I7 - Objektumkövetés

Mérési útmutató az
Irányítástechnika és Képfeldolgozás
Laboratóriumhoz: 7. mérés

Dr. Vajda Ferenc
3D érzékelés és Mobilrobotika kutatócsoport
Irányítástechnika és Informatika Tanszék
2015. október 7.

Jelen dokumentum a Budapesti Műszaki és Gazdaságtudományi egyetem Irányítástechnika és képfeldolgozás laborítórium 1. c. tárgyához kapcsolódó mérési útmutató. A mérés és az útmutató a laboratórium 7. méréséhez készült a tanszék 3D érzékelés és mobilrobotika kutatócsoportjában.

A mérések és a mérési útmutató kidolgozásában részt vettek a kutatócsoport tagjai:

Vajda Ferenc
Srp Ágoston Mihály
Szemenyei Márton

Verziók:

Dátum	Verzió	Módosította	Leírás
10/09/15	V0.1	Szemenyei Márton	Első változat

Tartalomjegyzék

1. Arckövetés 4
1.1 A rendszer áttekintése 4
1.2 Annotáció 5
1.3 Képrészlet modell tanulása 6
1.4 Alak modell tanulása 8
1.5 Arcdetektálás tanulása 12
1.6 Arckövetés megvalósítása 12
2. A mérés menete 14
2.1 Feladatok 14
3. OpenCV referencia 15
3.1 Kép létrehozása 15
3.2 Megjelenítése 15
3.3 Műveletek mátrixokkal 15
3.4 Képek mentése merevlemezre 15
3.5 Annotációs adatok olvasása és írása 16
3.6 Alak és Képrészlet modellek használata 16
4. Ellenőrző kérdések 17
5. Irodalomjegyzék 18

1. Arckövetés

Az objektumkövetés a számítógépes látás egyik alapproblémája, amelynek számos releváns felhasználási területe akad. A követés típusa lehet egyszerű pozíciókövetés, vagy a követendő objektum befoglaló téglalapjának meghatározása minden időpillanatban, vagy akár komplex 2D/3D pozíció és orientáció meghatározása. Természetesen a követés elvégzéséhez első lépésként a legtöbb esetben szükséges az objektum kezdeti detektálása, amelyet mindenféle előzetes információ nélkül kell elvégezni.

Ezt követően a legegyszerűbb módja az objektumkövetésnek, ha ezt a detektáló lépést minden képkockán elvégezzük. A teljes képen történő detektálás azonban rendkívül költséges művelet, így egy ilyen algoritmus nem lenne valósidejű. Így az objektumkövető algoritmusok esetén mindig kihasználjuk az objektum korábbi helyzetének (esetleg korábbi mozgásának) az ismeretét. Természetesen feltételeznünk kell, hogy az objektum sebessége a képen korlátos és relatíve alacsony.

Az arckövetés az objektumkövetés egy speciális esete, amely az általános esethez képest újabb nehézségeket vezet be. Ezek közül az egyik az emberi arc rengeteg variációjának figyelembe vétele, amely a kezdeti felismerést különösen nehéz problémává teszi. A másik az arc lokális deformációjának a lehetősége: követés közben az alany arckifejezése megváltozhat, így maga a követendő objektum is változik.

1.1 A rendszer áttekintése

Az előttünk álló feladat egyik fő nehézsége az alábbi kérdés: Mitől arc egy arc? Az agyunk képes emberi arcok robusztus felismerésére (itt most nem konkrét személyek felismeréséről van szó), azonban ez a képességünk nem tudatos, ami azt jelenti, hogy nem vagyunk képesek ezt a tudásunkat egy *egzakt* algoritmus formájába önteni.

A mérnöki tudományok területén ehhez hasonló problémák esetén előszeretettel

használunk tanuló algoritmusokat. A tanuló algoritmusok előnye, hogy nem szükséges előre tudnunk a döntés konkrét módszerét (a döntéshez használt függvényt), elegendő csupán a lehetséges döntésfüggvények családját megadnunk (pl.: a döntés álljon elő a bemenetek lineáris kombinációjaként). A tanuló algoritmusok képesek az optimális döntésfüggvényt a rendelkezésre álló tanító adatok segítségével meghatározni.

