



BUDAPESTI MŰSZAKI ÉS GAZDASÁGTUDOMÁNYI EGYETEM  
VILLAMOSMÉRNÖKI ÉS INFORMATIKAI KAR  
IRÁNYÍTÁSTECHNIKA ÉS INFORMATIKA TANSZÉK

# Tekintetkövető rendszer fejlesztése képfeldolgozási alapokon

DIPLOMATERV

*Készítette*  
Kovács Balázs

*Konzulens*  
Kertész Zsolt

2013. május 19.

# Tartalomjegyzék

<b>Tartalomjegyzék</b>	<b>I</b>
<b>Hallgatói nyilatkozat</b>	<b>IV</b>
<b>Kivonat</b>	<b>V</b>
<b>Abstract</b>	<b>VI</b>
<b>Bevezető</b>	<b>1</b>
<b>1. Felhasználási lehetőségek</b>	<b>2</b>
1.1. Kutatási felhasználás . . . . .	2
1.1.1. Kognitív pszichológia . . . . .	2
1.1.2. Érzelmedetektálás . . . . .	2
1.1.3. Orvosi felhasználás . . . . .	3
1.2. Gyakorlati felhasználás . . . . .	3
1.2.1. Webergonómia . . . . .	3
1.2.2. Vezetésbiztonság . . . . .	3
1.3. Összefoglalás . . . . .	3
<b>2. A tekintetkövetésről</b>	<b>4</b>
2.1. Tekintetkövetési módszerek . . . . .	4
2.1.1. Az ideális tekintetkövető . . . . .	4
2.1.2. Optikai elvű követés . . . . .	6
2.1.3. Elektromos potenciál-alapú követés . . . . .	9
2.1.4. Követés speciális kontaktlencse használatával . . . . .	9
2.1.5. A tekintetkövetési módszerek összehasonlítása . . . . .	10
2.2. Szemmozgások . . . . .	10
2.2.1. Szakkádok . . . . .	11
2.2.2. Lassú követések . . . . .	12
2.2.3. Fixációk . . . . .	12
2.2.4. Nystagmus . . . . .	13
2.2.5. Összegzés . . . . .	13
2.3. Optikai megfontolások . . . . .	13
<b>3. Elméleti alapok</b>	<b>15</b>
3.1. Hough-transzformáció . . . . .	15
3.1.1. Egyenesek keresése . . . . .	16

3.1.2. Körkeresés . . . . .	19
3.1.3. Összegzés . . . . .	22
3.2. Objektumdetektálás és -követés . . . . .	23
3.2.1. A Viola–Jones objektumdetektor . . . . .	23
3.2.2. A Lucas–Kanade optikai áramlás eljárás . . . . .	24
3.2.3. Összegzés . . . . .	25
3.3. Blob-műveletek . . . . .	27
3.3.1. Komponens-címkézés . . . . .	28
3.3.2. Blob-analizis . . . . .	31
3.3.3. Összegzés . . . . .	33
<b>4. Technológiák</b>	<b>34</b>
4.1. Az OpenCV könyvtár . . . . .	34
4.2. A Qt keretrendszer . . . . .	38
4.2.1. A Qt Creator fejlesztői környezet . . . . .	40
4.3. Fejkamera . . . . .	42
4.3.1. Eszközök . . . . .	42
4.3.2. Módosítások, elhelyezés . . . . .	43
<b>5. Megvalósítás I.</b>	<b>45</b>
5.1. Módszerek összehasonlítása . . . . .	45
5.2. Feldolgozási folyamat . . . . .	47
5.2.1. Pupilakövetés . . . . .	47
5.2.2. Kalibráció, leképezés . . . . .	49
5.3. Validáció . . . . .	51
5.4. Demonstráció . . . . .	53
5.4.1. Pszichológiai bemutató . . . . .	53
5.4.2. Webergonómiai bemutató . . . . .	54
5.5. Összefoglalás . . . . .	56
<b>6. Megvalósítás II.</b>	<b>57</b>
6.1. Követelmények . . . . .	57
6.2. Architektúra . . . . .	58
6.2.1. A MainWindow osztály . . . . .	59
6.2.2. A VideoHandler osztály . . . . .	60
6.2.3. Az ImageProcessor osztály . . . . .	60
6.2.4. A Calibrator osztály . . . . .	61
6.2.5. A Session osztály . . . . .	62
6.2.6. A Recorder osztály . . . . .	62
6.2.7. A Player osztály . . . . .	63
6.2.8. A Heatmapper osztály . . . . .	63
6.2.9. Kisegítő osztályok . . . . .	64
6.3. Grafikus felhasználói felület . . . . .	64
6.4. Felhasználói dokumentáció . . . . .	65
6.5. Tesztelés, eredmények . . . . .	67
<b>Értékelés</b>	<b>68</b>

<b>Köszönetnyilvánítás</b>	<b>69</b>
<b>Irodalomjegyzék</b>	<b>70</b>
<b>Függelék</b>	<b>72</b>
F.1. Paraméterezés . . . . .	72
F.2. Kibővített osztálydiagram . . . . .	73
F.3. Telepítés . . . . .	74

## HALLGATÓI NYILATKOZAT

Alulírott *Kovács Balázs*, szigorló hallgató kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelté után válik hozzáférhetővé.

Budapest, 2013. május 19.

---

*Kovács Balázs*  
hallgató

---

# Kivonat

Jelen dokumentum egy a Budapesti Műszaki és Gazdaságtudományi Egyetem Irányítástechnika és Informatika Tanszékén készített diplomaterv „*Tekintetkövető rendszer fejlesztése képfeldolgozási alapokon*” témában.

---

# Abstract

The current paper is a thesis from Department of Control Engineering and Information Technology at Budapest University of Technology and Economics in the topic of „*Development of a Gaze Tracking System Based on Image Processing*”.

---

# Bevezető

*Hová merült el szép szemed világa?  
Mi az, mit kétes távolban keres?*

...

Tekintetkövető rendszer fejlesztése összetett feladat. A probléma mind magas, mind alacsony absztrakciós szintről megközelítve megfelelő szakmai felkészültséget követel meg: a rendszer megtervezéséhez, összeállításához és implementációjához olyan önálló mérnöki munka szükséges, ami a ideálissá tette számomra a feladatot diplomatervem témájaként. Nem mellesleg az elkészült rendszer remélhetőleg sok hasznos kutatási vagy ipari felhasználásban bizonyulhat hasznosnak.

+++ meg 1-2 bekezdés szoveg +++

+++ melyik fejezetben mi van +++

# 1. fejezet

## Felhasználási lehetőségek

A tekintet megfelelő minőségű és robusztus követésének számos gyakorlati felhasználása lehetséges. Elég csak a *perceptuális* (észlelési), vagy *kognitív* (megértési) kutatási területekre gondolni, ahol például az olvasás, vagy az olvasási folyamatának vizsgálatánál bizonyulhat hasznosnak.

+++ kicsit hosszabb bevezető, meg kepek a fejezetbe mert nagyon szaraz  
+++

### 1.1. Kutatási felhasználás

#### 1.1.1. Kognitív pszichológia

Olvasási folyamatok terén a szakkádok gyors és pontos követése válthat lehetővé, amely a kognitív folyamatok elemzésében komoly segítséget nyújthat. Alvásvizsgálat terén leginkább – nevéből adódóan – a REM (Rapid Eye Movement) fázis kötődik a szemmozgás követéséhez, ebben az alkalmazásban azonban értelemszerűen optikai elvű követés nem jöhét szóba.

#### 1.1.2. Érzelmedetektálás

A pupillaátmérő nem csak a fénymennyiségi változás hatására módosulhat. Érzeli-, izgalmi állapotok is előidézőik a változást, mint például félelem, idegesség vagy örööm [1]. Ez a jelenség szintén potenciális felhasználási lehetőségeket rejti magában. Mivel a pupillareflex akaratlagosan nem koordinálható, állandó fénymennyiséggel mellett a pupillaméret változásának figyelésével detektálható válnak a fent említett érzelmi állapotok. Ehhez a változás mértékének és sebességének pontos mérése szükséges, ami azonban kellően nagy sebességű kamerával és elfogadható számítási teljesítményt nyújtó hardverrel kielégítő minőségben megtehető lehet. Az alkalmazás ráadásul nem feltétlenül igényli a szem közvetlen közelről (például fejre erősített kamerával) történő felvételét. Megfelelően nagy felbontású forrás esetén az arc-, majd szemrégió automatikus szegmentálása után a felismert zónát felhasználva, akár távolról is történhet a pupillareflex vizsgálata.

### 1.1.3. Orvosi felhasználás

Egyes betegségek is okozhatják a pupilla rendellenes méretét vagy viselkedését. Például a „miosis”, azaz a szem összehúzódása nemcsak a fent említett okokra vezethető vissza. Rendellenes összehúzódás alakulhat ki bonyos patológiai állapotok, gyógyszerek, vagy mérgek hatására, sőt a mikrohullámú sugárzásnak kitett szervezet is produkálja ezt a tünetet. A „mydriasis” (a pupilla tágulása) során ugyancsak nem megszokott viselkedés alakulhat ki bonyos gyógyszerek vagy kábítószerek használatkor, de akár komoly fizikai trauma hatására is a normálisnál jelentősebb mértékű vagy időtartamú lehet a pupilla tágulata. A két szem eltérő méretű pupillája (az „anisocoria”) olyan betegségek meglétét jelezheti, mint a Horner-, vagy az Adieszindróma [1]. Orvosi szempontból is van tehát mit vizsgálni: a pupilla követésével egyes betegségek, állapotok felismerése, vagy alakulásuk megfigyelése laikus és orvos számára is automatizálható, megkönyíthető lehet.

Továbbra is orvosi területen maradva a szemmozgás követése és regisztrálása *ontoneuroológiai* vizsgálatokban is szerepet kaphat. Az ilyen vizsgálat célja az egyensúlyszerv működésének megfigyelése és értékelése. Az összetett vizsgálat egyes fázisaiban a szemmozgások követése fontos információt hordoz az alany állapotáról, ugyanis a szemmozgató és az egyensúlyi információkat szállító idegpályák szoros kapcsolatban állnak egymással.

## 1.2. Gyakorlati felhasználás

### 1.2.1. Webergonómia

Ugyancsak felkapott kutatási terület manapság a *webergonómia* területe. Kellően pontos követéssel vizsgálható lehet, hogy a dizájnereknek mennyire sikerült a felhasználók igényeinek megfelelő felületet alkotniuk: könnyen eligazodnak-e rajta, esetleg idejük nagy részét a rossz tervezés következtében kaotikus bolyongással töltik.

+++ meg 1-2 bekezdés +++

### 1.2.2. Vezetésbiztonság

Vezetésbiztonsági alkalmazásban is elképzelhető lehet a pupillakövetés alkalmazása. A követés során mintegy járulékos információként mérhetjük a pislogások gyakoriságát és hosszát, ezzel felismerhetővé válhat a gépjárművezetés közben lankadó figyelem, és jelezhető, ha fennáll az elalvás veszélye. Az eljárás így hasznosnak bizonyulhat már meglévő elalvásdetektálási módszerek [2] kiegészítéseként, tovább javítva azok megbízhatóságát.

+++ meg 1-2 bekezdés +++

## 1.3. Összefoglalás

+++ par mondatos osszefoglalas +++

## 2. fejezet

# A tekintetkövetésről

A tekintetkövető rendszer fejlesztésének megkezdése előtt számos elméleti és gyakorlati szempont veendő figyelembe, hogy a rendszer által nyújtott kritériumok a lehető legjobban közelítsék az elvártat. Nem kerülhető meg az általános technikák, vagy már meglévő megoldások áttekintése a szakirodalomból, aminek segítségével képet alkothatunk a kutatási téma jelenlegi állásáról, és a gyakorlati szempontok figyelembe vételével döntethetjük el, hogy mely módszer mellett tesszük le végül a vokszunkat.

A fejezet 2.1. szakaszában mindenekelőtt a jelenleg elérhető tekintetkövetési módszereket veszem górcső alá, majd hasonlítom össze őket, hogy kellően megalapozott döntést hozhassak a később felhasználni kívánt technikával kapcsolatban.

### 2.1. Tekintetkövetési módszerek

Bár már lassan másfél évtizedes, a témaiban mégis hiánypótló munka Arne John Glenstrupnak és Theo Engell-Nielsennek, a dán Københavns Universitet (Kopenhágai Egyetem) hallgatóinak diplomamunkája [3]. Dolgozatuk jelen szempontból fontos második fejezetét a modern tekintetkövetési technikák bemutatásának és összehasonlításának szentelik, meglátásaik pedig mind a mai napig helytállóak.

Ebben a szakaszban főleg az ő munkájuk alapján szeretném összefoglalni a tekintet követésére felhasználható technikákat, néhol a lényeg kiemelésével, ahol pedig szükséges, a hivatkozott cikk keletkezése óta az idő műlásával érvénytelenné vált adatok, paraméterek aktualizálásával.

#### 2.1.1. Az ideális tekintetkövető

Manapság számos módszer kínálkozik a tekintet követésére. De mégis milyen követelményeket kell teljesíteni az „ideális” tekintetkövetőnek? Scott és Findlay 1993-as munkájukban [4] Hallett eredményeit [5] figyelembe véve meghatározták az ideális tekintetkövető eszköz (vagy rendszer) paramétereit és tulajdonságait.

