, tant a l'hora de segmentar la imatge com al moment d'aplicar la transferència de color.

Per tal de dur a terme l'explicació de la metodologia que s'ha seguit per a aconseguir els objectius del projecte la dividirem en els quatre blocs ens els que es divideix aquest: segmentació, transferència de color, comprovació de l'harmonia de color i disseny de la interfície.

4.1 Segmentació

En primer lloc, necessitem trobar un algorisme de segmentació que ens permeti obtenir la màscara de la peça o part de roba de la imatge a partir d'un sol punt d'aquesta. Doncs bé, l'algorisme *floodfill* [2] que ens proporciona la llibreria d'OpenCV [4], fa possible exactament el que necessitem. Aquest algorisme determina els píxels de la imatge que són similiars a un píxel origen o llavor definit, i estan connectats a aquest, i els pinta d'un mateix color. Aquesta relació de semblança es defineix a través del valors RGB (o d'escala de grisos) dels píxels tenint en compte un llindar especificat com a paràmetre. És a dir, la funció obté el valor RGB del píxel origen i el compara amb els seus píxels veïns i veïns d'aquests progressivament per a determinar si són similars i pertanyen a la mateixa regió connectada de la imatge. La definició de la funció i els dels seus paràmetres la podem trobar a l'apèndix.

En el nostre cas, els paràmetres que més ens interessen són els que determinen el llindar de semblança entre píxels. Inicialment es van establir uns valors fixes, però es va veure que no funcionaven correctament per a totes les imatges, ja que cada imatge té una distribució de colors diferent i necessita uns llindars específics. D'aquesta manera, es va decidir que aquests dos paràmetres serien ajustables per l'usuari a través de la interfície de l'aplicació per a trobar els valors adequats per a cada imatge, com proposa Samarth Brahmbhatt [3].

Per altra banda, el píxel llavor es fixarà a través d'un sol clic sobre la imatge original. A més, ens interessa que els dos flags disponibles de la funció estiguin activats, el primer (*FLOODFILL_FIXED_RANGE*) perquè volem comparar cada píxel respecte la llavor que hem fixat, i el segon (*FLOODFILL_MASK_ONLY*) perquè només ens interessa obtenir la màscara de la peça de roba segmentada per posteriorment utilitzar-la per modificar-ne el color, i no fer-ho a través de la pròpia funció, ja que d'aquesta manera no es mantindria la il·luminació i textura de la roba.

La funció retorna un valor enter, equivalent al nombre de píxels considerats de la mateixa regió, però no ens serà de cap ajuda i no l'utilitzarem.

4.1.1 Post-procés

Un cop vistos els resultats que oferia aquest algorisme, es va veure que presentava una sèrie de problemes:

El primer d'ells es dóna quan la peça de roba conté lletres o altres tipus d'estampats que creen espais buits (forats) corresponents a parts de la mateixa peça de roba. Aquestes regions no són segmentades ja que no formen part de píxels veïns connectats al píxel d'origen, a no ser que es seleccioni un punt origen per cada un d'ells. Per exemple, si es vol segmentar una samarreta que tingui una lletra 'O' d'un color diferent al d'aquesta, l'espai interior de la lletra no es considerarà.

El segon problema es va trobar al treballar amb imatges on la peça de roba contenia diferents elements del mateix color però que no es trobaven connectats, com per exemple samarretes amb ratlles alternades de dos colors o amb lletres del mateix color però que es trobaven separades. Per tal de segmentar aquests elements, hauríem de

seleccionar-los per separat (un punt origen per cada un d'ells), però seria interessant trobar un mètode que ens permetés segmentar-los tots a la vegada al seleccionar-ne un de sol.

L'últim defecte que es va identificar va ser que, en alguns casos, es segmentaven parts de pell de la imatge per semblança de color amb la part de roba seleccionada.

Per tal de solucionar aquests tres problemes es van introduir tres noves funcionalitats, no plantejades inicialment, a utilitzar de forma opcional (quan sigui necessari) un cop aplicada la funció *floodfill*.

La primera consisteix en reomplir els possibles espais buits (forats) de la peça de roba del mateix color que aquesta. La segona opció, la qual s'ha anomenat "color global", permet incloure elements de la imatge del mateix color que l'element seleccionat i segmentat inicialment que no es trobin connectats a aquest. La última funcionalitat consisteix en un senzill detector de pell per a eliminar de la màscara les parts de pell que s'hagin pogut confondre amb la roba.

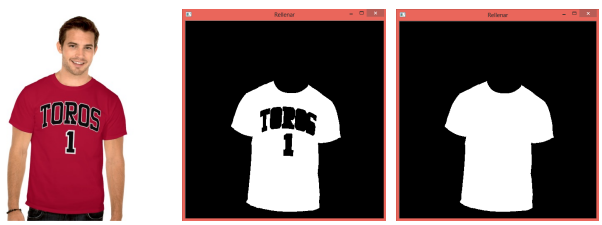
Aquestes tres opcions podran ser activades o desactivades a través de la interfície de l'aplicació, com veurem més endavant. A continuació n'explicarem el seu funcionament més detalladament.

4.1.1.1 Reomplir buits

Per a la primera opció, la de reomplir els forats de l'element del mateix color que aquest, en primer lloc necessitem que la màscara passi a ser de la peça de roba sencera, incloent les lletres o dibuixos que pugui contenir. Per tal d'aconseguir-ho, fem ús de les funcions *findContours* i *drawContours* [4] de la llibreria d'OpenCV. Les definicions d'aquestes funcions també les podem trobar a l'apèndix.

Primer necessitem trobar els contorns externs de la peça de roba. Per a fer-ho, utilitzem la funció *findContours*, on la imatge d'entrada serà la màscara obtinguda després d'aplicar l'algorisme *floodfill*.

Un cop tenim els contorns externs de la peça emmagatzemats, procedim a omplir-ne el seu interior a través de la funció *drawContours* [4]. Aquesta funció permet dibuixar els contorns trobats mitjançant la funció anterior, però si s'especifica un paràmetre determinat podem pintar l'espai interior delimitat per aquests contorns. Aquest paràmetre és el que determina la grossor dels contorns que es volen dibuixar, però si es defineix com un valor negatiu el que fa és exactament el que necessitem, omplir tot l'interior dels contorns.