A jelenlegi mérés során felügyelt tanuló algoritmusokat fogunk használni. A felügyelt tanulás során az algoritmusnak rendelkezésére áll számos példa kép, amelyeket az általunk elvárt helyes válaszokkal felcímkéztünk (annotáltunk). Az algoritmus ezek segítségével azt a döntésfüggvényt fogja kiválasztani az adott függvénycsaládból, amelyik a lehető legjobban képes az általunk megadott helyes válaszokat reprodukálni.

Természetesen a tanuló algoritmusoknak is számos nehézsége akad. Ezek közül az egyik az elégséges számú tanító adatok előállítása. Minél több és változatosabb példaképek segítségével tanítjuk az algoritmust, annál jobb eredményt érhetünk el. A tanító adatok beszerzése és felcímkézése azonban rendkívül fáradságos munka, ráadásul a számuk növelésével a tanulás folyamata is hosszabbodik. Így minden tanuló algoritmus legfontosabb nehézsége az általánosítás, vagyis az a képesség, hogy olyan példák esetén is jól működjön, amelyek nem szerepeltek a tanító adatok közt.

A mérésen használt algoritmus valójában három tanuló algoritmus ötvözete. Az első algoritmus egy képrészlet modell, amely képes a képen különböző, az emberi arcon előforduló jellemzők felismerésének megtanulására. A második algoritmus egy alak modell, amely az előző algoritmus által detektált arcjellemzők relatív helyzetét, valamint a relatív pozíciókban megengedett deformációkat tanulja meg. Ez az algoritmus nagymértékben segít, hogy az előző lépés hibái ne rontsák el a követést. A harmadik algoritmus a követés inicializálásához, a kezdeti detektáláshoz szükséges. Ez az algoritmus a képrészletek kezdeti helyzetének meghatározását tanulja meg az előzetes arcfelismerés eredménye alapján.

1.2 Annotáció

Egy tanuló algoritmus fejlesztése során az első lépés mindig a tanító adatok készítése. A jelen esetben ehhez képeket kell készíteni egy/több emberi arcról, majd ezeket a tanuló algoritmus igényei szerint fel kell címkézni. Ehhez használható a Mastering OpenCV könyv által adott annotációs eszköz, amely segítségével az alábbi címkézéseket végezhetjük el:

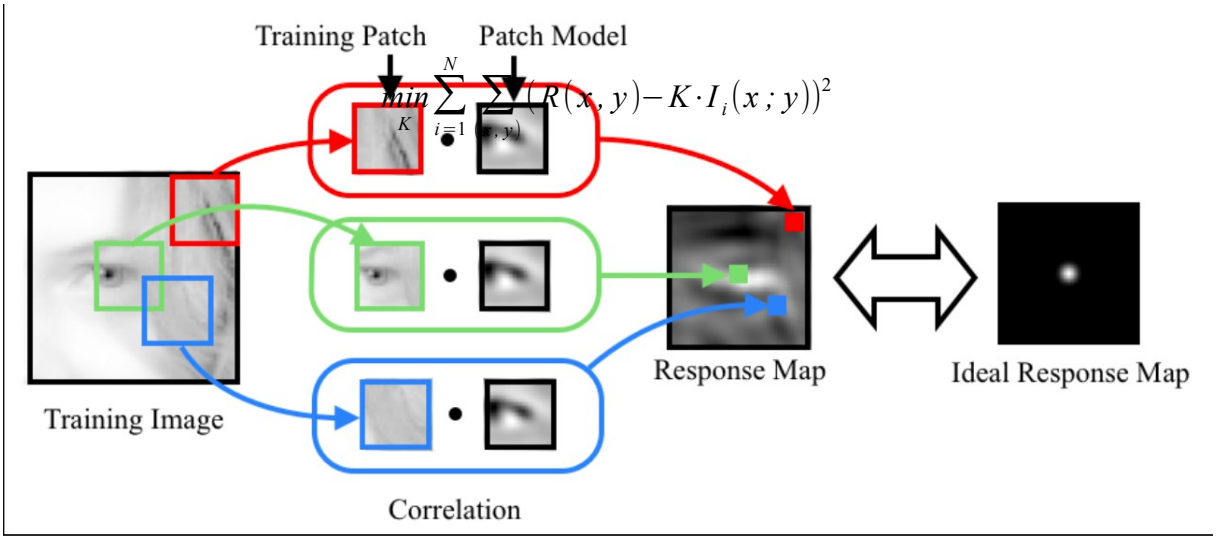
- Arcjellemzők pozíciójának megadása. Ez az elsődleges címkézés, amelyet mindhárom tanuló algoritmus használni fog.
- Arcjellemzők közti szimmetria tulajdonságok megadása. Az emberi arc jó közelítéssel szimmetrikus, így az arcjellemzők is azok lesznek. Ez lehetőséget nyújt a tanító adatok számának növelésére a képek y tengelyre történő tükrözésével. Ennek elvégzésére szükség van az egyes jellemzők szimmetria tulajdonságaira.
- Arcjellemzők közti összeköttetések megadása. Ez a címke a tanításhoz nem szükséges, az eredmények vizualizációját teszi könnyebbé.