Ezek szerint az ideális tekintetkövető eszköznek a következő 12 pont által támasztott követelményeket kell teljesítenie:

- a. az arc és a fejrégió könnyen hozzáférhető maradjon

- b. ne legyen fizikailag kapcsolatban a vizsgált szeméellyel
- c. ha szükséges, képes legyen stabilizálni a kapott eredményeket
- d. a rendszer *pontossága* néhány százalékos (1–2 szögperces) eltérést engedjen meg
- e. támogasson legalább egy szögperces *felbontást* másodpercenként, hogy a szempozíció legkisebb változása is követhető legyen; a felbontásnak csak az érzékelő eszköz zaja szabjon határt
- f. támogasson kellően széles *dinamika-tartományt* a szem mozogásainak leképezéséhez
- g. a rendszer időbeli dinamikája legyen megfelelő (jó erősítés, kis fázistolás)
- h. nyújtson *valósidejű* válaszidőt
- i. legyen *invariáns* minden forgási és eltolási szabadságfokra
- j. egyszerűen kiterjeszthető legyen minden két szem feldolgozására (*binokuláris* vizsgálat)
- k. *kompatibilis* meglévő legyen fej- és testfelvételek használatával
- l. tesztalanyok széles skáláján (pl. nem, kor, rassz szerint, vagy szemüvegesek és szemüveg nélküliek körében) használható legyen



**2.1. ábra.** *Tekintetkövető rendszer a Bournemouth University-n.*  
*Forrás: <http://bit.ly/1Zx3ii>*

A fent felsorolt követelmények valóban az ideális esetet testesítik meg. Sokszor egy követelmény enyhítésével, vagy figyelmen kívül hagyásával más követelmények kielégítése jelentősen egyszerűsödik. Ha például a b. pont szerinti fizikai kontaktust mégis megengedjük, rögzíthetjük a követésre használt kamerát a vizsgálni kívánt

alany fejéhez (természetesen kellően diszkrét és kényelmes módon). Ebben az esetben a *i.* pont szerinti invariancia biztosítása márás egyszerűbb feladat, mint külső nézőpontból geometriai transzformációk segítségével végezni ugyanezt.

Több más, látszólag egymásnak ellentmondó követelményt is észrevehetünk az ideális tekintetkövető eszköz 12 pontja között. Komoly megoldandó mérnöki feladatot jelent az ütközések feloldásával az ideális eszközt különböző, valós életben használható megoldásokká alakítani.

A gyakorlatban használható tekintetkövetési megoldások egy lehetséges csoportosítása az alábbi elveken működő iránymeghatározásokat tartalmazza:

1. a szem felületének (vagy a felület speciális megvilágításának) alapján, optikai elven
2. a szem körüli bőrfelület elektromos potenciáljának mérésével
3. speciális kontaktlencse használatával

Már csekély megfontolással látszik, hogy minden csoportnak vannak előnyei és hátrányai. Például az 1. csoportba tartozó megoldások igénylik a legkevésbé – többnyire egyáltalán nem – a fizikai kontaktust a vizsgált szeméellyel. Azonban elképzelhető, hogy ennek az az ára, hogy az ebbe a csoportba tartozó megoldások nem veszik fel a versenyt a másik két alapelt szerint fejlesztett rendszerekkel pl. a pontosság, vagy valamely más kritérium tekintetében.

Ennek megfelelően szükség van a lehetőségek számába vételére, értékelésére, majd összehasonlítására, hogy a legmegfelelőbb technikát tudjuk kiválasztani, ha tekintetkövető eszköz (alkalmazás) fejlesztésébe fogunk. A következő szakaszokban ezért röviden ismertetem a fenti kategóriákba eső megoldások alapjait, majd összehasonlítom jellemző tulajdonságaikat, valamint a használatukkal elérhető fontos pontossági- és sebességértékeket.

### 2.1.2. Optikai elvű követés

Az optikai elvű követés során – a nevéből adódóan – optikai úton próbáljuk meg- határozni a tekintet irányát. Ez történhet speciális megvilágítás nélkül, pl. az írisz (Pontosabban az írisz és az ínhártya közti határvonal, a limbus), vagy a pupilla követésével.

A másik lehetséges megoldásra a szem speciális, többnyire infravörös fénnnyel történő megvilágítása, majd a szemgolyó felületén megjelenő tükröződések, visszaverődések vizsgálata kínálkozik. Az infravörös megvilágítás előnye a látható fénnnyel szemben egyrészt az, hogy nem zavarja a vizsgált személyt, nem vonja el a figyelmét, másrészt pedig infraszűrők használatával a látható fény tartományába eső változásokkal a megfelelő körülmények között (lehetőleg kevés napfény a magas infratartalma miatt) invariánssá tehető.

#### Limbuskövetés

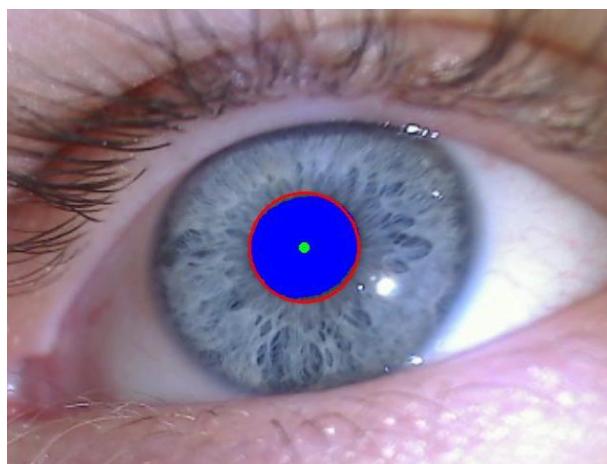
Az írisz és az ínhártya közötti határvonal a többnyire nagy intenzitásváltozás miatt könnyen detektálható lehet. Ugyanakkor meg kell jegyeznünk, hogy az esetek

jelentős részében a limbus számottevő része lehet a szemhéjak takarásában. Ennek megfelelően a technika csak vízszintes helyzet és mozgás követésére alkalmas kielégítő pontossággal [4].

Klasszikus esetben ez a követési technika a limbus *relatív* fejhez képest helyzetén alapul, ezért a vizsgálat közben a fejmozgások teljes hanyagolását, esetleg a tekintetkövető eszköz fejhez rögzítettségét igényli.

### Pupillakövetés

A pupillakövetési technika hasonló az előbb említett limbuskövetéshez, de némi többlet előnyt is magában hordoz. Az egyik ilyen előny, hogy a pupilla közel sincs olyan nagy részben a szemhéjak takarásában, mint a limbus, ennek következtében függőleges irányú követés is megvalósítható lehet. A másik előny, hogy az írisz és a pupilla közti határvonal jóval élesebb, mint az írisz-ínhártya határa, ezért jobb felbontással, pontosabban tudjuk meghatározni a nézeti irányt.



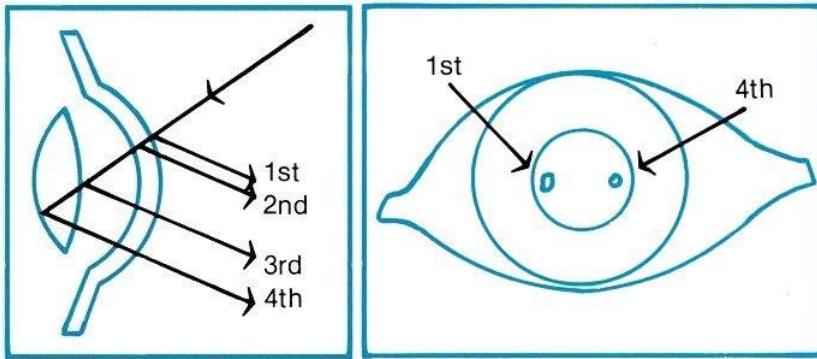
**2.2. ábra.** Pupilladetektálás.  
Forrás: <http://bit.ly/114AeHK>

Az előző technikához képest azonban meg kell említeni a felmerülő nehézségeket is. A kontraszt ugyan lehet, hogy nagyobb az írisz-pupilla határon, azonban az intenzitáskülönbség nem olyan jelentős, mint a limbus környezetében. Az sem elhanyagolható szempont, hogy a pupilla átmérője is szükségszerűen kisebb, mint az íriszé, vagyis a forrásnak relatívan nagyobb felbontásúnak kell lennie, hogy azonos pixelnyi méretű pupillát és íriszt detektálhassunk; ellenkező esetben a kisebb abszolút méret a pontosság rovására mehet.

### Visszaverődés-alapú követés

A szemet (infravörös) fénnel megvilágítva több tükröződés is megfigyelhető lesz a szemlencse és a szaruuhártya határán: ezek az úgynevezett *Purkinje-képek* (lásd 2.3. ábra).

Ezen visszavert képek intenzitása sorrendben egyre csökken, az első azonban (közkeletű nevén az úgynevezett „csillanás”, angolul *glint*) még viszonylag egyszerűen detektálható. Infravörös megvilágításban ugyancsak egyszerű a megfelelő kamerával a pupilláról visszavert fény detektálása – a pupilla a környezeténél jóval nagyobb



**2.3. ábra.** A Purkinje-képek elhelyezkedése.

Forrás: <http://bit.ly/YnAowY>

mértékben veri vissza az infravörös fényt, az infraképen egy „fényes”, kontrasztos objektumot alkotva.

A fenti két objektum egymáshoz viszonyított *relatív* helyzetéből következtethetünk a tekintet irányára, ugyanis az első Purkinje-kép világos pontja, illetve a pupilla kontrasztos ellipszise egymással összefüggésben mozdulnak el a fej vagy a szem mozgatásának következtében.

Az előző bekezdésben foglaltakból látszik, hogy a technika nem igényeli a fej modulatlanságát, vagy a képfelvétel eszköz fejhez rögzítését. Ezen előnye azonban egy megkötést is magával hordoz: egyszerű algoritmusok segítségével kb.  $\pm 12\text{--}15$  foknyi szabadsága van a felhasználónak a fejmozgásokra [4], nagyobb mértékű mozgások esetén ugyanis komplexebb matematikai számítások szükségesek a követett objektumok mozgásának modellezéséhez.

A visszaverődés-alapú megoldásokkal rokonítható még az a módszer, amikor szintén az első Purkinje-kép erős visszaverődését detektálva a képen, kivágunk egy viszonylag kicsi (párszor tíz pixel nagyságrendű) részt, a csillanással a középpontban. Az így kapott képről egy neurális hálózat dönti el, hogy milyen nézeti irányhoz tartozik [6].

A neurális hálózatot természetesen be kell tanítani a használhatóság érdekében. A betanítási procedúra első lépéseként tanítóképeket kell generálni, általános esetben pár percert vesz igénybe, ami alatt a követnie kell egy, a képernyőn megjelenő jelzést. A tanítóképek birtokában ezután a hálózat betanítható, ez a jelenlegi technikai szint mellett kb. tíz perces nagyságrendű időt vesz igénybe. Azonban azonos körülmények és tesztszemély esetén a tanítást nem kell máskor újra elvégezni.

A neurális hálózat előnye a betanítás után, használat közben mutatkozik meg leginkább: felépítéséből adódóan a háló nagyon rövid idő alatt „döntést tud hozni”, így a valósidejű feldolgozási sebesség kritériuma mindenkiéppen kielégített lesz. A döntés eredménye ráadásul akár közvetlenül felhasználható: mindenkorral úgy kell megterveznünk a rendszert, hogy a neurális háló kimenet közvetlenül egy kétdimenziós koordinátát szolgáltasson a felhasznált képernyő koordináta-rendszerében.

A módszer egy másik előnye, hogy nem igényel közeli, nagy felbontású képet a szemrégióról, egy átlagos felbontású kamerával, pl. kartávolságból is elegendő nagyságú lesz a szemrégió. Ennek egyik folyományaként – hogy nagyobb látószögű képek használhatók – adódik, hogy viszonylag nagy mozgási szabadsága lehetséges a vizsgált személy fejének tekintetében, anélkül, hogy a kamerát újra pozicionálnunk.

Ennek a szabadságnak azonban ára van: ha a kalibrálási fázis során több fejpozícióból is rögzítettünk tanító képeket, akkor a betanítás után a neurális hálózat jó esélyel fogja felismerni különböző fejpozíciókban is a tekintet irányát; a mozgási szabadság azonban a pontosság csökkenésével jár. A pontosság egyébként is érzékeny pontja az eljárásnak, ezt pedig éppen úgy növelhetjük, ha tanítás és előhívás közben *nem engedjük meg*, hogy a fejpozíció változzon.

### Purkinje-képek követése

Egy további lehetséges optikai elven működő tekintetkövetési technikát mutatott be Müller *et al* 1993-ban [7], „*Kettős Purkinje-kép*” (Dual-Purkinje Image) módszer néven. Az eljárás lényege, hogy a már említett első és negyedik Purkinje-kép egymáshoz viszonyított helyzetéből számítja a tekintet irányát. Megfelelő matematikai modell esetén a módszer rendkívül pontos, azonban a negyedik Purkinje-kép alacsony intenzitása miatt hatványozottan érzékeny a megvilágítási problémákra.

### 2.1.3. Elektromos potenciál-alapú követés

Az eddigiek től a pupillakövetési probléma egy merőben eltérő megközelítése az *elektrookulográfia* (EOG). Az eljárásból nyerhető elektro-okulogram rögzítése pl. megismerési és kognitív folyamatok, a vizuális információfeldolgozás, vagy az alvásvizsgálat terén szokásos.



**2.4. ábra.** Az EOG eljárás elektródái.

Forrás: <http://bit.ly/13a7IcL>

A módszer a szemgolyó elülső és hátulsó pólusa közötti potenciálkülönbség mérésén alapul, amellyel mind függőleges, mind vízszintes irányban követhető a szem mozgása, de csak körülbelül 1–2 fok pontossággal. A használata azonban korántsem mondható egyszerűnek: az potenciálváltozást érzékelő elektródák (lásd 2.4.) miatt szükséges fizikai kontaktuson kívül a rendszer kalibrációja rendkívül hosszadalmas, és hozzáértést igénylő feladat.