(a) (b) (c)

Figura 1: Opció de reomplir forats. (a) Imatge original. (b) Màscara obtinguda després d'aplicar l'algorisme *floodfill*. (c) Màscara obtinguda després d'aplicar les funcions *findContours* i *drawContours*.

Com podem veure a la Figura 1(c), aconseguim la màscara de la samarreta sencera, incloent-ne tot el seu interior, mentre que amb l'algorisme *floodfill* no podríem obtenir la màscara dels forats de les lletres, com veiem a la Figura 1(b), excepte si es seleccionessin individualment (un punt origen per a cada forat).

Doncs bé, ara que ja tenim la màscara de la samarreta sencera, passem a aplicar la funció d'OpenCV anomenada *inRange* [4] per a trobar elements interns de la samarreta amb color semblant al d'aquesta i crear la màscara definitiva. Aquesta funció simplement agafa una imatge d'entrada i n'extreu una màscara amb els píxels que es troben dins d'un rang de valors que se li especifica. La definició de la funció és la següent:

```
void inRange(InputArray src, InputArray lowerb, In-
        putArray upperb, OutputArray dst)
```

- **src** – imatge d'entrada.
- **lowerb** – límit inferior del valor del píxels.
- **upperb** – límit superior del valor dels píxels.
- **dst** – imatge de sortida de la mateixa mida que la d'entrada i de tipus CV_8U (binària).

La imatge d'entrada serà només la part de la imatge original corresponent a la màscara calculada per la funció *drawContours*, en aquest cas tota la samarreta, la qual primer convertim de l'espai RGB a l'espai HSV [5] (Hue, Saturation, Value) mitjançant la funció *cvtColor* [4] d'OpenCV, ja que així podrem diferenciar millor els colors establint un rang més precís a través dels paràmetres *lowerb* i *upperb*, obtenint millors resultats. Això és degut al fet que l'espai HSV, al contrari que l'espai RGB, separa el color (hue) de la intensitat dels píxels. Les operacions que es duen a terme a l'hora de realitzar la conversió d'aquests espais les podem trobar a [6].

Ara bé, la qüestió que sorgeix és com obtenir els límits que hauria de tenir aquest rang de valors. Per a calcular-los utilitzem la part de la imatge original corresponent a la màscara calculada inicialment per l'algorisme *floodfill*, sense les lletres ni els forats creats per aquestes (Figura 1(b)). Llavors la convertim també a l'espai HSV i en dividim els tres canals (h, s, v) a través de la funció *split* [4] d'OpenCV, la qual ens crea tres vectors, i per a cada un d'ells busquem quin és el valor mínim i màxim a través de la funció *minMaxLoc* [4], també d'OpenCV. D'aquesta manera, obtindrem els sis valors que necessitem (tres per l'escalar que indica el límit inferior del rang i tres més pel que indica el límit superior) per a buscar elements interiors de la samarreta i podrem aplicar la funció *inRange* de la següent forma:

```
inRange(src_hsv, Scalar(low_h, low_s, low_v), Sca-
        lar(high_h, high_s, high_v), mask)
```

En el nostre cas, després d'aplicar aquesta funció obtindrem la màscara de totes les parts de la samarreta que tinguin el color vermellós original, incloent els forats de les lletres, com podem veure a la Figura 2.

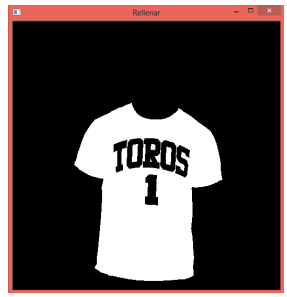
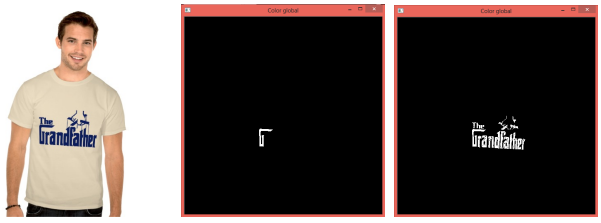


Figura 2: Màscara obtinguda mitjançant la funció *inRange* amb l'opció de reomplir forats.

És important recordar que la imatge d'entrada utilitzada per la funció *inRange* és la part de la imatge original corresponent a la màscara de la samarreta sencera (Figura 1(c)), però a l'hora de calcular els valors dels paràmetres *lowerb* i *upperb* s'utilitza la part de la imatge corresponent a la màscara calculada en primer lloc per l'algorisme *floodfill* (Figura 1(b)), ja que sinó es tindria també en compte els valors dels píxels pertanyents a les lletres a l'hora de calcular el valor mínim i màxim de cada canal.

4.1.1.2 Color global

Pel que fa a la segona opció introduïda, la del color global, simplement necessitem utilitzar la funció *inRange* però sobre tota la imatge original, també convertida a l'espai HSV, calculant el rang de valors a tenir en compte de la mateixa manera que abans. Per tant, no necessitem l'ajuda de les funcions *findContours* i *drawContours*.



(a) (b) (c)

Figura 3: Opció de color global. (a) Imatge original. (b) Màscara obtinguda després d'aplicar l'algorisme *floodfill*. (c) Màscara obtinguda amb la funció *inRange* amb l'opció de color global.

A la Figura 3(b) podem veure la màscara obtinguda després d'utilitzar l'algorisme *floodfill* utilitzant un punt de l'interior de la lletra 'G' com a origen. A l'activar aquesta opció, la funció *inRange* busca altres elements de la imatge de color semblant i els incorpora a la màscara, com veiem a la Figura 3(c), de manera que ens evita haver de seleccionar cada lletra per separat. També ens és especialment útil pel cas de samarretes amb ratlles de dos colors alternats, on simplement hauríem de seleccionar-ne una i automàticament es segmentaria la resta de ratlles del mateix color.

Ara bé, aquesta opció té un problema, i és que si hi haguessin altres elements a la imatge del mateix color que l'element seleccionat i no formessin part de la samarreta (fons, pantalons, cabell, etc.) o inclús altres elements de la samarreta que no es volguessin tenir en compte, també serien considerats, ja que la funció *inRange* s'aplica a tota la imatge, de manera que aquesta funcionalitat està limitada a determinades imatges.