***1. ábra**: Az annotáció típusai: A második képen az arcjellemzők pozíciója található, míg a harmadik képen a szimmetria tulajdonságok láthatóak. A negyedik kép egyszerűen a harmadik tükrözött változata. Az utolsó képen a kapcsolat annotációk láthatóak. Forrás: [1]*

1.3 Képrészlet modell tanulása

A képrészlet modell feladata az arc egyes jellemzőinek detektálása a képen. Értelemszerűen minden egyes arcjellemzőhöz külön detektor tartozik. A felismerés implementálására az egyik legegyszerűbb megoldást alkalmazzuk: a template matching eljárást. A template matching során egy előre elkészített speciális kernellel (a template-tel) konvolúciót hajtunk végre a képen, és azt várjuk, hogy a szűrő válaszának

csúcsértéke azokon a helyeken lesz, ahol az adott arcjellemző található.



2. ábra: *Template matching konvolúció/korreláció segítségével. A cél, hogy a válasz kép az ideálishoz minél inkább hasonlítson, tehát a szűrő kimenete csak az arcjellemző tényleges pozíciója körül legyen nagy. Forrás: [1]*

$$D = -\sum_{(x,y)} \left(R(x,y) - K \cdot I_i(x;y)\right) I_i(x;y)$$

$$K = K + \alpha D$$

Mivel a keresett jellemzők kinézetéről nincs előzetes információnk, így a template meghatározása sem lehetséges előzetesen. Ezért egy olyan tanuló algoritmust használunk, amely konkrétan a template pixel értékeit határozza meg. A tanulás során előállítunk egy ideális válaszképet, amely az arcjellemző pozíciója közül nagy, máshol pedig nulla, és megkeressük azt a kernelt, amely az ideális és a tényleges szűrőválasz közti négyzetes hibát minimalizálja. Ezek összege az összes képre lesz a tanítás minimalizálandó költségfüggvénye.

Hol N a tanításhoz használt képek száma, R az ideális válaszkép, T a template/kernel, $I_i(x;y)$ az i-edik kép, x és y középpontú, T-vel megegyező méretű részlete. A fenti tanulási kritérium a lineáris legkisebb négyzetek problémája, amely zárt alakban elvileg megoldható. Azonban ez rendkívül nagy számításigénnyel járna akár egy relatíve kicsi (40x40) kernel esetén is.

Az LLS probléma gyakorlatban elterjedt megoldási módszere a sztochasztikus gradiens módszer. A fent leírt költségfüggvényt elképzelhetjük, mint egy felületet a

kernel szabad változói által kifeszített n dimenziós tér felett. A gradiens módszer során az optimumot kis lépések sorozatával közelítjük meg, úgy, hogy mindig a költségfüggvény legnagyobb növekedésével (a gradienssel) ellentétes irányba lépünk. Sztochasztikus esetben a költségfüggvény gradiensét csupán egyetlen tanító adat segítségével közelítjük az alábbi módon:

Ahol D a gradienssel ellentétes irány, α pedig a tanulási ráta, ami a tanulás sebességét befolyásolja. A tanulás során ezt a lépést egymás után addig ismételjük véletlenszerűen kiválasztott képek segítségével, amíg egy előre választott leállási kritérium nem teljesül. Ez a kritérium általában a lépések számától és/vagy a költségfüggvény változásának nagyságától függ.