### 2.1.4. Követés speciális kontaktlencse használatával

A szem helyzetének, ebből közvetve a tekintet irányának számításához speciális kontaktlencséket is használhatunk. Az egyik lehetséges megoldás, hogy a lencse anyagában olyan véseteket alakítanak ki (tipikusan valamilyen könnyen felismerhető mintá-

zatot), amelyek a fénytörés felhasználásával megkönnyítik a szem helyzetének meghatározását.

Ha azonban egy megfelelően kis méretű indukciós tekercset is sikerült a kontaktlencse anyagába ágyazni, akkor a fej körül generált nagyfrekvenciás elektromágneses mezőkkel az tekercs helyzete közvetlenül is könnyen meghatározható. Az eljárás körülmenyessége azonban kétségesen teszi a technika laboratóriumokon kívüli felhasználását.

### 2.1.5. A tekintetkövetési módszerek összehasonlítása

A 2.1. táblázat tartalmazza az ebben a szakaszban felsorolt tekintetkövetési módszerek összehasonlítását. Az összehasonlítás szempontjai a módszer által igényelt kontaktust, az általa nyújtott pontosságot, illetve felbontást (hogy a bemeneti képen mekkora minimális elmozdulás jelent változást a kimeneti pozícióban) jelentik.

**2.1. táblázat.** A felsorolt tekintetkövetési módszerek összehasonlítása.

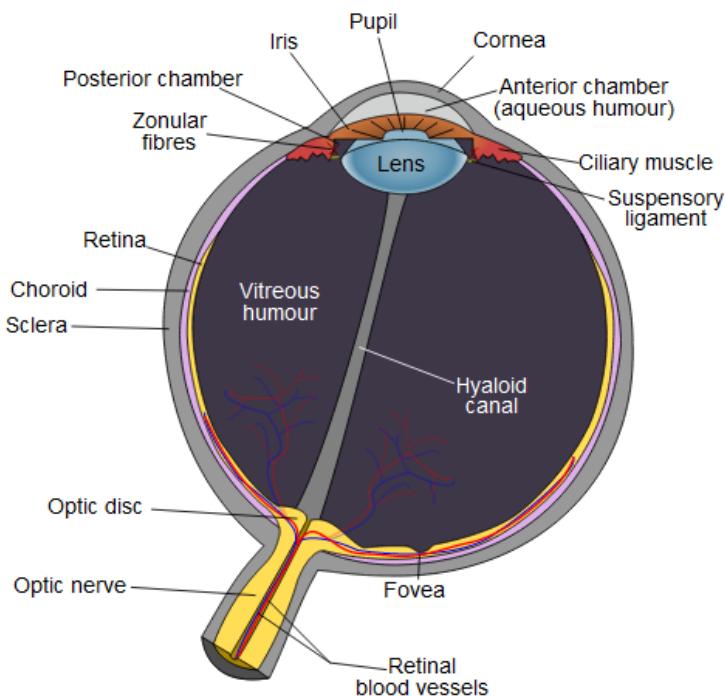
	kontaktus	pontosság	felbontás
limbuskövetés	pl. áll-tartó	1–7°	0,1°
pupillakövetés	nincs	0,003°	0,005°
visszaverődés alapú	nincs	0,5–2°	jó
neurális hálózat	nincs	1,5°	–
kettős Purkinje-képek	nincs	0,017°	0,25°
elektro-okulográfia	elektródák	±1,5–2°	jó
kontaktlencse	kontaktlencse	0,08°	0,017°

## 2.2. Szemmozgások

A szemmozgások többsége – így a későbbi vizsgálatok szempontjából fontosak is – a retinán elhelyezkedő sárgafolt (*fovea centralis*, lásd 2.5. ábra) helyzetének megváltoztatására szolgál. A látóideg kivezetése mellett elhelyezkedő sárgafolton csak tömörtől egymás mellé rendeződött csapok találhatók. A csapok jó fényviszonyok mellett nagyon magas felbontásban képesek színek gazdag skáláját feldolgozni, ezért a legélesebb látás a sárgafoltra vetített kép esetén lehetséges. Érthető tehát, hogy a sárgafoltot olyan pozícióba célszerű mozgatni, hogy a figyelem tárgyának képe ide vetüljön. Az ehhez szükséges pozicionáló típusú szemmozgások közé tartoznak a *szakkádok*, a *lassú követések*, és – bár meglepőnek tűnik – a *fixációk* is, ezekkel a fejezet további szakaszaiban részletebben foglalkozok.

A *fovea* pozicionálását elősegítő mozgások mellett természetesen más is szerepet kap a kívánt kép előállításában. A teljesség igénye nélkül, ilyenek például a szem divergenciáját illetve konvergenciáját beállító mozgások (mélységérzékelés), valamint azok a reflexszerű mozgások, amelyek az egyensúlyszervvel összhangban beállítják a szemeket a fej térbeli orientációjának megfelelően.

A szakasz első részében sorra vevzem, és bemutatom a pozicionáló típusú szemmozgások alapjait, a mérnöki megközelítésből jelentéktelen részleteket mellőzve. Végül a 2.2.5. szakaszban összefoglalom a megszerzett ismereteket, valamint hogy a



**2.5. ábra.** Az emberi szem felépítése.

Forrás: <http://en.wikipedia.org/wiki/Eye>

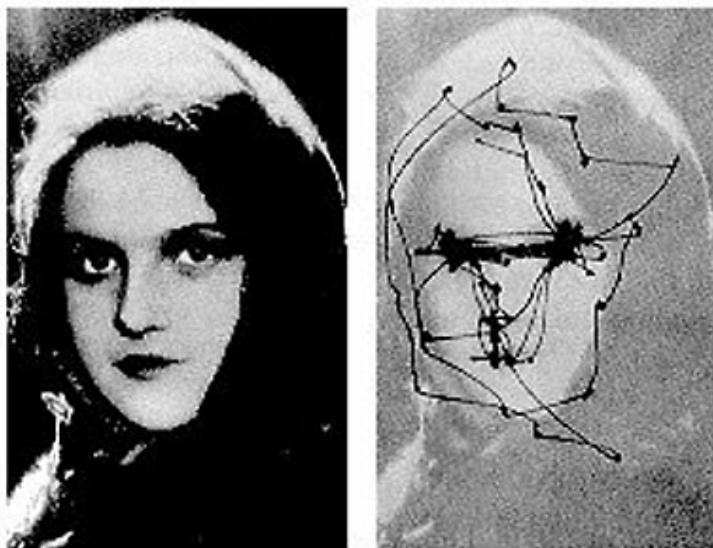
megismert mozgások milyen követelményeket, illetve korlátozásokat támasztanak a realizálandó rendszer tervezése során.

### 2.2.1. Szakkádok

A szakkád minden két szem gyors, egyidejű, azonos irányú mozgását jelenti. Célja a fixáció áthelyezése az egyik tárgyról a másikra. A mozgás során észlelt jellegzetes „ugráló” mintáról (lásd 2.6. ábra) kapta a nevét: a francia *saccader* szó jelentése „rángatás”, „ugrás”.

A szakkádok lehetnek akaratlagosak és reflexszerűek is. Az egyensúlyszervből érkező jelre például reflexszerűen aktiválódnak, míg a tekintetünk, figyelmünk áthelyezése nyilvánvalóan akaratlagos cselekedet. Időtartamban a szakkádok körülbelül 10 és 100 ms közé tehetők. Ez kellően rövid ahhoz, hogy az agy a gyakorlatban ne észlelj, hogy a mozgás alatt nem történik információfelvétel – vagyis a szakkádikus mozgás közben pár pillanatra effektíve vakok vagyunk. [8]

A szakkádok végrehajtásáról megoszlanak a vélemények. Korábban a szakkádokat ballisztikusnak tartották, vagyis amint a következő fixációs pont helye kiszámításra kerül (nagyjából 200 ms idő alatt), a mozgás már nem megszakítható vagy megváltoztatható. [9] Ezt a feltevést támasztotta alá a tény, hogy a végrehajtás 10-100 ms-os időtartama alatt vizuális visszacsatolásra nincs elegendő idő. Léteznek feltevések azonban, amelyek szerint nincs szükség vizuális visszacsatolásra a végrehajtás közbeni megváltoztatáshoz, így a mozgás ballistikussága (a nagy sebességek miatt) csak látszólagos. [10]



**2.6. ábra.** Szakkádok emberi arc vizsgálata során – Alfred L. Yarbus kísérletéből

### 2.2.2. Lassú követések

A lassú követés (*smooth pursuit*) mozgó tárgyak vizuális követésére szolgál. Az ilyen típusú mozgás könnyen modellezhető egy negatív visszacsatolású szabályozóval. [9]

Egy bizonyos sebességgel a szem képes tisztán lassú követést használni a fixáció tárgyon tartására, azonban a  $30^{\circ}/\text{s}$  sebességnél gyorsabban mozgó tárgyak esetén a megfelelő követéshez általában szakkádok beiktatása szükséges. Itt jegyzendő meg, hogy a lassú követés vízszintes és függőleges irányban nem szimmetrikus: a legtöbb ember a vízszintes mozgásokat jobban, míg a függőleges mozgásokat kevésbé tudja lekövetni (ahol „jó” követésen azt értjük, hogy nem szükséges szakkádok beiktatása). A szakkádokkal ellentétben a lassú követést egyértelműen megváltoztathatja az érzékelt vizuális visszacsatolás, a mozgás nem ballisztikus.

Érdekesség, hogy a legtöbben vizuális stimuláció (tényleges mozgó tárgy) nélkül nem tudnak lassú követéses szemmozgást előidézni, csak rövid szakkádok gyors egymásutánját. Szintén megemlíteni kell, hogy a lassú követés bár hasonlónak tűnik ahhoz, amikor a fej mozgását „kompenzálva” egy álló tárgy képét fixáljuk a látómezőn, a két mozgás alapvetően különbözik: már abban is, hogy míg az egyik akaratlagos, a másik reflexszerűen hajtódi végre.

### 2.2.3. Fixációk

A fixációs mozgások a retina stabilizálására szolgálnak, aminek köszönhetően az álló objektumokat tudjuk figyelmünk középpontjában tartani. Az előző szakasz alapján azt gondolhatnánk, hogy a fixációt ugyanazok az idegi pályák aktiválhatják, mint amelyek a lassú követésért felelősek, minden össze a „mozgás” sebessége zérus.

A fixáció azonban több, mint nulla sebességű mozgás: a fixációk közben *tremor*, *drift* és *mikroszakkád* jellegű mozgások váltják folyamatosan egymást. Az állandó mozgást a fényérzékelő sejtek felépítése indokolja. Ha a kép pár másodpercnél tovább marad teljesen változatlanul a retinán, a sejtek telítésbe kerülnek, és a tekintet

elhomályosul.

### 2.2.4. Nystagmus

A nystagmus szakkádok és lassú követések egymásutánja, nem akaratlagos szemmozgás. Előidéződhet optokinetikus úton, illetve patológiásan is (pl. kábítószer-használat hatására). Patológiás nystagmus jelenléte orvosi szempontból lehet jóindulatú, de akár mélyebb neurológiai problémákra is utalhat.

### 2.2.5. Összegzés

A fejezet első szakaszában röviden bemutattam a *fovea* pozicionálására szolgáló szemmozgásokat, amelyek alapszintű megismerése elengedhetetlen a tekintetkövető rendszer fejlesztése során. A rendszernek – ahogy ez a mozgások tulajdonságaiból kitűnik – mind a gyorsaság, mind a pontosság tekintetében komoly követelményeket kell teljesítenie.

Az előrendő *gyorsasághoz* a szakkádok, mint a leggyorsabb sebességű mozgások időtartama (10-100 ms) adhat támpontot. Egy átlagos kamerakép 60 Hz-es frissítése legjobb esetben 16 ms időközönkénti vizsgálatot tesz lehetővé. Ez az eredmény egy tipikus drágább, digitális webkamera 30 képkocka/másodperces képförissítésével már 33 ms-ra csökken, nem is beszélve a 30 FPS-nél gyengébb teljesítményű kaméráról. Tisztában kell lennünk tehát azzal, hogy átlagos képrögzítő hardver használatával előfordulhat olyan szituáció, hogy fizikailag képtelenek vagyunk a nagyon gyors szakkádok pontos követésére.

A *pontosság* tekintetében azt kell figyelembe vennünk, hogy (pl. olvasásnál) a két egymást követő fixációs pont közti eltérés akár szögperces nagyságrendű is lehet. Ezek korrekt elkülönítéséhez nyilvánvalóan minél nagyobb felbontás szükséges. Minél kisebb felbontásúak a felvételek (azaz a szem, illetve a pupilla megjelenítésére relatíve kevesebb képpont kerül), annál nagyobb elmozdulás jut egyetlen képontra, ami a feldolgozás során a legkisebb elkülöníthető egység.

## 2.3. Optikai megfontolások

Az optikai megfontolások közül az első a tekintetkövetés során felhasznált fénytartomány, ami lehet a látható fény tartománya, valamint az infratartomány. A második fontos szempont pedig a kamera látószöge, ami alapjaiban határozza meg a felhasználótól megkívánt beavatkozás mértékét, valamint a szem felbontását (ezzel közvetve az elérhető maximális pontosságot).

A **látható fényben** történő követés egyértelmű előnye, hogy nem igényel speciális hardvert, a legtöbb, a piacon kapható analóg vagy digitális (web)kamera ebben a tartományban „lát”. Sőt, a digitális eszközökben a CMOS/CCD érzékelő elé szinte minden esetben ún. infratüköröt (felülvágó szűrőt, amely kirekeszti a látható fény feletti spektrumot) helyeznek, ezeket az eszközöket tehát sikkerrel csak látható fényben használhatjuk. A látható fény használatakor azonban nem feledkezhetünk meg annak hátrányairól sem: a követéshez analizálálandó kép meglehetősen érzékeny lesz

a megfelelő megvilágításra. A fényviszonyok változásának kiküszöbölése komolyabb előfeldolgozást kíván, és extrém esetekben nem is minden teljesíthető.