4.1.1.3 Detector de pell

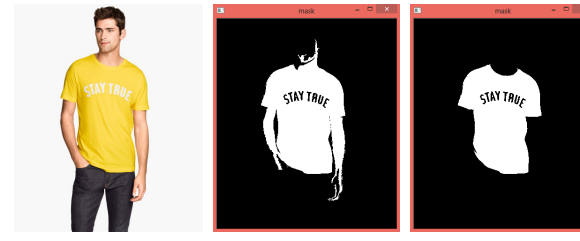
Per últim, la tercera funcionalitat incorporada a l'hora de crear la màscara de l'element seleccionat es tracta d'un detector de pell per tal d'eliminar zones de pell que s'hagin pogut escapar a l'aplicar l'algorisme *floodfill* per semblança de color amb la roba seleccionada. No es tracta d'un detector complex, però s'ha vist que funciona correctament per a la gran majoria d'imatges amb les que treballem.

El funcionament d'aquest detector es basa en utilitzar novament la funció *inRange*, convertint de la mateixa manera la imatge a l'espai HSV i utilitzant el rang de valors que es mostra a continuació:

```
inRange(src, Scalar(3, 48, 80), Scalar(15, 170, 255), dst)
```

Uns valors semblants es van trobar inicialment a [7], però després de vàries proves es va veure que no funcionaven del tot bé per a alguns pocs casos, de manera que es van modificar una mica per a poder-se adaptar a aquests casos de la millor manera possible, els quals es tractaven d'imatges amb colors vermellosos i taronges que es confonien amb alguns colors de pell depenent de la il·luminació. Aquesta és una limitació òbvia d'aquesta funcionalitat, i és que s'està tractant un problema de detecció de pell com un problema de detecció de color, de manera que els valors HSV definits com a límits del rang podran no servir sota diferents condicions d'il·luminació. Es va pensar en la possibilitat d'ajustar aquests valors a través de la interfície de l'aplicació, però com que funcionen adequadament per a la majoria de casos s'ha decidit no dur-ho a terme. Tot i així, en una aplicació real caldria implementar un detector de pell més robust que funcionés per a tots els casos. Una manera de fer-ho podria consistir en implementar un mecanisme de detecció de cares com l'algorisme *Viola-Jones* [11] per a determinar el color de la cara i posteriorment utilitzar aquest model per a detectar la resta de pell del cos.

Una vegada tenim la màscara de les parts de pell detectades, simplement la restem a la màscara actual i ja tenim la màscara final.



(a) (b) (c)

Figura 4: Detector de pell. (a) Imatge original. (b) Màscara obtinguda després d'aplicar l'algorisme *floodfill*. (c) Màscara obtinguda després d'aplicar el detector de pell.

A la Figura 4(b) podem veure com s'ha segmentat algunes parts del coll i els braços, així que apliquem el detector de pell i eliminem les zones no desitjades de la màscara, obtenint el que veiem a la Figura 4(c). En aquesta imatge el resultat es podria dir que és perfecte, però en l'apartat de resultats veurem casos on el detector de pell no es comporta tan bé.

4.2 Transferència de color

Un cop obtenim la màscara definitiva de l'element de la imatge seleccionat, ja podem aplicar la transmissió de color. L'algorisme que s'ha utilitzat per a tal finalitat es tracta del proposat per Reinhard et al. [1]. Aquest algorisme es basa en transferir la distribució de colors d'una imatge d'entrada a una altra de diferent, a través de l'ús de la mitjana i la desviació estàndard de les distribucions de les dues imatges. El mètode proposat però, fa ús de l'espai de color Lαβ, mentre que nosaltres utilitzarem el CIELAB, més estrictament anomenat CIE 1976 L*a*b [5], ja que els dos espais són molt semblants però ens serà més fàcil treballar amb el segon a l'hora de realitzar conversions entre espais. El motiu pel qual s'utilitza aquest espai de color és principalment perquè es tracta d'un espai perceptualment uniforme, el que implica que la distància euclidiana entre dos colors a l'espai es correlaciona amb la diferència perceptual o visual d'aquests, de manera que un canvi determinat en un valor de color, produeix un canvi aproximadament de la mateixa importància visual en aquest color. A l'espai RGB, la majoria dels píxels tenen un coeficient de correlació elevat entre les seves components, de manera que si volem canviar un píxel de manera coherent, hem de modificar els valors de cada canal al mateix temps. Per tant, el que volem és un espai de color ortogonal amb la mínima correlació possible entre els seus canals per a poder treballar amb cadascun d'ells per separat.

Així doncs, l'algorisme de Reinhard et al. es serveix dels següents passos per tal d'aconseguir transferir la distribució de colors d'una imatge a una altra:

1. Agafar una imatge origen i una imatge objectiu. La imatge origen conté la distribució de colors que volem transferir a la imatge objectiu.
2. Convertir les dues imatges a l'espai de color Lab (CIELAB).
3. Separar els tres canals L*, a* i b* per les dues imatges.
4. Calcular la mitjana i desviació estàndard de cadascun dels canals per les dues imatges.
5. Restar la mitjana dels canals de la imatge objectiu als valors d'aquests.

$$\begin{aligned} L &= L - \mu_t^L \\ a &= a - \mu_t^a \\ b &= b - \mu_t^b \end{aligned} \tag{1}$$

   on μ_t^c és la mitja del canal c de la imatge objectiu.
6. Multiplicar els canals de la imatge objectiu resultants per la divisió entre la desviació estàndard dels canals de la imatge objectiu i la desviació estàndard dels canals de la imatge origen.

$$\begin{aligned} L &= L * \frac{\sigma_t^L}{\sigma_s^L} \\ a &= a * \frac{\sigma_t^a}{\sigma_s^a} \\ b &= b * \frac{\sigma_t^b}{\sigma_s^b} \end{aligned} \tag{2}$$

   on σ_t^c és la desviació del canal c de la imatge objectiu i σ_s^c la desviació del canal c de la imatge origen.
7. Sumar-hi la mitjana dels canals de la imatge origen.

$$\begin{aligned} L &= L + \mu_s^L \\ a &= a + \mu_s^a \\ b &= b + \mu_s^b \end{aligned} \tag{3}$$

   on μ_s^cés la mitjana del canal c de la imatge origen.
8. Combinar els canals resultants.
9. Convertir la imatge resultant a l'espai RGB de nou.