***3. ábra**: A megtanult arcjellemzők.*
Forrás: [1]

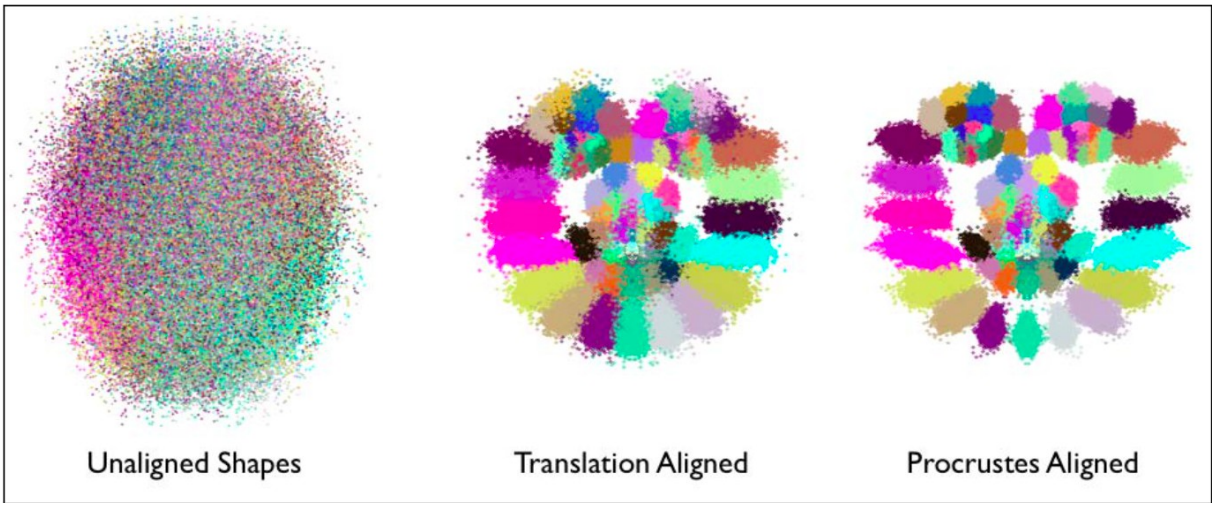
1.4 Alak modell tanulása

A tanuló arckövető algoritmus során alkalmazott alakmodell feladata annak eldöntése, hogy a korábbi lépésben megtanult arcjellemzők milyen geometriai elrendezései tekinthetőek arcszerűnek és melyek nem. Fontos megjegyezni, hogy ebben a lépésben egy merev geometriai modell helyett egy olyat szeretnénk megalkotni, amely képes a különböző arckifejezésekkel járó lokális transzformációk

$$\begin{Bmatrix} (x_1, y_1) \\ (x_2, y_2) \\ \vdots \\ (x_n, y_n) \end{Bmatrix} \rightarrow \begin{bmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \\ \vdots \\ x_n \\ y_n \end{bmatrix}$$

figyelembevételére is.

Mivel azonban a tanuláshoz megadott arcképeken nem csak lokális, hanem globális transzformációk (eltolás, forgatás, skálázás) is találhatóak, ezért egy előzetes illesztési lépésre is szükség van. Ezt egy iteratív eljárással a Procrustes analízissel oldhatjuk meg, melynek során minden képhez külön megkaphatjuk azt a globális transzformációt, amellyel egy közös alakba vihető.



4. ábra: *A Procrustes analízis alapján történő illesztés kimenete. az első képen az arcjellemzők helyzete látható illesztés előtt. A második képen az illesztéskor csak az egyes arcok közti eltolást határoztuk meg, míg a harmadik képen a forgatás, illetve a skálabeli különbségek is figyelembe lettek véve. Forrás: [1]*

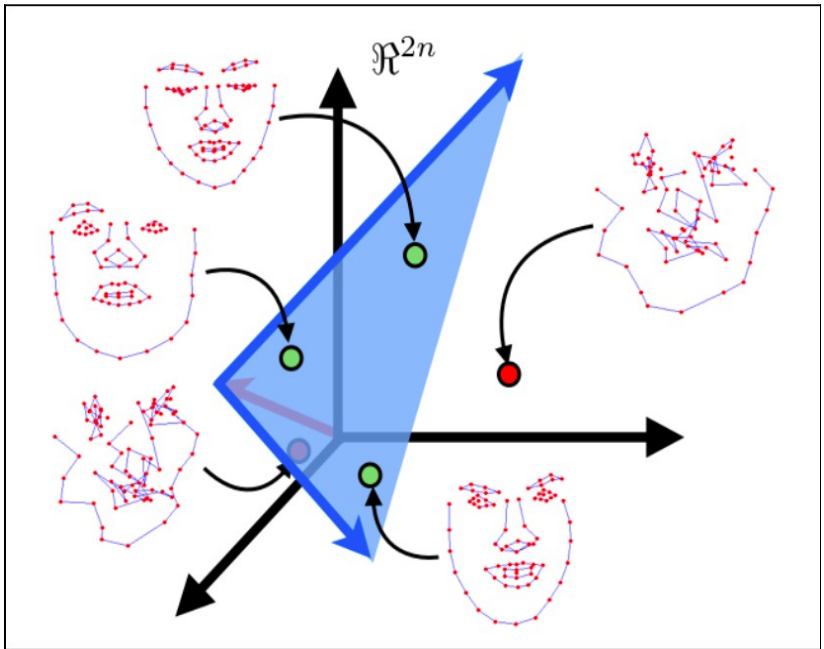
A következő lépés, hogy valamilyen alkalmas matematikai módszerrel leírjuk, hogy mely alakzatok felelnek meg egy arcszerű objektumnak. Az arc modellt igazából arra szeretnénk felhasználni, hogy a képrészlet modell által detektált arcjellemzők halmazához megkeressük a legközelebbi olyan ponthalmazt, amely arcszerű.