**2.7. ábra.** Az emberi szem képe infravörös fényben. Forrás: <http://bit.ly/aKwzsZ>

Az **infratartománynak** a látható fénnyel szemben vannak előnyei, és hátrányai is. Az előnyök közé tartozik, hogy a pupilla képe a infra megvilágításban nagyon kontrasztos (lásd 2.7. ábra), jóval könnyebben szegmentálható az írisztől (különösen sötétebb szemű alany esetén), mint látható fényben. Az infrás megvalósítás hátránya azonban, hogy a megfelelő működéshez sötétet igényel (igaz, ezzel összhangban a látható megvilágítási változásokra invariáns), de sötétben, a tisztán infra megvilágítás megvalósítása nehéz feladat, különösen a piacon kapható egyszerűbb infra LED-ekkel, vagy LED-es reflektorokkal.

A követéshez használt kamera optikája lehet egyrészt **nagy látószögű**. Ez azzal az előnnyel jár, hogy a vizsgálat alatt álló személy fejmozgása szabadabb lehet, nincs szükség a fejpozíció fixálására (pl. álltámasszal). A szemrégió külön szegmentálható, majd ezen régióban belül a pupillát követve valósulhat meg a tekintetkövetés. A módszer hátránya, hogy a szemrégió nem tölti be az egész képet, vagyis nem használja ki maximálisan a rendelkezésre álló felbontást. Ennek következtében a tekintetkövetés pontossága jelentős mértékben romolhat, esetleg lehetetlenné is válhat.

További lehetőség a **teleobjektívek**, vagyis zoomoptikák használata. Teleobjektívvvel a szemrégió „közel hozható” még nagy távolságról is, hogy teljesen kitöltsse a feldolgozandó képet, kihasználva annak teljes felbontását. Ez azonban azzal a hátránnal jár, hogy a fejet fix pozícióba kell kényszeríteni (pl. egy álltámasz segítségével), ugyanis a fej mozgásával a pupillarégió kikerülhet a kamera látószögéből.

Végül lehetőség nyílik egyfajta „hibrid” megoldás használatára is, amelynél a két megoldás előnyeit ötvözhetjük.

## 3. fejezet

# Elméleti alapok

Már a szakdolgozatomban [11] is arra kerestem a választ, hogy melyek lehetnek a pupilla követésének gyors és robusztus megvalósításai. A kérdés a tekintetkövető rendszer esetében is kardinális: a pupilla gyors és pontos követése a elengedhetetlen a megfelelő minőségű követés megvalósításához.

Munkám során számos módszer elméletét kellett megértenem, majd a gyakorlatba ültetnem. Dolgozatomnak ebben a fejezetében ezért a legfontosabb általam vizsgált módszerek alapjaival szeretném megismertetni az olvasót.

A 3.1. szakaszban a Hough-transzformációt mutatom be, kitérve a pupillakeresés szempontjából fontos körkeresési probléma megoldására. A fejezet 3.2. szakaszában az objektumdetektálás és -követés lehetőségének elméletét ismertetem a Viola–Jones objektumdetektor és a Lucas–Kanade optikai áramlás módszerein keresztül. Végezetül a blob-alapú követés alapjait jelentő blobok felismerésével és szűrésével kapcsolatos tudnivalókat foglalom össze a 3.3. szakaszban.

### 3.1. Hough-transzformáció

Képfeldolgozási szűrők, függvények segítségével viszonylag könnyen előállíthatjuk a képtérben számunkra érdekes pontok halmazát. Azokat a pontokat, amelyek valamilyen magasabb absztraktiós szintű képjellemzőhöz kapcsolódnak: például a tárgyak, alakzatok formáját megadó kontúrok pontjait. Ha viszont szeretnénk „megérteni” is a képet, az ilyen formák automatikus és robusztus felismerése szinte elengedhetetlen.

A valóságban a kontúrok pontjai azonban csak többé-kevésbé illeszkednek ideális formákra (egyenekre, körökre, vagy egyéb görbékre): az éleket torzíthatja zaj, egyes él pontok hiányozhatnak, vagy a felismerni kívánt formák kismértékben el is térhetnek az ideálistól. A kép analízise szempontjából azonban ezzel együtt is nagyon értékes információt rejthetnek magukban, ezt az információt pedig hasznos lenne kinyerni. Viszont egyáltalán nem triviális probléma a valamelyen szempontból összetartozó (például egy egyenesre, vagy egy körvonalra eső) pontok csoportosítása.

A **Hough-transzformáció** erre a problémára kínál megoldást. Feladata egyszerű formák, úgymint egyenesek, körök, ellipszisek keresése képeken. Az alakzatok keresését a transzformáció egy ún. paramétertérben végzi egy szavazási mechanizmus segítségével. A felismert potenciális objektumok az akkumulátoron – a paramétertér egyfajta futás közbeni leképezése – lokális maximumai alapján adódnak.



**3.1. ábra.** Egyenesek keresése Hough-transzformáció használatával.  
Forrás: <http://href.hu/x/c91h>

Történelmi áttekintésként megjegyezhetjük, hogy a Hough-transzformációt Paul Hough alkotta meg 1959-ben buborékkamra-fényképek gépi analíziséhez [12]. Ebben a cikkben Hough még csak egyenesek felismeréséről beszél. A manapság használt transzformációt Richard Duda és Peter Hart fejlesztette ki 1972-ben „általánosított Hough-transzformáció” néven [13], tíz évvel azután, hogy Paul Hough szabadalmaztatta módszerét. Az Ő kiegészítésük már lehetővé tette, hogy a transzformáció nemcsak egyenesek, hanem körök és más analitikusan leírható görbek keresésére is felhasználható legyen.

A gépi látás felhasználói körében azonban igazán csak Dana H. Ballard 1981-es cikke [14] nyomán lett népszerű, aki a transzformáció még további felhasználási lehetőségeit vetette fel. Eredményeinek köszönhetően a módszer analitikusan nem (vagy csak nehezen) leírható formák keresésére is alkalmazhatóvá vált. Félreértésekre adhat okot, hogy mind Duda és Hart, mind Ballard az „általánosított” megnevezést használja a módszer általuk kitalált kiegészítéseinél. A továbbiakban az „általános” jelzőt a Duda–Hart-féle kiegészítés megnevezésére használom, ahol Ballard módszeréről esik szó, azt külön jelzem.

A transzformáció általános formájában tehát analitikusan leírható formák keresésére használható. A legegyszerűbb ilyen forma az egyenes, ennek segítségével érthető meg legkönnyebben az algoritmus működési elve. A 3.1.1. szakaszban ezért az egyenesek, majd a 3.1.2. szakaszban a körök keresésének elméletére térek ki.

### 3.1.1. Egyenesek keresése

#### Egyenes-reprezentációk

Egyenesek leírására többféle modell használható. Az egyenes „klasszikus” modellje Descartes-koordinátarendszerben az

$$y = m \cdot x + b \quad (3.1)$$

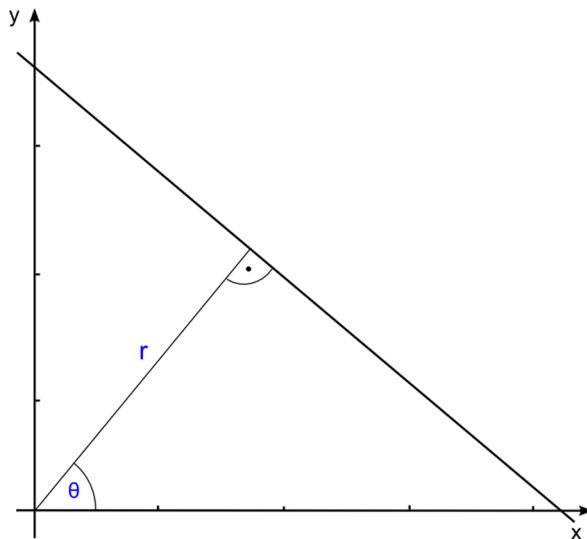
reprezentáció, ahol  $m$  az egyenes meredeksége (iránytangense), a  $b$  konstans az ordinátatengely-metszet, vagyis az egyenes és az  $y$  tengely metszéspontja.

A (3.1) egyenlettel megadott modellel az a probléma, hogy a koordinátarendszer  $y$  tengelyével párhuzamos, vagy ahhoz közelítő egyenesek leírására nem használható  $m$

végtelen nagy értéke miatt. Nem szerencsés azonban az ilyen „függőleges” egyenesek kizárasa a számításból. Ennek kiküszöbölésére – jelen esetben – jobb modellként segítségül hívhatjuk a **Hesse-féle normálalakos** reprezentációt. Ez az

$$r = x \cdot \cos \theta + y \cdot \sin \theta \quad (3.2)$$

egyenlettel adja meg a kívánt egyenest, ahol  $r$  az origótól mért távolságot,  $\theta$  pedig az egyenes pozitív valós féltengellyel bezárt szögét jelenti (lásd 3.2. ábra). Ezzel a megvalósítással már nem ütközik akadályba az  $y$  tengelyhez közelítő egyenesek modellezése.



**3.2. ábra.** Egyenes jellemzése  $r$  és  $\theta$  paraméterekkel.

Érdekességeként megemlíthető, hogy bár nyilvánvalónak látszik a Hesse-féle normálalak előnyösebb volta, Paul Hough eredeti szabadalmában<sup>1</sup> mégis az  $y = m \cdot x + b$  formát használta az egyenesek jellemzésére. Az  $(r, \theta)$  paraméterezés Duda és Hart már hivatkozott cikke nyomán vált általánosan használttá. [13]

### A paramétertér és az akkumulátor

Minden képtérbeli egyenes tehát (3.2) felhasználásával megfeleltethető a paramétertérből egy  $(r, \theta)$  párral jellemzett pontnak. Az egyértelmű megfeleltetés végett két lehetőségünk is adódik a paraméterek korlátainak megválasztására. Ezek formálisan

$$\theta \in [0, 180^\circ) \quad \wedge \quad r \in \mathbf{R} \quad (3.3)$$

vagy

$$\theta \in [0, 360^\circ) \quad \wedge \quad r \geq 0 \quad (3.4)$$

<sup>1</sup>U.S. Patent 3,069,654 – <http://www.google.com/patents?q=3069654>

Az  $(r, \theta)$  párok terét **paramétertérnek**, vagy **Hough-térnek** hívjuk. A paramétertér dimenzióját az ismeretlen paraméterek száma adja, egyenesek esetében ez tehát kettő.

Egy  $(x, y)$  ponton keresztül végtelen sok egyenes húzható, és minden egyenes kielégíti a (3.2) összefüggést. Ezek az egyenesek a paramétertérben ábrázolva egy szinusz-jellegű görbét alkotnak. Természetesen nem határozhatunk meg tetszőleges pontossággal egy pontot, és nem is vehetünk számításba minden (végtelen számú) olyan egyenest, ami az adott ponton átmegy. Szükség van ezért a paramétertér diszkrétizálására.

A paramétertér gyakorlati (kvantált) reprezentációja az **akkumulátor**. Az akkumulátor egy tömb, dimenziósáma megegyezik a paramétertér dimenziósával.

A tömbök indexelése egész számokkal történik ezért az  $r$  paramétert egész számokra célszerű kvantálnunk. A  $\theta$  paraméter kvantálásához pedig a végtelen sok egyenes figyelembe vétele helyett az  $(x, y)$  koordinátán keresztül húzhatunk egyeneseket  $\Delta\theta$  fokonként. Ebből következően a (3.3) vagy (3.4) összefüggések alapján az akkumulátor ezen dimenzióját  $180^\circ/\Delta\theta$ , illetve  $360^\circ/\Delta\theta$  darab diszkrét részre osztja. A  $\Delta\theta$  paraméter megválasztásával, a módszer „finomságát” növelhetjük. Például  $\Delta\theta = 5^\circ$  választás esetén a transzformáció csak a pozitív valós féltengellyel 0, 5, 10, ... fokot bezáró egyenesek felismerésére képes. Túl kis érték választása esetén viszont az eljárás memória- és időigénye fog az egekbe szökni. A  $\Delta\theta$  érték gondos megválasztásával tehát a pontosság és gyorsaság egy lehetőleg optimális értékét kell meghatároznunk.

Gyakorlati szabályként elmondható, hogy 1 fok pontosságnál többre ritkán van szükség. Túl vastag vonalak (amik pedig logikailag összefüggő egyenesek), zajos kép, vagy túl rövid vonalszakaszok esetén így is romlik a felismerés esélye. Hogy miért, arra a következő szakasz – a szavazási mechanizmus ismertetése – ad választ.

## A keresési folyamat

A továbbiakban tételezzük fel, hogy a kép bináris, és megfelelően előfeldolgozott (pl. élkeresés, küszöbözés), azon már csak a felismerni kívánt egyenesek pontjai találhatóak. A háttér képpontjait jelölje a 0 érték, az érdekes pontokat tartalmazó előtér képpontjai legyenek 1 értékűek.

A keresés folyamán bejárjuk a képteret pixelről pixelre. A 0 értékű képpontok biztosan nem részei egy képen található egyenesnek sem, ezért ezekkel nem kell foglalkozunk. Ha 1 értékű pixelt találunk az  $(x, y)$  helyen, az potenciálisan része lehet egy, a képen található egyenesnek. Az akkumulátor minden  $\theta$  értékhez (pl. fokonként) meghatározzuk az  $(x, y)$  ponton átmenő,  $\theta$  irányú egyenes origótól vett  $r$  távolságát a (3.2) összefüggés alapján. Az akkumulátortömb értékét minden így kiszámolt  $(r, \theta)$  indexekkel jelölt helyen megnöveljük ( $r$  meghatározásánál természetesen a tömb kvantáltságát figyelembe véve). Ezt a folyamatot úgy is mondhatjuk, hogy az  $(x, y)$  pont a kiszámított  $(r, \theta)$  pontok halmazára, vagyis ezen paraméterek által a képtérben képviselt egyenesekre „**szavaz**”.