En el nostre cas, però, no tenim una imatge d'origen, sinó un sol color que volem transferir a la imatge objectiu, la qual no és la imatge sencera sinó la part corresponent a l'element de roba seleccionat projectat sobre un fons negre. Aquesta part de la imatge la podem obtenir fàcilment a través de la màscara calculada anteriorment mitjançant la funció *copyTo* [4] d'OpenCV, la qual crea una còpia de la imatge d'entrada únicament de la part corresponent a la màscara que se li passi com a paràmetre (opcional). Per tant, al no tenir la distribució de colors d'una imatge d'entrada, la mitjana de cada canal d'origen passa a ser directament el valor corresponent del color que volem transferir en l'espai CIELAB, mentre que la desviació estàndard fem que sigui la mateixa que la de l'objectiu. D'aquesta manera, la divisió comentada en el pas 6 passa a valdre 1 i, per tant, al fer la multiplicació esmentada ens quedem amb els mateixos valors en els tres canals. Així, en el nostre cas, ignorem el pas 6 i només tenim en compte la mitjana (restem la de l'objectiu i sumem la d'origen, la qual hem quedat que equivalia al valor en l'espai CIELAB del color que volem transferir).

Per tal de convertir la imatge de l'espai RGB a l'espai CIELAB i viceversa utilitzem la funció d'OpenCV anomenada *cvtColor* [4] com hem fet abans amb l'espai HSV, especificant-li la imatge d'entrada i de sortida i el tipus de conversió, en aquest cas *CV_BGR2LAB* o *CV_LAB2BGR* (OpenCV utilitza els canals RGB en ordre invers, BGR). Les operacions que es duen a terme per tal de realitzar la conversió entre aquests espais les podem trobar a [6]. Aquestes operacions s'han implementat en funcions específiques per tal de convertir el valor RGB del color que volem aplicar a la roba seleccionada al seu valor corresponent en l'espai CIELAB.

Per altra banda, per a separar i combinar els tres canals d'una imatge utilitzem les funcions *split* i *merge* [4], les quals ja han aparegut abans.

El resultat que obtenim després d'aplicar aquest algorisme és la part de roba repintada amb el nou color, de manera que l'únic que ens fa falta és projectar-la a sobre de la imatge original per tal d'obtenir la imatge sencera final, fent ús novament de la funció *copyTo*.

4.3 Harmonia de color

Finalment, l'última funcionalitat que s'ha incorporat al projecte es basa en la comprovació de l'harmonia dels colors dels elements seleccionats. Un conjunt de colors són harmònics quan són visualment atractius. Per tal de determinar si la combinació de colors actuals dels elements es tracta d'una combinació harmònica, ens hem basat en el mètode proposat per D. Cohen-Or et al. [8], el qual defineix set models o plantilles, les quals podem veure a la Figura 5, per a buscar la que millor s'adapta als colors d'una imatge i corregir-ne la seva distribució de colors en el cas que no sigui harmònica. Les mides precises dels sectors de cada plantilla les podem trobar a l'apèndix.

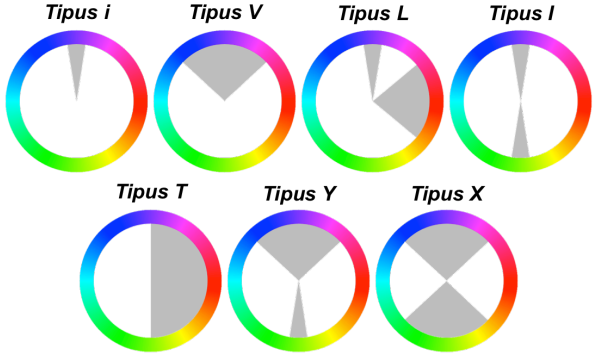


Figura 5: Plantilles harmòniques a la roda de valors *hue*.
Els colors que es troben dins de les zones grises es consideren harmònics. Les plantilles poden ser rotades per un angle arbitrari.

Segons aquest mètode, per tal de determinar la millor plantilla per a una imatge determinada, el primer que es fa és convertir-la a l'espai HSV. Un cop tenim la imatge convertida, el següent pas a dur a terme consisteix en calcular el sumatori de les distàncies de cada píxel de la imatge respecte la plantilla T_m, la qual juntament amb una orientació associada α defineix un *esquema d'harmonia*, denotat per (m, α). Donat un esquema d'harmonia (m, α), es defineix una funció $F(X,(m,\alpha))$, la qual mesura l'harmonia de la imatge X respecte l'esquema (m, α):

$$F\big(X,(m,\alpha)\big) = \sum_{p \epsilon X} ||H(p) - E_{T_m(\alpha)}(p)|| * S(p) \qquad (1)$$

on $H(p)$ i $S(p)$ denoten els valors *hue* i *saturation* del píxel p, i $E_{T_m}(\alpha)(p)$ correspon a la frontera del sector de la plantilla T_m amb orientació α més propera al valor $H(p)$. Els valors de *hue* que resideixen a l'interior dels sectors de la plantilla T_m es considera que tenen una distància zero respecte la plantilla, que és el que busquem.

Llavors, donada una imatge X i una plantilla T_m, es busca el valor de l'angle α que minimitza l'expressió anterior (Eq. 1), el qual definirà el millor esquema d'harmonia de la imatge X respecte la plantilla T_m:

$$M(X,T_m) = (m,\alpha_0)\ \ s.t\ \ \alpha_0 = \underset{\alpha}{argmin}\ F\big(X,(m,\alpha)\big) \qquad (2)$$

D'aquesta manera, el millor esquema d'harmonia $B(X)$ és determinat minimitzant la funció F per totes les plantilles possibles:

$$B(X) = (m_0,\alpha_0)\ \ s.t\ \ m_0 = \underset{m}{argmin}\ F\big(X,M(X,T_m)\big) \qquad (3)$$

Per tant, el que fem és buscar el mínim valor de F per a cada orientació de cada una de les set plantilles i llavors determinar el mínim d'aquests set valors obtinguts. Si aquest valor final és zero, direm que la combinació de colors és harmònica, ja que voldrà dir que s'adapta perfectament a alguna de les set plantilles.

Aquestes fórmules, però, estan destinades a buscar l'harmonia en imatges senceres, considerant-ne tots els píxels, mentre que nosaltres només necessitem observar cinc o menys valors, corresponents als colors dels elements seleccionats a través de l'aplicació (se'n permet un màxim de cinc). D'aquesta manera, si aquests cinc o menys colors s'adapten perfectament a alguna de les set plantilles utilitzant un angle de rotació determinat, direm que la combinació d'aquests colors és harmònica. Quan diem que els colors s'adapten perfectament a alguna de les plantilles ens referim a que els seus valors estan continguts a les regions grises d'aquesta.