Első lépésként az arcjellemzők geometriai elrendezéséből egy vektort konstruálunk: a

különböző arcjellemzők x és y koordinátáit egyszerűen egymás alá írva n db arcjellemzőből egy 2n dimenziós vektort kapunk. Ha a különböző arcjellemzőket az összes képen konzisztens módon sorszámozzuk, akkor ez a konstrukció egyértelmű és az eredeti elrendezés egyszerűen visszaállítható.

Könnyen belátható, hogy ha az egymáshoz illesztett képeken lévő arcjellemző elrendezésekből konstruált vektorok átlagát képezzük, akkor a kapott átlagvektor és átlagelrendezés arcszerű lesz. A kérdés már csak az, hogy melyek azok az elmozdulások, amelyek során az alakzat arcszerű marad. A kérdés megválaszolásához két fontos dolgot kell belátnunk.

Először is, az arcszerű elrendezésekhez tartozó pontok a 2n dimenziós térben nagy valószínűséggel az átlaghoz közel fognak esni, mivel nagy elmozdulások a 2D térben a konstruált vektort is jelentősen megváltoztatják. Fontos továbbá, hogy a kis mértékű elmozdulások közül sem eredményez mindegyik arcszerű objektumot, bizonyos irányú elmozdulások olyan torzításokat jelentenek, amelyek a valós életben nem fordulnak elő

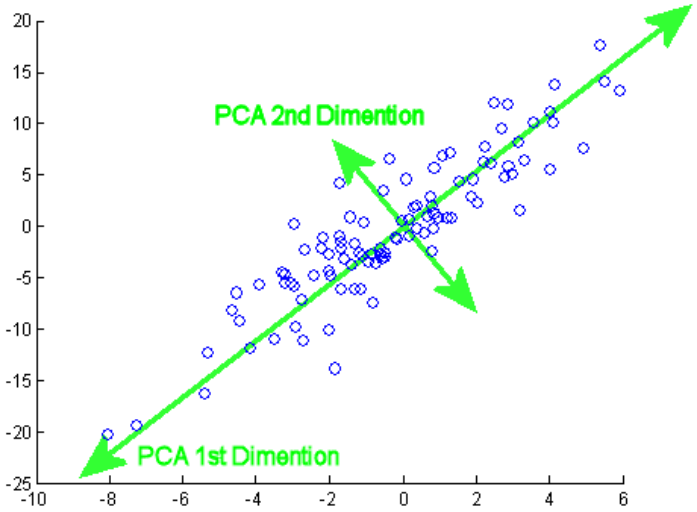


5. ábra: *Az arcszerű elrendezésekhez tartozó pontok a 2n dimenziós térben valamilyen alacsony dimenziós altéren találhatóak, míg a torz ponthalmazok ezen az altéren nincsenek rajta. Az altér leírható az őt kifeszítő bázissal. Forrás: [1]*

emberi arcokon. Ez azt jelenti, hogy az arcszerű elrendezésekhez tartozó vektorok közül 2n dimenziós térben csak egy kevesebb dimenziós felület fog megfelelni.

A továbbiakban az egyszerűség kedvéért feltételezzük, hogy ez a felület lineáris (valamilyen hipersík), amely a 2n dimenziós tér egy altere. Az alakmodell feladata az lesz, hogy a rendelkezésre álló adatok segítségével tanulja meg, hogy pontosan melyik ez az altér. Ehhez fontos azt belátni, hogy mivel a tanító adatok között arcok szerepelnek, ezért az általunk keresett arcszerű deformációkhoz tartozó irányok pont azok, amelyek a tanító adatok között gyakran előfordulnak.