Az algoritmus első fázisának lefutása után az akkumulátortömböt kell vizsgálnunk. Optimális esetben a képen egy egyenesre eső pixelek mindegyike (a többi más szavazatuk mellett) szavazott a „valódi” egyenest jelentő  $(r, \theta)$  párra. Az akkumulátortömb maximális értékű  $(r, \theta)$  elemei adják meg tehát a képen található egyeneseket. A zaj, az esetleg több pixel széles vonalak, és a kvantálási pontatlanság miatt azonban cél-

ravezetőbb **lokális maximumokat** (lásd 3.3. ábra) keresni az akkumulártortömbben így meghatározva a **legvalószínűbb** egyenesek  $r$  és  $\theta$  paramétereit.



**3.3. ábra.** A képtér (balra) és az ebből származó paramétertér (jobbra)

egyenesek keresése során.

Forrás: <http://href.hu/x/c91i>

Az előző szakasz végén felvetett problémát most már megválaszolhatjuk. A túl nagy zaj miatt megnő azon képpontok száma, amik nem részei egy egyenesnek sem. Azonban szavazni ezek a képpontok is szavaznak, ezzel mintegy „háttérzajt” adva az akkumulártortömbnek. Túl vastag vonalak esetén pedig több, látszólag helyes paraméterezésű egyenes adódna ugyanarra a vonalra (pl. egymással a vonalon belül párhuzamos egyenesek, de kellően vastag vonal esetén akár „átlós” egyenesek is). Az első esetben az akkumulártortömb elemeinek átlagos értéke nő meg, a második esetben pedig a sok egymáshoz közeli szavazat miatt lokális maximumok értékei kerülnek közelebb az átlagos tömbértékekhez. Mindkét esetben nehezebbé válik a lokális maximumok pontos meghatározása, ez pedig drasztikusan csökkenheti a keresés pontosságát.

### 3.1.2. Körkeresés

#### Körök reprezentációja

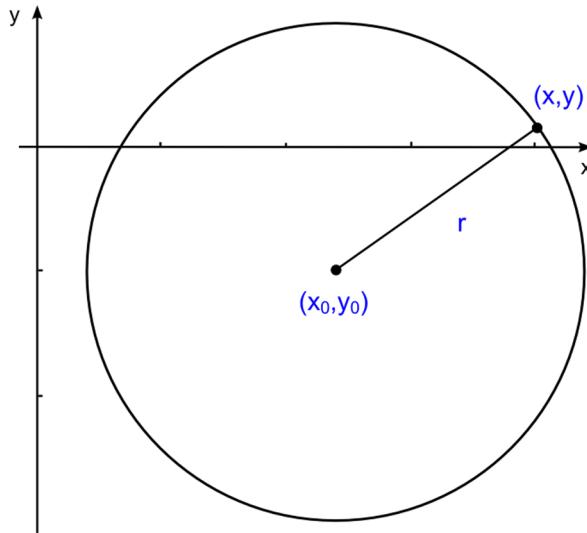
A körökhöz megfelelő reprezentáció megtalálásához még annyit sem kell törnünk a fejünket, mint az egyenesknél tettük. Descartes-koordinátarendszerben az  $(x_0, y_0)$  középpontú  $r$  sugarú körvonall pontjait az

$$(x - x_0)^2 + (y - y_0)^2 = r^2 \quad (3.5)$$

egyenlet adja meg (3.4. ábra). Szögfüggvények használatával az (3.5) egyenletet átalakíthatjuk

$$\begin{aligned}x &= x_0 + r \cdot \cos \theta \\y &= y_0 + r \cdot \sin \theta\end{aligned}\quad (3.6)$$

formára, ahol  $(x_0, y_0)$  szintén a kör középpontjának koordinátái,  $r$  a sugár,  $\theta$  pedig a körvonal egy pontját a kör középpontjával összekötő szakasz pozitív valós félten-gellyel bezárt szögét jelenti.



**3.4. ábra.** Körök reprezentációja.

### Paramétertér körök esetében

Látható, hogy körök esetében már csak akkor elegendő két paraméter – az  $(x_0, y_0)$  középpont-koordináták – a forma leírásához, ha előre adott sugarú köröket keresünk, vagyis  $r$  konstans. Ez azonban csak bizonyos esetekben használható, általános esetben túl nagy megkötést jelent. Tehát a paramétertér dimenziószámát növelnünk kell: az  $r$  sugárparaméter dimenziójával együtt az már **három dimenziós lesz**.

A paraméterteret az akkumulátor tömb mivoltából adódóan itt is egész számokra kell kvantálnunk. Körkeresés esetén az akkumulátor tehát egy három dimenziós tömb, amely a középpont két koordinátájának és a sugárnak egész értékeivel inde-xelhető. Már a pontos algoritmus ismerete nélkül is látszik, hogy a potenciális körök keresése jóval komplexebb művelet az ismeretlen paraméterek magasabb száma miatt.

### A körkeresés folyamata

Az alaphelyzet legyen az egyenesek keresésekor felállított: a felismerni kívánt körvonalak képpontjai legyenek 1, a háttér felismerés szempontjából érdektelen képpontjai pedig legyenek 0 értékűek.

A bejárás során 1 értékű pixelt találva jóval több számítani valónk van, mint egyenesek esetében. Az akkumulátor **összes lehetséges  $r$  sugárparaméterére** meg kell határoznunk a potenciális körök középpontjait. Ehhez felhasználjuk az (3.6)

összefüggéseket  $r$  és  $\theta$  ismeretében ( $\theta$  lehetséges értékeinek megválasztása – tipikusan például fokonként – azonos elven történhet a 3.1.1. szakaszban olvashatóakkal).

Hogy próbálhatjuk meg csökkenteni a rengeteg számítást? Egyfelől **korlátozhatjuk**  $r$  lehetséges értékeinek halmazát. Az értékeknek  $r_{min}$  alsó és  $r_{max}$  felső korlátot adva a keresést a két érték közötti sugarú körökre szűkíthetjük. Némi előzetes információ birtokában (körülbelül, vagy a kép oldalainak arányában mekkora köröket kell keresnünk) jelentősen csökkenthető a számítási idő- és tárigény.

Másfelől pedig felhasználhatunk lokális információkat is: a vizsgálat alatt lévő képpont egy adott ablaknyi környezetét vizsgálva, meghatározhatjuk a körvonali adott pontbeli **gradiensét**. Ez az információ felhasználható a középpont keresésében, ugyanis a kör középpontjának a körvonali pont normálisán kell feküdnie. Így, ha nem is egyetlen kitüntetett irányra, de a számítási pontatlanságot és a zajt figyelembe véve egy szűk tartományra korlátozhatjuk a keresett körközéppontok irányát. Ez a választott tartomány méretével fordított arányosságban csökkenti a szükséges számítások számát. Példának okáért, ha a kiszámított gradiens segítségével akár csak 90 fok pontossággal meg tudjuk határozni a pontban vett normális irányát, a számítások háromnegyedét nem kell elvégezni, ráadásul az akkumulált tömb sem telítődik feleslegesen olyan paraméterekkel, amelyekből biztosan (vagy elég nagy valószínűséggel) nem fog helyes megoldás születni.

### Egyéb körkeresési eljárások

A körkeresésre használt Hough-transzformáció további kiegészítéseivel találkozhatunk H. K. Yuen *et al* hivatkozott cikkében [15]. A cikkben az előző alfejezetben ismertetett sztenderd változat mellett még további négy módosulat vizsgálatát és ezek összehasonlítását végzik el a szerzők. Ezek közül a cikkben „2-1 Hough-transzformáció” néven hivatkozott módszert emelném ki, a későbbiek során ugyanis ennek fontos szerepe lesz. Elnevezésként a cikkben használt „21HT” rövidítést fogom használni.

A **21HT** módszer a [16] és [17] számon hivatkozott cikkekben volt először használatos. A kiegészítés felhasználja a rendelkezésre álló gradiens-információt (lásd: 3.1.2.), és ennek ismeretében pedig a problémát két részre osztja. Mivel a kör középpontjának a körvonali pontok normálisán kell feküdnie, ezen normálisok közös metszéspontja valójában tehát meghatározza a középpontot. Egy kétdimenziós akkumulált tömb pedig elég minden pont saját normálisára eső szavazatainak nyilvántartásához. Ezután a sugár meghatározása a következő módszerrel történhet: meghatározuk minden pont és az előző lépésben kiszámított középpont-jelölt távolságát, majd ezekből az információkból egy sugár-hiszrogramot állítunk elő. Ennek vizsgálatával már meghatározhatjuk a középpontokhoz tartozó sugarakat is. A módszer tárigénye az eredeti megközelítéshez képest jóval kisebb, hiszen csak egy **2D akkumulátort** és egy **1D hiszrogramot** kell használnunk – innen a módszer „2-1” elnevezése.

Eredmények tekintetében az összehasonlító cikk szerint a 21HT módszer felveszi a versenyt a sztenderd megoldással. Igaz, hogy a kétfázisú számítás miatt az első fázisban összeszedett hiba szükségszerűen rárakódik a második fázisra, ezt a pontatlanságot ellensúlyozni tudja a módszer jelentősen kisebb tárigénye.

### 3.1.3. Összegzés

Az eddigiek alapján látszik, hogy a transzformáció felhasználhatóságának szűk keresztmetszetét leginkább a nagy számításigénye adja. Ez egyben legfőbb **hátránya** is: a számításigény a paramétertér dimenziószámának, vagy a kép méretének növekedésével egyre csak növekszik. A dimenziók száma tehát korlátot ad a transzformáció gyakorlati felhasználhatóságának, ezért eredeti formájában leginkább csak egyenesek, vagy körök keresésére használják. A körök keresésének lehetősége azonban kielégítheti a pupillakövetési probléma által támasztott követelményeket. A követés során **előnyére** vállhat, hogy a szavazási mechanizmus használata miatt meglehetősen záros képeken is képes lehet a pupillakontúr megbízható felismerésére. Valós életből vett tesztesetekben – pupillakövetés esetén ezek számítanak – pedig ez korántsem elhanyagolható szempont.

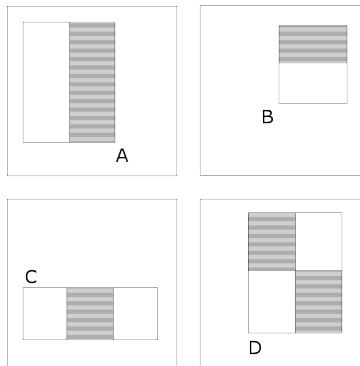
## 3.2. Objektumdetektálás és -követés

Az objektumdetektálás többnyire magasabb absztraktíós szintű képi elemek megtalálására törekszik. Minél magasabb szintű a detektálni kívánt jellemző, azzal arányosan nő a megtalálásához szükséges számítási igény is – hiszen a kép egyre mélyebb megértésére van szükség. Ezáltal veszélybe kerülhet a valós idejű feldolgozási sebesség. A megnövekedett számítási igény azzal kompenzálható, hogy az objektumdetektálást nem minden képkockán végezzük el, hanem két detektálási lépés között próbáljuk egy „olcsóbb” algoritmus használatával követni a felismert objektumot.

A fent vázolt felépítésre lehet jó példa a Viola–Jones objektumdetektor és a Lucas–Kanade optikai áramlás együttes használata, ezek elméleti alapjait mutatom be a szakasz további részében.

### 3.2.1. A Viola–Jones objektumdetektor

A Paul Viola és Michael Jones által 2001-ben elővezetett [19] Viola–Jones objektumdetektor az első olyan objektumfelismerő rendszer, megközelítheti, vagy akár teljesítheti is a valósidejű feldolgozás által támásztott követelményeket. A rendszer eleinte arcdetektálás céljából készült, de mint kiderült, működése könnyen általánosítható, így megfelelő tanítás után bármilyen objektum felismerésére képes lehet.



**3.5. ábra.** A Viola és Jones által használt jellemzőtípusok.  
Forrás: [19]

Az osztályozó úgynevezett *jellemzőkkel* (features) dolgozik, amelyek téglalap alakú területeket alapul véve a bennük lévő képpontok összegét jelentik. Ebben a formában a jellemzők rokonságot mutatnak a Haar bázisfüggvényekkel, amelyek az objektumdetektálásban korábban meghatározó *wavelet transzformáció*nál használatosak. Lehetséges jellemzőtípusokat mutat be a 3.5. ábra.

A jellemző-alapú, úgynevezett *integrális* képrepresentáció esetén az egyes jellemzők ugyan nagyon egyszerűek, viszont konstans időben kiértékelhetők. A jellemzők gyors értékelhetősége azonban nem kompenzálja nagy számukat. minden egyes területre minden jellemzőt kiértékelve csökkenne a rendszer használatával nyert előny. Ennek kiküszöbölésére a betanítási fázisban a Viola–Jones detektor az AdaBoost [20] eljárás egy módosított változatát alkalmazza, amely segít kiválasztani a leginkább fontos jellemzőket, valamint korlátokat ad a kész osztályozó általánosító képességére.

Ha csak az AdaBoost eljárás által kiválasztott fontos jellemzőket értékeljük ki, a rendszer alapesetben akkor sem képes a valós idejű osztályozásra. Ennek kivédésére

a tanítási folyamatban az egyes elemi osztályozókat egymás után kötik (kaszkádosítják), és minden osztályozó bemenetére csak azok a minták jutnak, amelyeket a sorban előtte lévő összes osztályozó elfogadott.