En la nostra aplicació, el que es fa és comprovar l'harmonia dels colors dels elements seleccionats cada vegada que s'afegeix un nou element (agafant com a valor la mitjana dels colors d'aquest) o es modifica el color de qualsevol d'ells, excepte en el cas que només tinguem un sol element, ja que sempre s'adaptarà a totes les plantilles. Llavors, es mostra un missatge a través de la interfície de l'aplicació informant sobre si la combinació de colors actual és harmònica o no, com veurem a continuació.

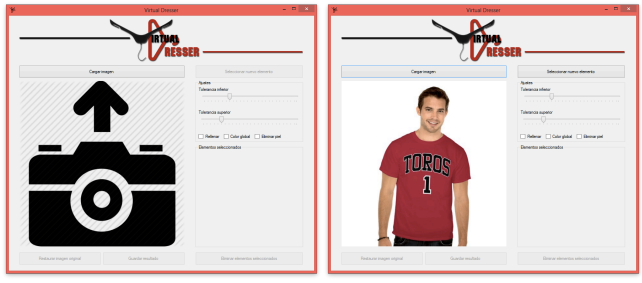
4.4 Interfície gràfica

Per últim, ens queda parlar sobre la interfície gràfica d'usuari de l'aplicació. L'eina utilitzada per a la seva creació ha estat *Windows Forms* [9], la qual es troba integrada al *Visual Studio*. Es tracta d'una interfície de programació d'aplicacions gràfiques que s'inclou com a part del Microsoft .NET Framework i proporciona accés als elements de la interfície de Microsoft Windows nativa, de manera que ens ofereix les eines necessàries per a construir la interfície de l'aplicació desitjada de manera ràpida, simple, i amb els controls típics. Per tal d'aconseguir un disseny adequat i agradable a la vista de l'usuari i evitar males pràctiques, s'han seguit les pautes proposades per J. Spolsky [10].

Per a explicar totes les seves funcionalitats i opcions que permet, ens basarem de l'ajuda de captures per tal que quedi de forma més clara i ordenada.

4.4.1 Carregar imatge

A la Figura 6(a) podem veure el que seria la primera finestra que es mostraria a l'obrir l'aplicació. El logotip de la part superior s'ha dissenyat a través de l'eina *Photoshop CS6* per tal de donar-li un toc d'originalitat. L'única opció disponible que tenim és la de carregar una imatge de disc. A la Figura 6(b) podem observar com la imatge carregada es mostra en el seu espai corresponent i ara ja tenim disponible l'opció de seleccionar un nou element de la imatge. Un cop premem aquest botó, el cursor es convertirà en una creu i podrem seleccionar un punt de la imatge, el qual correspondrà al punt d'origen de l'algorisme de segmentació. Es calcula la correspondència d'aquest punt a la imatge original, ja que les coordenades del punt en la imatge mostrada no coincidiran amb les del punt en la imatge original, excepte en el cas que tinguin la mateixa mida.

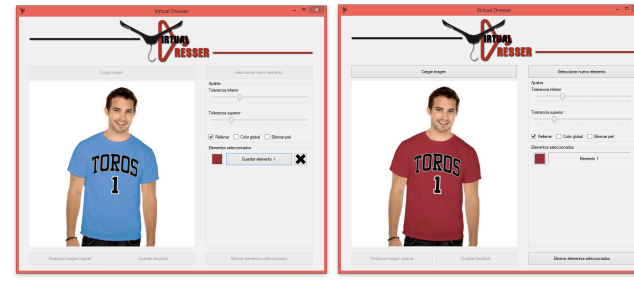


(a) (b)

Figura 6: Pantalla inicial de l'aplicació. (a) Sense imatge carregada. (b) Amb imatge carregada.

4.4.2 Seleccionar un nou element

Un cop cliquem sobre un punt de la imatge, s'aplicarà l'algorisme *floodfill*, es crearà la màscara de l'element seleccionat tenint en compte els valors dels *trackbars* (tolerància superior i inferior de la diferència de color) i els *checkboxs* (opcions de reomplir forats, color global i eliminar zones de pell), i es canviarà el color de l'element utilitzant un color aleatori, controlant que no s'assembli a l'original, per a mostrar una previsualització del resultat i poder veure com varia la màscara depenent dels paràmetres, com podem observar a la Figura 7(a). Cada vegada que es modifiqui un valor dels ajustaments es crearà la nova màscara amb els nous paràmetres i s'aplicarà el mateix color per veure el nou resultat i poder trobar els millors paràmetres per a la imatge determinada. Un cop premem el botó de guardar l'element, es restaurarà el color original d'aquest i se'n guardarà la màscara calculada per a poder aplicar el color desitjat posteriorment. Si l'usuari desitja cancel·lar la operació, simplement ha de clicar la creueta i s'eliminarà l'element. A més, podem observar com el color del botó quadrat de l'esquerra correspon al color original de la samarreta (es calcula la mitjana dels colors de la part de la imatge corresponent a la màscara creada). A la Figura 7(b) podem veure com efectivament es torna al color original de la samarreta i l'element 1 queda "bloquejat". Si en aquest moment modifiquem algun dels ajustaments no es produirà cap efecte, simplement es tindran en compte per al següent element que es seleccioni. També podem observar com ara l'opció d'eliminar els elements seleccionats està disponible, la qual eliminarà tots els elements seleccionats fins al moment i restaurarà la imatge original si aquesta s'ha modificat, obtenint novament el que veiem a la Figura 6(b). L'opció de restaurar la imatge original sense eliminar els elements seleccionats (màscares) i la de guardar la imatge resultant encara no estan disponibles ja que no s'ha modificat la imatge.



(a) (b)

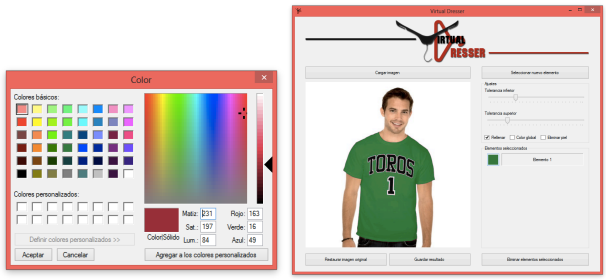
Figura 7: Selecció del primer element. (a) Previsualització del resultat. (b) Primer element guardat.