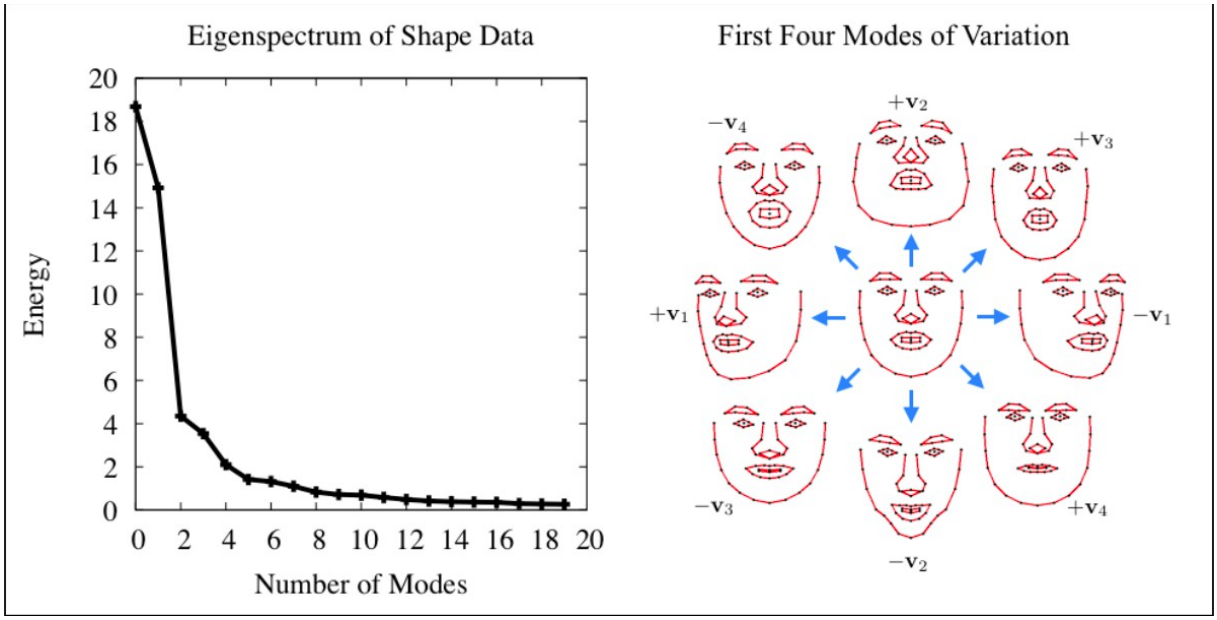
Az alkalmazott módszer a PCA, vagyis a főkomponens analízis lesz, amely egy olyan tanuló algoritmus, amely képes egy sokdimenziós adathalmaz átlag körüli szóródásának szignifikáns irányait meghatározni. A PCA algoritmust gyakran alkalmazzák az adathalmaz dimenziószámának csökkentésére, úgy, hogy az információveszteség minimális legyen.



6. ábra: *A főkomponens analízis alapelve. Egy adathalmazban a legnagyobb szóródás irányait megkeresni és sorba rendezni.*

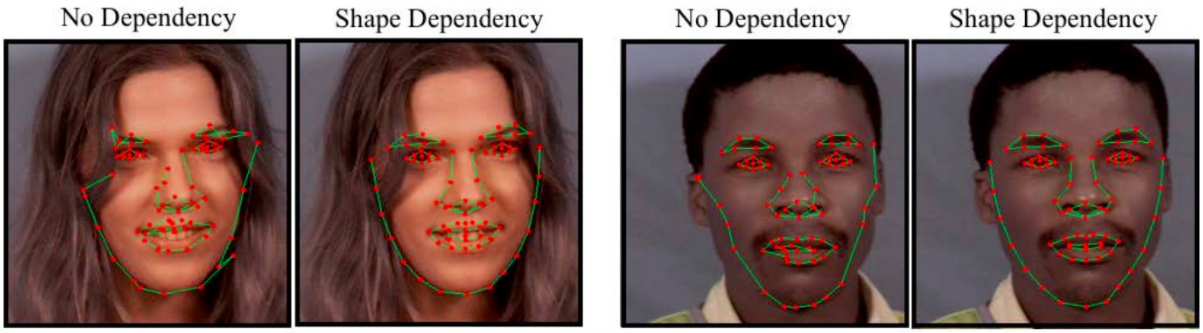
Az algoritmus a működése során először megkeresi a nulla köré centrált adatbázisban legnagyobb szóródás irányát (az első főkomponenst) és a hozzá tartozó szinguláris értéket (a szóródás mértékét). Ezt követően az első főkomponens mentén történő változásokat levonja az adathalmazból, majd ismétli az első lépést. Az így meghatározott főkomponensekből ezt