### 3.2.2. A Lucas–Kanade optikai áramlás eljárás

A számítógépes látás területén az optikai áramlás közelítésének kérdése fontos kutatási terület, ennek megfelelően a problémára számos különböző megoldás született. A Bruce D. Lucas és Takeo Kanade által 1981-ben nyilvánosságra hozott [21] differenciális algoritmus manapság az egyik legszélesebb körben használt módszer optikai áramlás számításakor.



**3.6. ábra.** A kép jobb oldalán piros nyílak jelképezik az optikai áramlást.  
Forrás: <http://www.cs.tufts.edu/~dz36>

Az eljárás feltételezi, hogy az áramlás konstans a vizsgált képpont helyi környezetében, és ebben a környezetben (ablakban) a legkisebb négyzetek módszerével próbálja kiszámolni az aktuális pont elmozdulásvektorát. Azzal, hogy nem egy-egy képpontot, hanem egy kis lokális ablakot vesz figyelembe a számítás során, a Lucas–Kanade eljárás kevésbé érzékenyen reagál a zajos képekre. Másrészt viszont a lokális tulajdonság hatására az eljárás nem tud információt szolgáltatni a kép nagy, összefüggő, azonos intenzitású területein.

#### Matematikai háttér

Tegyük fel, hogy két egymás követő időpillanatban a képek közti eltérés megfelelően kicsi. Az optikai áramlás eljárás alapegyenlete ebben az esetben következő

$$I_x V_x + I_y V_y = -I_t \quad (3.7)$$

formában írható fel, ahol  $V_x$  és  $V_y$  az  $(x, y, t)$  képpont sebességének (vagyis optikai áramlásának)  $x$  és  $y$  koordinátái,  $I_x$ ,  $I_y$  és  $I_t$  pedig a kép parciális deriváltjai.

A Lucas–Kanade eljárásban egy, a vizsgált képpont körüli ablak tartalmát vesszük figyelembe, és feltesszük, hogy az áramlás ezen a régióban belül állandó. A (3.7) egyenletet ennek megfelelően a következő formára hozhatjuk

$$\begin{aligned}
 I_x(q_1)V_x + I_y(q_1)V_y &= -I_t(q_1) \\
 I_x(q_2)V_x + I_y(q_2)V_y &= -I_t(q_2) \\
 &\vdots \\
 I_x(q_n)V_x + I_y(q_n)V_y &= -I_t(q_n)
 \end{aligned} \tag{3.8}$$

amelyben  $q_1, q_2, \dots, q_n$  az aktuálisan vizsgált pont körüli ablak egyes képpontjai. A (3.8) egyenletet  $Av = b$  formában, mátrixokkal is felírhatjuk, ekkor

$$\begin{aligned}
 A &= \begin{bmatrix} I_x(q_1) & I_y(q_1) \\ I_x(q_2) & I_y(q_2) \\ \vdots & \vdots \\ I_x(q_n) & I_y(q_n) \end{bmatrix} \\
 v &= \begin{bmatrix} V_x \\ V_y \end{bmatrix} \\
 b &= \begin{bmatrix} -I_t(q_1) \\ -I_t(q_2) \\ \vdots \\ -I_t(q_n) \end{bmatrix}
 \end{aligned} \tag{3.9}$$

A (3.9)-ban definiált egyenletrendszer felülhatározott, vagyis több egyenlet szerepel benne, mint ismeretlen. A Lucas–Kanade algoritmus a legkisebb négyzetek módszerével számítja ki a sebességkomponenseket, vagyis

$$v = \frac{A^T b}{A^T A} \tag{3.10}$$

ami (3.9)-at behelyettesítve a következő alakra hozható

$$\begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^n I_x(q_i)^2 & \sum_{i=1}^n I_x(q_i)I_y(q_i) \\ \sum_{i=1}^n I_x(q_i)I_y(q_i) & \sum_{i=1}^n I_y(q_i)^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_{i=1}^n I_x(q_i)I_t(q_i) \\ -\sum_{i=1}^n I_y(q_i)I_t(q_i) \end{bmatrix} \tag{3.11}$$

vagyis az aktuálisan vizsgált  $q_1, q_2, \dots, q_n$  pontokat tartalmazó ablak  $p$  középpontjának  $x$  és  $y$  irányban vett sebességkomponenseit adja meg közvetlenül a (3.11) egyenlet.

### 3.2.3. Összegzés

A 3.2.1. szakaszban felsorolt eljárásoknak köszönhetően a Viola–Jones objektumdetektor képes az elődeinél jóval gyorsabban osztályozni a bemenetére küldött képeket. Ha azonban nem is lehetséges a videofolyam minden egyes képkockáján a rendelkezésre álló időn belül önállóan döntést hozni, valamilyen kiegészítő technika használatával (pl. a felismert objektumok optikai áramlás-alapú követése) *valós*

*idejű* objektumdetektáló rendszer létrehozása válik lehetővé.

A Lucas–Kanade optikai áramlás-alapú követés jó kiegészítésként szolgálhat egy nagyobb számításigényű objektumdetektor mellé. Azonban a használat során a két módszer egymáshoz viszonyított ideális összetételenek megtalálása döntő a követés pontossága és gyorsasága szempontjából.

Pupillakövetés esetén még kérdéses lehet a módszer pontossága, hiszen alapesetben a Viola–Jones objektumdetektort általánosabb objektumok (pl. arc, vagy szemrégió) felismerésére használják.

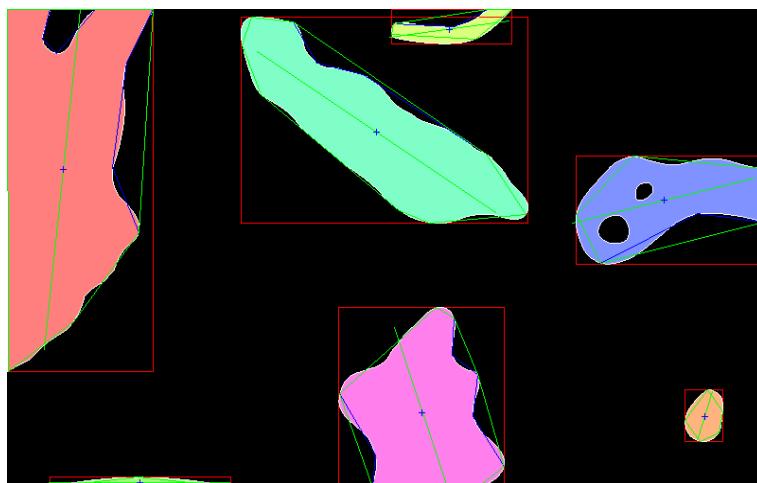
### 3.3. Blob-műveletek

A képfeldolgozásban a *blob* fogalmat a szó „folt” értelmében használják, nem kevérődő össze az adatbázisok BLOB (Binary Large Object – nagy bináris objektum) fogalmával. Egy *blob* informálisan a következő módon definiálható:

**1. Def.** A képtér valamely tulajdonság egyezősége, vagy korlátosozott mértékű eltérése alapján összetartozó régióját **blobnak** nevezzük.

A definícióban említett tulajdonság a gyakorlatban legtöbbször a képpontok színe vagy intenzitása. Azonban más, nem RGB színtérben reprezentált képek esetén egyéb tulajdonság szerinti összetartozás is hasznos információt hordozhat (pl. a HSV színtérben vett szaturáció).

A szakirodalomban két eljárást tartanak számon a blobok meghatározásával kapcsolatban. A **blob-detektálás** (blob detection) differenciális vagy lokális szélsőértékekkel alapuló metodusok segítségével keresi a képtér érdekes pontjait, legszélesebb körben a *Laplacian of the Gaussian (LoG)* operátort használva. A blob-detektálás olyan további információkat szolgáltathat az él- és sarokdetektáló algoritmusok mellett, amelyek elősegíthetik a kép feldolgozását illetve megértését, teret pontosabb nyitva objektumfelismerési és -követési módszerek fejlesztésének.



**3.7. ábra.** Példák blobokra, néhány jellemző jelölésével.

Forrás: <http://bit.ly/10k61se>

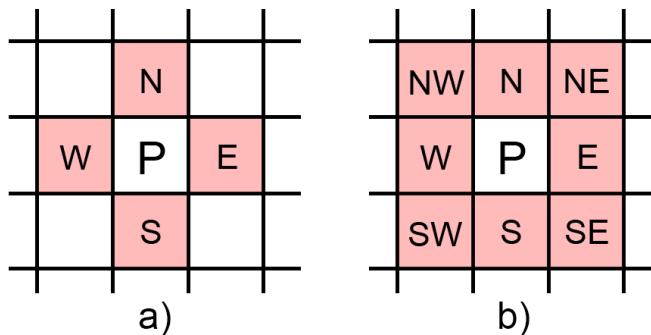
A blob-detektálással rokon, de attól független eljárás a **komponens-címkézés**. A módszer több néven is ismert, pl. régió-címkézés, blob-címkézés, komponens-analízis. A gépi látás terén komponens-címkézésen legtöbbször két dimenziós bináris képek összetartozó régióinak – ezek a blobok – felkutatását és elkülönítését értjük, de a későbbiekben bemutatott elvek kiterjeszhetők szürkeárnyalatos vagy színes képek, illetve akár többdimenziós adathalmazok elemzésére is.

Munkám során a bináris képek feldolgozása került előtérbe, ennek megfelelően a továbbiakban a címkézással kapcsolatos elméleti alapokat foglalom össze. A 3.3.1. szakaszban a címkézási algoritmusokat ismertetem, majd a 3.3.2. szakaszban áttekintem azokat a jellemzőket, amelyek hasznosnak bizonyulhatnak a blobok osztályozása és szűrése során.

### 3.3.1. Komponens-címkézés

Az egyszerűség kedvéért a komponens-címkézés alapelveit két dimenziós képek esetén mutatom be. Könnyen belátható, hogy a későbbiekben ismertetett módszerek egyszerűen kiterjeszhetők többdimenziós halmazok címkézésére is – természetesen az idő- és tárigény növekedése mellett.

A címkézés megkezdése előtt minden esetében fontos, hogy megszabjuk a használni kívánt **szomszédosság-mértéket**. Ez a címkézés során a legtöbbször 4- vagy 8-szomszédosságot jelent. A **4-szomszédossági** (Von Neumann-szomszédosság) esetben egy pixel szomszédait a tőle  $N$ ,  $W$ ,  $S$  és  $E$  irányban lévő képpontok jelentik az égtájak angol elnevezései alapján. **8-szomszédosság** (Moore-szomszédosság) használatakor a képponthoz csak a sarkával illeszkedő pontokat is szomszédnak vesszük, tehát az óramutató járásával ellentétes irányban  $N$ ,  $NW$ ,  $W$ ,  $SW$ ,  $S$ ,  $SE$ ,  $E$  és  $NE$  szomszédokról beszélhetünk (lásd 3.8. ábra).



3.8. ábra. a) 4-szomszédosság b) 8-szomszédosság

A feladat tehát olyan algoritmust találni, amely lehetőleg minél kisebb számítási komplexitással képes adott szomszédosság-mérték mellett elkülöníteni képeken az összetartozó régiókat.

### Kétlépéses algoritmusok

A kétlépéses (two-pass) algoritmusok – nevükből adódóan – két különálló lépében határozzák meg a bináris képen található összefüggő blobokat.

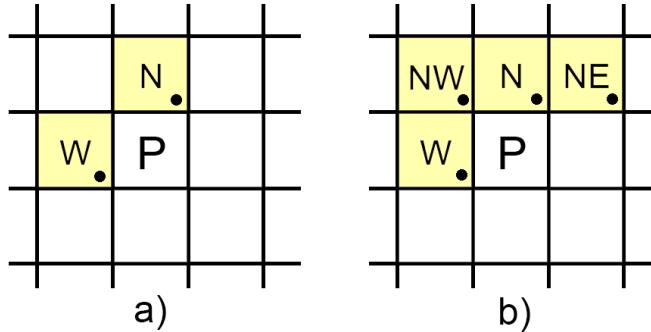
1. Első lépében **átmeneti címkék** kiosztására, valamint az átmeneti címkék közti ekvivalenciák meghatározására kerül sor.
2. Második lépében már meghatározhatjuk a komponenseket a korábban ekvivalensnek jelzett címkék **összevonásával**.

**Rosenfeld–Pfalz** A klasszikus algoritmus [22] minden lépésben sorra veszi a képtér pontjait balról-jobbra, majd fentről lefelé haladva. Ebben a konfigurációban az összetartozást az aktuálisan vizsgált képpontra

- 4-szomszédosság esetén az  $N$  és  $W$  szomszéd,

- 8-szomszédosság esetén az  $NE$ ,  $N$ ,  $NW$  és  $W$  szomszédok

vizsgálatával döntjük el (lásd 3.9. ábra). Az algoritmust 4-szomszédosságot feltételezve ismertetem.



**3.9. ábra.** Kitüntetett szomszékok a Rosenfeld–Pfalz algoritmusnál

- a) 4-szomszédosság b) 8-szomszédosság  
esetén.

Az első lépés során az algoritmus minden képpont esetén a következő négy feltétel vizsgálatát végzi el, és az eredménynek megfelelően látja el a pontot átmeneti címkével. Egy feltétel teljesülése esetén a vizsgálat leáll, egyébként a következő feltétel vizsgálata következik.

Bár egyértelműnek tűnik, de érdemes megjegyezni, hogy az algoritmus leírásában megkülönböztetjük a képpontok értékét (bináris képek esetén a hátteret figyelmen kívül hagyva ez egyetlen értéket jelent), valamint a hozzájuk rendelt címkét!