A la Figura 8 podem veure de forma més clara els paràmetres de segmentació ajustables.

Figura 8: Paràmetres de segmentació.

4.4.3 Canvi de color

A la Figura 9(a) tenim la paleta de colors que s'obre al prémer el botó quadrat del color. Al seleccionar un color i clicar l'opció d'acceptar, entrarà en joc l'algorisme de transferència de color i es substituirà el color actual de l'element pel color seleccionat utilitzant la màscara guardada corresponent. A la Figura 9(b) veiem com quedaria el resultat utilitzant un verd fluix i l'opció de reomplir els forats de les lletres de la samarreta. Podem modificar el color tants cops com es desitgi, i el procediment és molt ràpid. Ara observem que les opcions de guardar el resultat i restaurar la imatge original ja estan disponibles, ja que s'han produït canvis. D'aquesta manera, podem aplicar el mateix procediment pel següent element, i així fins a arribar a un màxim de cinc elements seleccionats, els quals s'han trobat suficients, ja que en general el nombre màxim de colors presents a la roba d'una persona no supera aquesta quantitat, tot i que podria donar-se el cas. A més, no es permet la selecció d'un punt de la imatge que ja pertanyi a la màscara d'algun element existent.

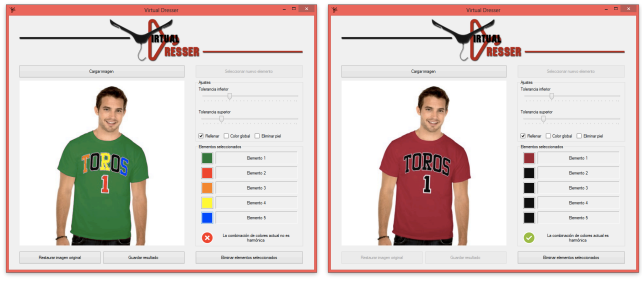


(a) (b)

Figura 9: Canvi de color d'un element. (a) Paleta de colors. (b) Primer element modificat de color.

4.4.4 Comprovació de l'harmonia dels colors

A la Figura 10(a) podem veure el màxim d'elements seleccionats, guardats i canviats de color. En qualsevol moment podem canviar el color de qualsevol d'ells. A la Figura 10(b) veiem el resultat de prémer el botó de restaurar la imatge original. Com es pot apreciar, els elements es conserven i es torna al seu color original, i les opcions de guardar el resultat i restaurar la imatge original tornen a desactivar-se.



(a) (b)

Figura 10: Màxim d'elements seleccionats. (a) Cinc elements modificats de color. (b) Restaurar imatge original.

A l'afegir un element que no sigui el primer ja es comprova l'harmonia del seu color respecte al dels altres elements existents, a l'igual que quan es modifica el color d'algun d'ells. El resultat d'aquesta comprovació el podem veure a les Figures 10(a) i 10(b), notificat al marge dret inferior de la interfície, a sota dels elements seleccionats. A la Figura 11(a) i 11(b) podem veure més clarament el missatge que es mostra en cada cas, utilitzant cinc colors escollits a l'atzar.

(a) (b)

Figura 11: Comprovació de l'harmonia de colors dels elements seleccionats. (a) Cinc colors harmònics. (b) Cinc colors no harmònics.

5 Resultats

A continuació es mostren uns quants resultats obtinguts amb vàries imatges de diferents classes, agrupades per categories per a facilitar-ne l'estudi i comparació.

5.1 Samarretes llises

En primer lloc mostrem resultats obtinguts per a imatges amb samarretes llises, per les quals no és necessari utilitzar la funcionalitat de reomplir buits ni la del color global, simplement aplicant la funció *floodfill*.

Figura 12: Primer grup de resultats. Samarretes llises simples.

Com podem observar a la Figura 12, la fila superior correspon a les imatges originals i la fila inferior conté les imatges resultants corresponents després d'aplicar el canvi de color. Ens adonem fàcilment que els resultats són molt bons i es manté la il·luminació de la roba. A més, no veiem cap rastre de pell que s'hagi escapat. És important destacar el bon resultat de la última imatge, ja que el color de la samarreta és molt semblant al del fons.

Figura 13: Segon grup de resultats. Samarretes llises més sofisticades.

A la Figura 13 veiem imatges amb una aparença més real, amb més ombres i textura, mentre que a la Figura 12 tenim samarretes més simples, amb quasi la mateixa il·luminació en totes elles. Tot i així, observem que els resultats són igual de bons, i es respecta la textura i les zones més i menys il·luminades de les samarretes.

5.2 Samarretes estampades

5.2.1 Opció de reomplir forats

A la Figura 14 ja trobem samarretes amb lletres o dibuixos, de manera que activem l'opció de reomplir els buits causats per aquests estampats pertanyents a la samarreta. Veiem com els forats de les lletres en la primera, tercera i quarta imatge queden perfectament modificats de color, així com l'espai interior del requadre de la segona imatge. A més, veiem com a la última imatge també s'ha canviat el color dels pantalons mantenint-ne la il·luminació original. En cap dels quatre casos s'han escapat franges de pell, ja que els colors de les samarretes no s'assemblen als de la pell.

Figura 14: Tercer grup de resultats. Samarretes estampades amb opció de reomplir forats.

5.2.2 Opció de color global

A la Figura 15 veiem resultats d'aplicar l'opció del color global per a seleccionar totes les lletres de la samarreta del mateix color a través d'un sol clic sobre una d'elles. Com podem observar, en els dos primers casos el resultat és adequat. En canvi, en les altres dues imatges, veiem que s'agafen també parts dels pantalons i una mica del cabell, ja que els colors són molt semblants. Així, podem comprovar els límits d'aquesta funcionalitat. El que s'hauria de fer en aquests casos és seleccionar cada lletra individualment, però recordem que només podem seleccionar un màxim de cinc elements.

Figura 15: Quart grup de resultats. Samarretes estampades simples amb opció de color global.