követően kiválaszthatjuk az első néhányat, amelyek egy olyan alteret alkotnak, amelyre az adathalmazunkat levetítve a lehető legkevesebb információt veszítjük el.



7. ábra: *A bal oldali grafikon a főkomponensekhez tartozó szinguláris értékeket mutatja, vagyis azt, hogy az egyes főkomponensek irányába mennyire szóródik az adathalmaz. A jobb oldali ábrán azt láthatjuk, hogy az első négy főkomponens irányában történő mozgás esetén hogyan változik az arcjellemzők elrendezése. Forrás: [1]*

Mivel a rendelkezésünkre álló adathalmazban arcszerű geometriai elrendezések találhatóak, ezért a legnagyobb szóródással rendelkező irányok a megengedett deformációkhoz tartoznak. Ha a főkomponens analízis segítségével kiválasztjuk az első néhány szignifikáns irányt, akkor egy tetszőleges arcjellemző ponthalmazt erre az altérre levetítve megkaphatjuk a kezdeti elrendezéshez leginkább hasonló arcszerű elrendezést.



8. ábra: *Az arcdetektálás minősége az alakmodel használatával és anélkül. Forrás: [1]*

1.5 Arcdetektálás tanulása

A korábban említett tanuló algoritmusok közül az utolsó az arckövetés inicializálásáért felelős. Az arc detektálásához külön fejlesztésű algoritmus helyett az OpenCV beépített Cascade detektor osztályát használjuk, amely egy objektumdetektáláshoz kifejlesztett tanuló algoritmus. A Cascade algoritmushoz számos előre betanított modell használható, melyek között egy arcdetektáló modell is található.

9. ábra: *Az OpenCV beépített Cascade detektorának a kimenete. Forrás: [1]*

A Cascade detektor azonban csupán egy befoglaló téglalapot ad vissza, amelyben a keresett arc található, így az arcjellemzők kezdeti pozícióját nem tudjuk meghatározni. Ennek a problémának a megoldására egy harmadik tanuló algoritmust használunk. Az algoritmus minden tanító képen meghatározza az arcjellemzők tömegközéppontjának relatív helyzetét a befoglaló téglalaphoz képest, majd ezeket távolság szerint sorba rendezve a medián meghatározásával a relatív pozíció robusztus becslését adja.

1.6 Arckövetés megvalósítása

Az algoritmus utolsó lépése a valósidejű arckövetés megvalósítása. Ez a lépés a korábbi lépésben meghatározott tanuló algoritmusokat használja fel. A lépései a következők:

1. Az openCV Cascade detektorának segítségével az arc befoglaló téglalapjának meghatározása.
2. A befoglaló téglalap segítségével az arcjellemzők kezdeti pozíciójának becslése
3. Minden képkockán:
    1. Az előző/kezdeti arcjellemző koordináták levetítése az alakmodell alterére.
    2. Visszavetítés segítségével megkapjuk a bemeneti ponthalmazhoz legközelebb eső arcszerű ponthalmazt.
    3. Az így korrigált korábbi ponthalmaz kis környezetében megkeressük az arcjellemzők új pozícióját.
    4. Az új pozíciókból kapott ponthalmazt újra levetítjük az altérre, majd visszavetítjük, így megkapva az arcjellemzők új, korrigált pozícióit.

2. A mérés menete

A mérés során a hallgatók betekintést nyernek az objektumkövetés, valamint a tanuló algoritmusok használatának témakörébe. Ehhez a Mastering OpenCV with Practical computer Vision Projects c. könyv által bemutatott arckövető algoritmust, valamint az online elérhető MUCT arckövető adatbázist használják.