Az éppen vizsgálat alatt lévő pontot jelölje  $P$ , a képpont értékét megadó függvényt  $v$ , a címkézést definiáló függvényt pedig  $c$ . A régiók címkéit jelöljük növekvő pozitív egész számokkal.

1. feltétel:  $v(P) = v(W)$

$W$  szomszéd értéke megegyezik a vizsgált pont értékével.

- $c(P) := c(W)$

Mindenképpen azonos régióban vagyunk, alkalmazzuk  $P$ -re  $W$  címkéjét!

2. feltétel:  $v(W) = v(N)$  és  $c(W) \neq c(N)$

$W$  és  $N$  szomszéd értéke megegyezik, de címkéjük különböző.

- $c(P) := \min\{c(W), c(N)\}$

A két szomszédnak nyilvánvalóan egy régióba kellene tartoznia. Alkalmazzuk a kisebbik címkét, és jegyezzük fel a címkék ekvivalenciáját!

3. feltétel:  $v(W) \neq v(P)$  és  $v(N) = v(P)$

$W$  értéke különbözik, de  $N$  értéke megegyezik vizsgált pont értékével.

- $c(P) := c(N)$

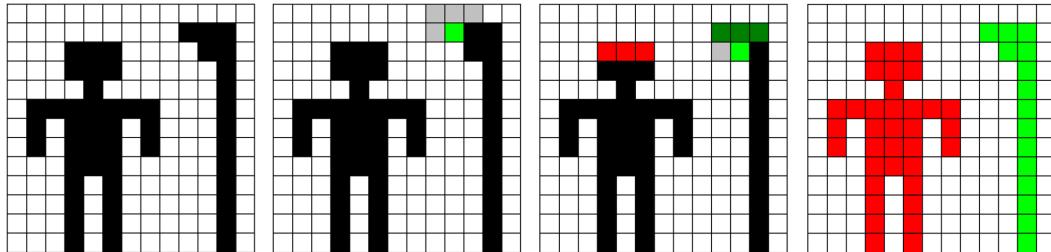
Alkalmazzuk az  $N$  szomszéd címkéjét!

4. feltétel:  $v(W) \neq v(P)$  és  $v(N) \neq v(P)$

$W$  és  $N$  értéke különbözik  $P$  értékétől.

- $c(P) := \max\{c\} + 1$

Készítsünk egy új címkét és alkalmazzuk!



**3.10. ábra.** A kétlépéses komponencímkekéző algoritmus néhány fázisa.

Forrás: <http://bit.ly/ZSuWQA>

A **második lépésben** az algoritmus újra végigiterál a képpontokon, és minden képpont címkéjét az ekvivalensnek jelzett címkék közül a legkisebb helyettesíti.

A megvalósítás során az algoritmus *diszjunkt-halmaz erdő* (disjoint-set) adatszerkezetet használ, amely ideális a régiók közti ekvivalenciák tárolására és elérésére.

**Lifeng-Yuyan-Suzuki** A szerzők által 2008-ban prezentált algoritmus [23] a klasszuskusnál gyorsabb feldolgozást tesz lehetővé. Az előzőekhez hasonlóan itt is két lépésben iterálunk végig a képen balról-jobbra, fentről-lefelé, a háttérhez tartozó képpontokat figyelmen kívül hagyva.

Az **első lépés** során elvégzendő feladatok az aktuális képpontra:

1. Vizsgáljuk a képpont szomszédait, a szomszédosság-mértéknek megfelelően!
2. Ha nincsenek szomszédok (mind a háttérhez tartozik), rendeljünk új címkét az elemhez és folytassuk az algoritmust a következő képponttal!
3. Ha vannak szomszédok, címkézzük a képpontot a legkisebb szomszédos címkevel!
4. Tároljuk a szomszédos címkék közti ekvivalenciákat!

A **második lépésben**, hasonlóan az előző megoldáshoz, a képpontokat újra sorba véve újracímkézzük azokat az ekvivalens címkék legkisebbikével.

### Egyéb algoritmusok

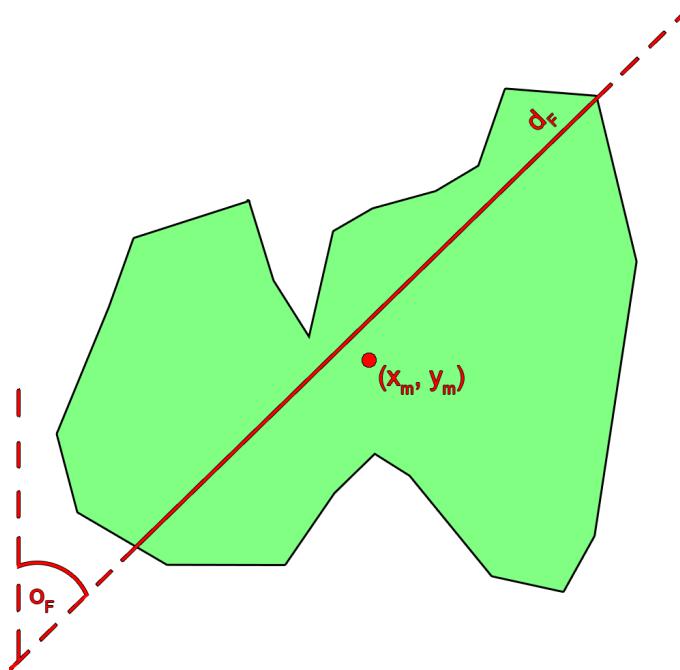
A komponens-címkézés problémájára a 3.3.1. szakaszban taglaltakon kívül is számos más megoldás született. Léteznek a feldolgozást összevonó egylépéses (one-pass) algoritmusok, de olyan többlepések (multi-pass) algoritmust is kifejlesztettek már, amely lineáris komplexitással oldja meg a feladatot. [24]

Az egyesével, szinte egymástól függetlenül vizsgált képpontok feldolgozása párhuzamosításért kiált: párhuzamosított komponens-címkéző algoritmusok fejlesztésére is sor került [25], és a téma továbbra is nagy érdeklődésre tart számot a párhuzamosított számítási platformok (pl. CUDA, OpenMP) előretörésével.

### 3.3.2. Blob-analizis

Sok esetben a képen felismert blobok nem mindegyik tartalmaz a feldolgozás szempontjából hasznos információt. A blobok algoritmikus megtalálása mellett szükséünk lehet a foltok által hordozott absztrakt információ megértésére is. A feladat elvégzéséhez hasznunkra válhat olyan mértékek meghatározása, amelyekkel **szűrni tudjuk** a felismert blobokat valamely előre definiált kritérium alapján.

A pupillakeresési feladatnál például az ember számára könnyen feldolgozható „*keressünk egy relatíve nagy, kör alakú objektumot*” feltételt kell a gépi látás által is megoldható problémává formalizálni.



**3.11. ábra.** Példa néhány blob jellemzőre: súlypont, Feret-átmérő, orientáció

Nem feltétlenül kell azonban rögtön komplex mértékek felhasználásába bonyolódnunk. Jól megfogalmazott kritérium esetén sokszor a legegyszerűbb jellemzők is segítségünkre lehetnek: legyen szó akár egyszerű zajszűrésről, vagy a feldolgozás szempontjából fontos blobok azonosításáról.

Az egyszerűbb mértékek közé tartoznak az objektum kiterjedésével és kerületével kapcsolatos jellemzők.

- **terület**

- objektum területe
- lyukak száma és területe
- teljes terület  
(objektum plusz lyukak területe)

- **kerület**, vagyis a kontúr hossza

- **befoglalók**

- befoglaló keret (bounding box)  
(pl. két ellentétes sarok megadásával)
- elforgatott befoglaló keret  
(pl. középpont, szélesség/magasság valamint irányszög megadásával)
- konvex héj (convex hull)  
(pontjai megadásával)
- befoglaló ellipszis  
(pl. középpont, kis-, nagytengely, irányszög megadásával)

A blobok képen belüli elhelyezkedésének meghatározásakor lehet hasznos a blob **súlypontjának** meghatározása.

**2. Def.** *Súlypont:  $x_m = \frac{\sum_{i=1 \dots N} x_i}{N}$  és  $y_m = \frac{\sum_{i=1 \dots N} y_i}{N}$ , ahol  $N$  a blobhoz tartozó képpontok száma*

A szabályos formáktól való eltérés mértékét határozhatsuk meg a **kompaktság** definiálásával. Jelölje  $P$  a kerületet,  $A$  a területet, valamint  $W$  és  $H$  a befoglaló téglalap szélességét és magasságát.

**3. Def.** *Kompaktság 1:  $\frac{P^2}{A} \Rightarrow$  diszkre minimális*

**4. Def.** *Kompaktság 2:  $\frac{A}{W \cdot H} \Rightarrow$  téglalapra minimális*

A befoglaló keret (bounding box) szélesség/hosszúság aránya az objektum általában vett **elnyújtottságáról** ad információt.

**5. Def.** *Elnyújtottság:  $\frac{W}{H}$*

Az objektum **körkörösséget** (Heywood Circularity Factor) is könnyen definiálhatjuk a kerület és a terület felhasználásával. A képlet tökéletesen kör alakú objektumra 1 értéket ad vissza.

**6. Def.** *Körkörösség:  $\frac{P}{2\sqrt{\pi A}}$*

A blob orientációjának meghatározásában lehet segítségünkre a **Feret-átmérő** fogalma. Az általános definíció szerint Feret-átmérőn azt a értéket értjük, amely távolságba az objektum „beszorítható” a megadott iránnyal párhuzamos egyenesek közé. A fogalom legkönyebben úgy szemléltethető, mintha az objektumot egy tolómérő adott irányba álló pofái közé szorítanánk. A tolómérőn leolvasható érték az objektum mérési irányra vett Feret-átmérője – ebből származik a másik, főleg angolszász területen elterjedt elnevezés, a *caliper diameter* (caliper: tolómérő, tolómérce).

A gépi látás területén Feret-átmérőn általában automatikusan a legnagyobb Feret-átmérőt értjük. A legnagyobb Feret-átmérőnek kitüntetett szerepet tulajdoníthatunk: irányszögét kitűnően használhatjuk a blob **orientációjának** jellemzésére.

### **3.3.3. Összegzés**

Látható, hogy a szakirodalom kiforrott eljárásokat tart számon blobok címkézésére és szűrésére. Munkám során joggal számíthattam arra, hogy megfelelően binarizált képből a fent ismertetett algoritmusokkal lehetséges lesz a pupilla mint egy kitüntetett blob felismerése és követése.

Kellően gyors implementációjú algoritmusok használatával nem jelenthet gondot az akár 30 – párhuzamosított eljárásokkal pedig akár ennél is több – képkockás másodpercenkénti feldolgozási sebesség sem. A felismert blobok pontos és effektív szűrésével a blob-alapú azonosítás és követés performenciában is bőven felveheti a versenyt a többi vizsgált követési módszerrel.

## 4. fejezet

# Technológiák

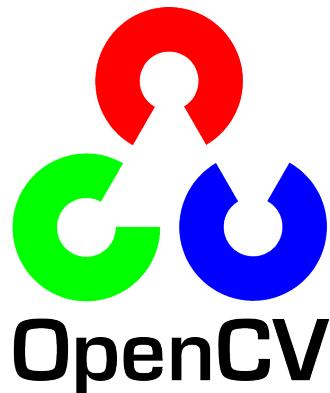
Diplomatervem készítése során többféle szoftveres és hardveres technológiával kellett megismerkednem. Jelen fejezet megírásával az a célom, hogy az olvasó képet kapjon a felhasznált szoftver- és hardveres közök fejlődéséről, valamint jelenlegi állapotukról; együttesen hogy megindokoljam, miért ezeket a technológiákat választottam a dolgozatom témáját feldolgozó alkalmazás megírásakor.

A fejezet 4.1. szakaszában az OpenCV gépi látás könyvtár, a 4.2. szakaszában pedig a Qt keretrendszer rövid, összefoglaló jellegű bemutatására vállalkozom. A fejezet utolsó harmadában a projekthez felhasznált webkamera kiválasztását, összeállítását, illetve előnyeit/hátrányait dokumentálom.

### 4.1. Az OpenCV könyvtár

Az **OpenCV**<sup>1</sup> egy nyílt forráskódú gépi látás (computer vision) könyvtár. Elsődleges célja, keretet nyújtani **válosidejű** képfeldolgozási alkalmazások fejlesztésére. A könyvtár szabadon letölthető és felhasználható a BSD licenc<sup>2</sup> keretein belül. [26]

Az OpenCV projekt hivatalosan 1999-ben indult az Intel kezdeményezésében. A nagyközönségnek a 2000. évi „*IEEE Conference on Computer Vision and Pattern Recognition*” konferencián mutatkozott be, majd öt béta-verziót követően 2006-ban jutott el az 1.0-ás hivatalos kiadásig. A fejlesztése itt úgy tűnt, hogy megáll, de végül a projektet a Willow Garage<sup>3</sup> robotikai kutatólabor vette szárnyai alá. Az ő irányításuk alatt 2008 októberében elkészült az 1.1-es verzióval közel egy időben látott napvilágot az első hivatalos OpenCV-val foglalkozó könyv „*Learning OpenCV: Computer Vision with the OpenCV Library*” címmel Gary Bradski és Adrian Kaehler fejlesztők tollából [27]. Az egy évvel később, 2009 októberében megjelent 2.0-ás verzióval a projekt nagy fejlődésen esett át. Ebben a verzióban található meg először



<sup>1</sup><http://opencv.org/>

<sup>2</sup><http://www.linfo.org/bsdlicense.html>

<sup>3</sup><http://www.willowgarage.com/>

a C++ és Python interfész (ez a meglévő C mellett már három hivatalosan fejlesztett interfészt jelentett), amely az egyszerűbb kezelhetőség, új függvények mellett a meglévő eljárások teljesítmény tekintetében – különösen többmagos rendszereken – jobb implementációját kínálta a felhasználóknak.