A la Figura 16 es mostren més resultats d'aplicar l'opció de color global, però amb imatges més sofisticades. En el cas de les samarretes amb ratlles, simplement se'n selecciona una i s'activa l'opció del color global per a considerar-ne la resta, i el mateix amb les samarretes amb requadres. En aquest cas els resultats són molt bons, però si hi haguessin altres elements de la imatge amb color semblant que no fossin ni ratlles ni requadres de la samarreta tindríem el mateix problema que abans, també es segmentarien. És el cas de la quarta imatge, on es selecciona una de les dues bandes negres de la dessuadora però, per semblança de color, també es segmenta part del cabell i ombres produïdes per la caputxa.

Figura 16: Cinquè grup de resultats. Samarretes complexes amb opció de color global.

5.2.3 Imatges amb varis elements

A la Figura 17 tenim més resultats diferents per acabar de completar aquesta secció. Es veu com a totes les imatges s'ha modificat més d'un element de la roba, obtenint uns resultats satisfactoris. El primer cas és destacable ja que la il·luminació és força peculiar i es pot observar com es manté perfectament a l'aplicar el canvi de color. Per a la última imatge s'ha activat la opció de color global a l'hora de seleccionar una de les bambes per a aplicar el mateix color per a les dues, però si es prefereix es pot no activar aquesta opció i aplicar un color diferent a cada bamba.

Figura 17: Sisè grup de resultats. Casos amb varis elements.

5.3 Detector de pell

Pel que fa al detector de pell, com hem dit abans aconsegueix bons resultats per a la majoria d'imatges, però hi ha alguns casos excepcionals en els que no es comporta del tot bé. A la Figura 18 veiem, a la primera fila, les imatges originals, a la segona fila tenim els resultats d'aplicar el canvi de color sense activar el detector de pell, on veiem que es segmenten zones de pell, i a la última fila tenim els resultats després d'activar l'eliminació de pell. A les dues primeres imatges veiem que les zones de pell inicialment detectades són completament eliminades. No obstant, a les altres dues imatges observem que no s'han eliminat algunes franges de pell, les quals formen part d'ombres amb un color vermellós fosc, de manera que els seus valors no es troben dins el rang de la funció *inRange* i el detector no els considera. D'aquesta manera, veiem també els límits d'aquesta funcionalitat al tractar-la com un problema de segmentació de color.

Figura 18: Setè grup de resultats. Eliminació de pell.

6 Conclusions

Arribats a aquest punt, es pot dir que s'han satisfet els objectius proposats del projecte amb èxit tot i existir petites limitacions. En primer lloc, s'ha aconseguit segmentar una o vàries peces o parts de roba determinades i diferenciar-les del seu entorn. En segon lloc, s'ha trobat la manera de modificar el color d'aquests elements mantenint-ne les propietats d'il·luminació i textura originals. Tercerament, s'ha pogut comprovar l'harmonia de colors entre els diferents elements seleccionats. Per últim, ha estat possible acompanyar tot el procés d'una còmode i atractiva interfície gràfica d'usuari per a acabar de satisfer l'objectiu global del projecte.

Per altra banda, s'han introduït noves funcionalitats no plantejades inicialment per a fer el projecte més complet, les quals, tot i presentar algunes limitacions, han demostrat ser de gran utilitat i obtenir bons resultats en la majoria de casos.

A partir d'aquí, el següent pas consistiria en fer l'aplicació més robusta al tipus d'imatges, és a dir, intentar arreglar-ne de la millor manera possible les limitacions actuals. Es podria buscar un mecanisme de segmentació que funcionés correctament per a totes les imatges fent ús dels mateixos paràmetres per a totes elles, i no haver d'ajustar-los mitjançant la interfície de l'aplicació. A més, a l'inici del projecte es va proposar fer proves amb diferents algorismes de segmentació i de transmissió de color, però es va veure que amb els primers que es van utilitzar ja s'obtenien resultats prou bons, així que no se'n van estudiar de nous. D'aquesta manera, estaria bé provar altres algorismes i comparar els resultats obtinguts. Finalment, s'hauria d'implementar un detector de pell més robust, utilitzant un mètode més sofisticat com el proposat anteriorment, i no basat en la segmentació de color.

Agraïments

M'agradaria donar les gràcies al tutor d'aquest treball, Robert Benavente, per la seva ajuda i atenció al llarg de tot el projecte. També voldria agrair al Ramón Baldrich i l'Adrià Ciurana per donar-me un cop de mà en una de les parts del projecte, la d'harmonia de color. Encara que no acabés utilitzant el seu mètode proposat, em va servir d'ajuda per a acabar d'entendre el concepte i els hi agraeixo el temps que em van dedicar.

Bibliografia

[1] E. Reinhard, M. Ashikhim, B. Gooch, and P.Shirley. *Color Transfer Between Images*, IEEE Computer Graphics and Applications, 21(5):34-41, 2001.

[2] J. Elfring. *Image Processing Using OpenCV*, Embedded Motion Control, University of Technology Einhoven, 2013.

[3] S. Brahmbhatt. *Practical OpenCV*, Chapter 7: Image Segmentation and Histograms, 100-103, Apress, 2013.

[4] "OpenCV Documentation". http://docs.opencv.org (Últim accés el 10/02/2015).

[5] G. Wyszecki, W. S. Stiles. *Color Science: Concepts and Methods, Quantitative Data and Formulae*, A Whiley-Interscience publication, 2nd Edition, 1982.

[6] "Color Conversion Algorithms". http://www.cs.rit.edu/~ncs/color/t_convert.html (Últim accés el 10/02/2015).

[7] "Skin Detection: A Step-by-Step exemple using Python and OpenCV". http://www.pyimagesearch.com/2014/08/18/skin-detection-step-step-example-using-python-opencv/ (Últim accés el 10/02/2015).

[8] D. Cohen-Or, O. Sorkine, R. Gal, T. Leyvand, and Y. Xu. *Color Harmonization*, ACM SIGGRAPH 2006 Papers, 624-630, 2006.

[9] "Introducción a los formularios Windows Forms". https://msdn.microsoft.com/es-es/library/ms229601 (Últim accés el 10/02/2015).

[10] J. Spolsky. *User Interface For Programmers*, Apress, 2001.

[11] P. Viola and M. J. Jones. *Robust Real-Time Face Detection*, International Journal of Computer Vision, 57(2):137-154, 2004.