2.1 Feladatok

a) Implementálja az arckövető algoritmust az alak és a képrészlet modellek szolgáltatásainak segítségével.

b) Készítsen arc adatbázist a tanuló arckövető algoritmus tanításához.

c) Az elkészített adatbázis segítségével tanítsa be az arckövető algoritmust. Tesztelje az algoritmust robusztusság szempontjából.

d) Fejlesszen OpenCV rutint a korábban elkészített arc adatbázis méretének növelésére.

e) Tesztelje a mérés során elkészített, valamint a mérésvezető által adott előre betanított algoritmusokat. Értékelje a különböző algoritmusok robusztusságát és különbözőségüket.

3. OpenCV referencia

Az alábbiakban néhány példán keresztül bemutatjuk az OpenCV [2] használatát. Részletesebb információ az OpenCV dokumentációjában található.

3.1 Kép létrehozása

```
// szürkeárnyalatos, egycsatornás kép ( a kép alapvetően cv::Mat )
cv::Mat image( cv::Size( width, height ), CV_8UC1 );

// színes, háromcsatornás kép
cv::Mat image( cv::Size( width, height ), CV_8UC3 );

// másik képpel megegyező méretű és típusú kép
cv::Mat image( other.size(), other.type() );

// betöltés fájlból
cv::Mat image = cv::imread( "filename.ext" );
```

3.2 Megjelenítése

```
// kép megjelenítése
cv::imshow( "Ablak neve és egyben fejléce", image );

// várás billentyűzetre
int key = cv::waitKey( 0 ); // várás ideje ms-ban. Ha 0, akkor végtelen
// waitKey LEGALÁBB EGYSZER KELL!!!
```

3.3 Műveletek mátrixokkal

```
// Mátrixszorzás, összeadás, skalárral való szorzás és összeadás
cv::Mat matrix, matrix2;
cv::Mat prod = matrix * matrix2
cv::Mat sum = matrix + matrix2;
cv::Mat mul = matrix * 1.5;
cv::Mat add = matrix + cv::Scalar( 40, 20, 30 );
// Az utolsó művelet egy három csatornás kép/mátrix csatornáihoz a 40, 20,
illetve a 30 értékeket adja hozzá!

// Transzponálás, elemek elérése
cv::Mat transposed = mat.t();
float pixel = mat.at< PIXEL_TYPE >( row, column );
// PIXEL_TYPE általában float, double, vagy char a kép típusától függően
```

3.4 Képek mentése merevlemezre

```
// a kép sorszáma
int cntr = …;

// A sorszám stringgé alakítása 3 vezető 0-val
std::stringstream ss;
ss << setw( 3 ) << setfill( '0' ) << counter;
std::string num = ss.str();

// A kép fájlnevének előállítása
std::string fName = path + „/image” + num + „.jpg”;

// Kép kiírása
cv::imwrite ( fName, image );
```

3.5 Annotációs adatok olvasása és írása

```
// Annotációs fájl beolvasása és mentése
ft_data data = load_ft< ft_data >( path );
save_ft( path, data );

// Képek és annotációs pontok kinyerése
std::vector< cv::Point2f > points = data.get_points( i, false );
cv::Mat image = data.get_image( i );
// Ahol i a kért kép sorszáma
```

3.6 Alak és Képrészlet modellek használata

```
// Arc képrészletekhez tartozó ponthalmaz levetítése az altérre
shape_model shModel;
std::vector< cv::Point2f > points;
shModel.calc_params( points );

// Visszavetítés az altérről
std::vector< cv::Point2f > newPoints = shModel.calc_points();

// Képrészletek lokalizációja a modell segítségével
patch_model pModel;
cv::Size searchSize;
vector< cv::Point2f > detectedPoints = pModel.calc_peaks( image, newPoints,
searchSize );
```

4. Ellenőrző kérdések

1. Foglalja össze néhány mondatban, a mérés célját és feladatait.
2. Mit jelent az, hogy egy algoritmus tanuló?
3. Írja le néhány mondatban, hogy hogyan működik a méréshez használt arckövető algoritmus.
4. Adja meg, hogy milyen dolgokat tanul meg a méréshez használt algoritmus.
5. Miért határozzuk meg és vonjuk le a globális transzformációt a példaképeken?
6. Mi az a PCA (főkomponens analízis)? Hogyan használható a lokális deformációk megtanulására?

5. Irodalomjegyzék

1: Daniel Lélis Baggio; Shervin Emami; David Millán Escrivá; Khvedchenia Levgen; Naureen Mahmood; Jason Saragih; Roy Shilkrot, Mastering OpenCV with Practical Computer Vision Projects, 2012

[2] OpenCV - Open Source Computer Vision Library, www.opencv.org