APÈNDIX

A1. DEFINICIÓ FUNCIÓ FLOODFILL

A continuació tenim la definició de la funció *floodfill* d'OpenCV i tots els seus paràmetres:

```
int floodFill(InputOutputArray image, InputOutputAr-
ray mask, Point seedPoint, Scalar newVal, Rect* rect=0,
   Scalar loDiff=Scalar(), Scalar upDiff=Scalar(),
                    int flags=4)
```

- **image** – imatge d'entrada de 8 bits i 1 o 3 canals. És modificada per l'algorisme ja que la regió segmentada es pinta a sobre.
- **mask** – aquest paràmetre és opcional. Si s'especifica, la màscara ha de ser inicialitzada com una imatge d'un sol canal i de 2 píxels més ampla i 2 píxels més alta que la imatge d'entrada. L'algorisme actualitza aquesta màscara (inicialment un fons negre) amb els píxels repintats de la imatge original, de manera que un cop aplicat obtindrem la màscara de la regió de la imatge segmentada.
- **seedPoint** – punt origen, el qual es compara amb els punts veïns.
- **newVal** – el valor nou dels píxels repintats, és a dir, el nou color amb el qual es vol repintar la imatge d'entrada.
- **loDiff** – màxima diferència inferior de color entre el píxel sent actualment observat i un dels seus veïns. Dit d'una altra manera, es tracta de la tolerància o llindar inferior de color a l'hora de determinar la semblança entre els píxels.
- **upDiff** – màxima diferència superior de color entre el píxel sent actualment observat i un dels seus veïns.
- **rect** – paràmetre opcional de sortida establert per la funció com el mínim rectangle que conté la regió segmentada.
- **flags** – flags d'operació. S'especifica el tipus de connectivitat, la qual determina quins píxels veïns són considerats i que per defecte és 4, però pot ser 8. A més, es poden especificar dos flags més:
  - **FLOODFILL_FIXED_RANGE**: Si s'activa, es considera la diferència entre el píxel actual i el píxel llavor. Si no, es considera la diferència entre píxels veïns.
  - **FLOODFILL_MASK_ONLY**: Si s'activa, l'algorisme no té en compte el **newVal** i no repinta la imatge d'entrada, únicament crea la màscara.

A2. DEFINICIÓ FUNCIÓ FINDCONTOURS

A continuació tenim la definició de la funció *findContours* d'OpenCV i tots els seus paràmetres:

```
void findContours (InputOutputArray image, OutputAr-
rayOfArrays contours, OutputArray hierarchy, int mode,
          int method, Point offset=Point())
```

- **image** - imatge d'entrada de 8 bits i un sol canal. Els píxels amb valor diferent de zero són tractats com a 1's i els de valor zero es conserven com a 0's, de manera que la imatge és tractada com a binària i és modificada marcant-ne els contorns trobats.
- **contours** – contorns detectats. Cadascun d'ells és emmagatzemat com un vector de punts.
- **hierarchy** – vector de sortida opcional. Conté informació sobre la topologia de la imatge. Té tants elements com nombre de contorns trobats. Especifica les relacions entre els contorns (pare, fill, veï, etc).
- **mode** – mode de cerca de contorns.
  - **CV_RETR_EXTERNAL.** Agafa només els contorns externs de l'element.
  - **CV_RETR_LIST.** Agafa tots els contorns sense establir cap relació de jerarquia.
  - **CV_RETR_CCOMP.** Agafa tots els contorns i els organitza en una jerarquia de dos nivells. Al nivell superior hi van els contorns externs i al nivell inferior els contorns de l'interior.
  - **CV_RETR_TREE.** Agafa tots els contorns i reconstrueix una jerarquia completa de contorns veïns.
- **method** – mètode d'aproximació de contorns.
  - **CV_CHAIN_APPROX_NONE.** Emmagatzema tots els punts que constitueixen contorns.
  - **CV_CHAIN_APPROX_SIMPLE.** Comprimeix segments horitzontals, verticals i diagonals i deixa únicament els seus punts extrems. Per exemple, un contorn d'un rectangle seria codificat amb 4 punts.
- **offset –** paràmetre opcional segons el qual cada punt del contorn és desplaçat. És útil si els contorns són extrets de la regió d'interès de la imatge (ROI) i llavors s'haurien d'analitzar en el context de la imatge sencera.

A3. DEFINICIÓ FUNCIÓ DRAWCONTOURS

A continuació tenim la definició de la funció *drawContours* d'OpenCV i tots els seus paràmetres:

```
void drawContours(InputOutputArray image, InputArrayO-
fArrays contours, int contourIdx, const Scalar& color,
      int thickness=1, int lineType=8, InputAr-
   ray hierarchy=noArray(), int maxLevel=INT_MAX,
              Point offset=Point())
```

- **image** – imatge destí, on es dibuixaran els contorns.
- **contours** – vector de contorns d'entrada.
- **contourldx** – paràmetre indicant el contorn a dibuixar. Si és negatiu és dibuixen tots.
- **color** – color dels contorns.
- **thickness** – gruix dels contorns. Si és negatiu (per exemple, *CV_FILLED*), es dibuixa l'interior dels contorns.
- **lineType** – connectivitat de la línia, 8 per defecte.
- **hierarchy** – informació opcional de la jerarquia dels contorns. Només és necessari si es vol dibuixar únicament alguns dels contorns.
- **maxLevel** – nivell màxim de jerarquia dels contorns a dibuixar. Si és 0, només es dibuixa el contorn especificat. Si és 1, la funció dibuixa el contorn i els seus contorns veïns. Si és 2, es dibuixa el contorn, els veïns

d'aquest i els veïns dels veïns d'aquest, i així progressivament.

- **offset** – paràmetre opcional segons el qual cada punt del contorn és desplaçat. És útil si els contorns són extrets de la regió d'interès de la imatge (ROI) i llavors s'haurien d'analitzar en el context de la imatge sencera.

A4. MESURES DELS SECTORS DE LES PLATILLES HARMÒNIQUES

Les mides precises dels sectors de les plantilles harmòniques (veure Figura 5) són les següents: els sectors grans dels tipus V, Y i X corrresponen a un 26% del disc (93.6º); els sectors petits dels tipus i, L, I i Y són un 5% del disc (18º); el sector gran del tipus L ocupa un 22% (79.2º); el sector del tipus T és un 50% (180º). L'angle entre els centres dels dos sectors dels tipus I, X i Y és de 180º, i el del tipus L és de 90º.