A projekt életében a következő mérföldkő 2012 augusztusa volt. Ekkor az OpenCV támogatását és fejlesztését az erre a cérla alapított *OpenCV.org* non-profit alapítvány vette át. A 2.5-ös verzió kiadásával tovább bővült a támogatott programozási nyelvek listája: innentől már a Java és a MATLAB/OCTAVE is felkerült az elérhető interfések listájára. Nem hivatalos támogatással, de további wrapper-ek is elérhetők a könyvtárhoz, például C# vagy Ruby nyelven, ezzel is elősegítve a széleskörű elterjedését.

Az elmúlt években a GPU-gyorsítás támogatása terén is komoly előrelépésekre került sor. 2010 szeptemberétől érhető el a CUDA, 2012 októberétől pedig az OpenCL interfész.

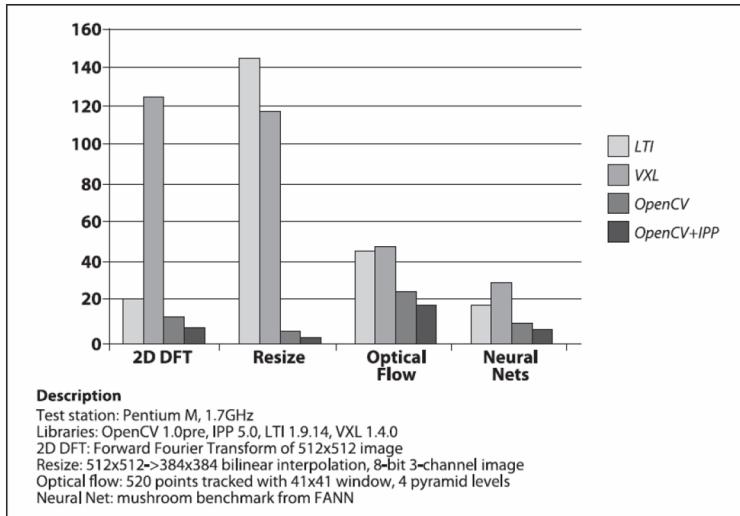
A fejlesztést különböző platformokra párhuzamosan (**cross-platform**) történik. Mivel a könyvtár alapvetően a C nyelvre épül, ezért a számos rendszeren működésre bírható. A dolgozat írásának idején elérhető 2.5-ös stabil verzió a következő operációs rendszerekre elérhető:

- asztali (desktop) rendszerek
  - Windows
  - Linux
  - OS X
  - OpenBSD
  - FreeBSD
- mobil operációs rendszerek
  - iOS
  - Android
  - BlackBerry 10
  - Maemo

Az OpenCV jelenleg elérhető 2.5-ös verziója széles körben, mondhatni világszerte használt, felhasználói tábora több, mint 47 000 főt számlál. Köszönhető ez többek között annak, hogy felhasználási lehetőségei igencsak sokrétűek: több, mint 500 optimalizált algoritmust kínál annak érdekében, hogy „ne kelljen újra feltalálnunk a kereket”. Sebesség tekintetében érződik a kipróbált, optimalizált algoritmusok használata: az OpenCV a jelenleg elérhető leggyorsabb alternatíva gépi látás terén (4.1. ábra). Teljesítménye azonban adott esetben még tovább növelhető, mivel ha Intel IPP<sup>4</sup> (Integrated Performance Primitives) támogatást észlel, az abban található szálakra optimalizált algoritmusok használatát fogja preferálni.

---

<sup>4</sup><http://software.intel.com/en-us/intel-ipp/>



**4.1. ábra.** Az OpenCV teljesítménye az LTI és VXL képfeldolgozási könyvtárakkal összehasonlítva.  
Forrás: [27]

A teljes funkcionálitás részletekbe menő bemutatása – mint láthattuk [27] – egy könyvet is megtölt, de a teljesség igénye nélkül tekintsük át, hogy milyen alapvető jellemzői és alkalmazásai vannak a környezetnek:

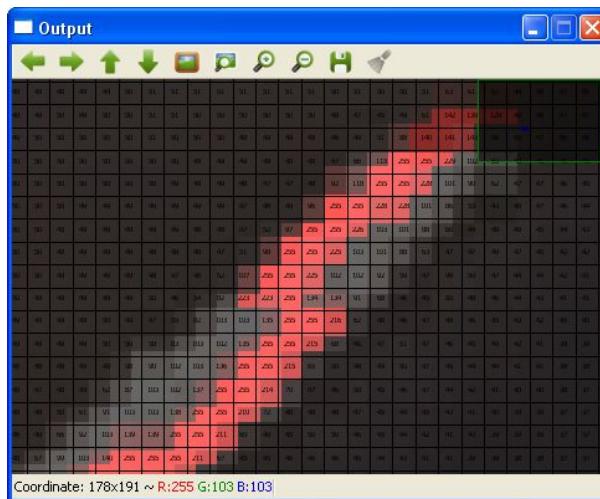
- alap adatstruktúrák
  - mátrixok, vektorok
- mátrix és vektor manipuláció, lineáris algebra
- dinamikus adatstruktúrák
  - listák, sorok, halmazok
  - gráfok és fák
- kép és videó input/output
  - beolvasás fájlból (kép vagy videó) és kameráról
  - kiírásí lehetőség képként vagy videóként
- előfeldolgozás
  - él- és sarokkeresés
  - mintavételezés és interpoláció
  - színkonverzió
  - morfológiai operátorok
- struktúraanalízis
  - távolság- és Hough-transzformáció
  - kontúrfeldolgozás

- sablonillesztés
- különböző momentumok
- Delaunay háromszögelés
- kamerakalibráció
  - kalibrációs mintázatok felismerése és követése
  - fundamentális mátrix becslés
  - homográfia becslés
  - sztereó megfeleltetés
- mozgásanalízis
  - optical flow
  - mozgásszegmentálás és -követés
- objektumfelismerés
  - eigen-módszerek
  - rejtett Markov-modell (Hidden Markov Model – HMM)
- kiterjesztett valóság-támogatás (Augmented Reality – AR)
- gépi tanulás könyvtár
  - döntési fa-alapú tanulás
  - klaszterezési algoritmusok (pl. k-szomszédság)
  - neurális hálózatok
  - Bayes-osztályozó
  - szsupport vektor gépek (Support Vector Machine – SVM)
- GUI és rajzolás
  - kép és videó megjelenítés
  - billentyűzet és egérkezelés
  - egyenes, kör, poligon, szöveg rajzolása

Láthatjuk, hogy a fent felsorolt funkciókkal a gépi látás terén rengeteg egyszerűbb feladatot szinte „egy lépésben”, beépített, optimalizált eljárások segítségével oldhatunk meg. Ha nagyobb szabású projektbe kezdünk, akkor is hasznunkra lehet, hogy részben vagy egészében egy több tízezres felhasználói tábor (melynek jelentős részét aktív kutatók alkotják) visszajelzései alapján fejlesztett környezetre építhetjük munkánkat.

Az OpenCV régebbi verzióiban a számos előny mellett hátrányként volt megemlítető, hogy segítségével a felhasználói felületet csak nagyon leegyszerűsített módon szabhattuk testre. Igaz, hogy a könyvtár feladata elsősorban a képfeldolgozás és

nem a megjelenítés, így ebből a nézőpontból a beépített kép és videó megjelenítési lehetőség, billentyűzet- és egérkezelés valamint trackbarok (csúsztatható kezelőszerv értékek beállítására) létrehozásának lehetősége inkább hozzáadott értékként jelent meg, azonban ez nem változtat azon a tényen, hogy ha igény van felhasználóbarát kezelőfelület készítésére, az OpenCV-t mindenkorban integrálnunk kellett valamely elterjedt grafikus felhasználói felület toolkittel.



**4.2. ábra.** Az OpenCV beépített Qt-alapú ablaka a 2.1-es verziótól kezdve.

Forrás: <http://bit.ly/YDigQ6>

A 2.1-es verziótól kezdődően megjelent egy némileg fejlettebb **Qt alapú felhasználói felület**, lásd 4.2.. Teljes szabadságot még ez sem ad a felületek kialakításában, de az egyszerűbb projektek kezelőszerveinek és megjelenítésének problémája már kielégítően elvégezhető egyéb eszköz használata nélkül.

## 4.2. A Qt keretrendszer



A **Qt**<sup>5</sup> (ejtsd az angol „cute” szó mintájára: kjút) keretrendszer egy széles körben használt szoftver, amelyet egyrészt grafikus felhasználói felületek (Graphical User Interface – GUI), másrészt parancssoros (Command Line Interface – CLI) alkalmazások fejlesztésére is használnak. A grafikus felhasználói felületek fejlesztése során a Qt úgynevezett widget-toolkitként funkcionál – kis, önálló elemek (widgetek) felhasználását és összekötését teszi lehetővé, ezek összességeből áll össze az elkészült grafikus felület. A szoftver a GPL v3<sup>6</sup> illetve LGPL v2<sup>7</sup> licencék szerint szabadon elérhető. [28]

A keretrendszer egyik nagy előnyének számít más rendszerekkel összehasonlítva, hogy a grafikus felületek kirajzolásakor minden operációs

<sup>5</sup><http://qt.digia.com/>

<sup>6</sup><http://www.gnu.org/licenses/gpl.html>

<sup>7</sup><http://www.gnu.org/licenses/lgpl-2.1.html>

rendszeren igyekszik az elérhető natív elemeket használni, ennek köszönhetően (a natív kirajzolásokból származó nagyszerű performancia mellett) a Qt-tal fejlesztett alkalmazások semelyik platformon nem tűnnek rendszeridegennek.

A Qt projekt első hivatalos kiadása a Nokia *Qt Development Frameworks* csoport égisze alatt látta meg a napvilágot, miután a finn mobilgyártó felvásárolta a technológiát a norvég Trolltech vállalattól. Ennek megfelelően a kezdeti években az asztali platformok mellett leginkább a Symbian operációs rendszer támogatása jelentette a fő csapásirányt. A Nokia azonban 2011-ben szakított saját fejlesztésű operációs rendszerével, és a további bizalmát a Microsoft akkori és elkövetkező mobilplatformjaiba (Windows Phone 7, majd 8) fektette.

Ezt követően a Nokia megvált a rendszer kereskedelmi jogaitól, és azok a Digia<sup>8</sup> vállalathoz kerültek, amely azóta is a projekt karbantartója, bár a tényleges fejlesztői munka nagy részét továbbra is a Nokia mérnökei végzik. A tranzakció után az új tulajdonos közvetlen célnak tűzte ki, hogy aktualizálják a Qt által támogatott platformok listáját az időközben népszerűvé vált rendszerekkel.

A dolgozat írásakor támogatott fontosabb operációs rendszerek:

- Windows (XP és 7)
- OS X
- Linux, FreeBSD, OpenBSD
- iOS
- Android
- QNX (BlackBerry 10)

A Qt-ot használó alkalmazások programozási nyelve alapvetően a sztenderd C++, a *Meta Object Compiler* (moc) kódgenerátorral kiegészítve, amely számos beépített makróval teszi könnyebbé a fejlesztést. Több más programozási nyelv is használható az ún. **kötések** (bindings) használatával. A Qt 4-es verziójához elérhető kötések listája a teljesség igénye nélkül:

- C# és .NET
- Java
- Python
- szkriptnyelvek
  - Perl
  - Ruby
  - PHP
  - Tcl

---

<sup>8</sup><http://www.digia.com/>

A Qt a GUI események célba juttatását a **signals/slots** rendszer segítségével végzi. A módszer az *Observer*<sup>9</sup> (Megfigyelő) tervezési minta megvalósítására nyújt egyszerű és szabványosított lehetőséget. A felhasználói felület elemei (widgetek) jelzéseket (signal) bocsátanak ki – pl. „a X gombot megnyomták”, „a beviteli mező értéke Y-ra változott” – amelyeket más elemek „foglalataihoz” (slot) köthetünk, ahol az eseményt lekezelhetjük. Látható, hogy ez a struktúra remekül illeszkedik a grafikus felhasználói felülete eseménykezelésére, de ezen kívül időzítések és értesítések kézbesítésére is alkalmas.

A signalok és slotok a megvalósításukat tekintve speciális függvények, melyeket deklarálásukkor a két bekezdéssel előbb említett *moc* (Meta Object Compiler) automatikusan generál, elkerülve ezzel a feleslegesen ismétlődő kód részletek elburjánzását (boilerplate kód).

A Qt 4-es verziójához nagyjából 6 éven át érkeztek a kisebb/nagyobb frissítések, egészen a 4.8-as kiadásig. Ezt követően a koncepcionális változásokat hozó **Qt 5** fejlesztése folytatódik tovább. Az 5-ös verzió funkciói között között a hardveres gyorsítású megjelenítés, a QML<sup>10</sup> és a JavaScript támogatása jelentik a legfontosabb újdonságokat.

A Qt 5 kiadásával a keretrendszer tett egy lépést a nyílt (open source) fejlesztés irányába: a **Qt Project**<sup>11</sup> néven futó projektbe már külsős fejlesztők is delegálhatják javításaikat vagy új funkciókat, amelyeket a Digia és a Nokia csapata elbírál, és ha megfelelőnek találtatnak, bekerülhetnek a soron következő verzióba.

#### 4.2.1. A Qt Creator fejlesztői környezet

A **Qt Creator** egy integrált fejlesztői környezet (Integreated Development Environment – IDE) Qt alapú fejlesztésekhez. Használatával jelentősen megkönnyíthető mind a grafikus felületek kialakítása, mind az alkalmazások forráskódjának szerkesztése és fordítása. [29]

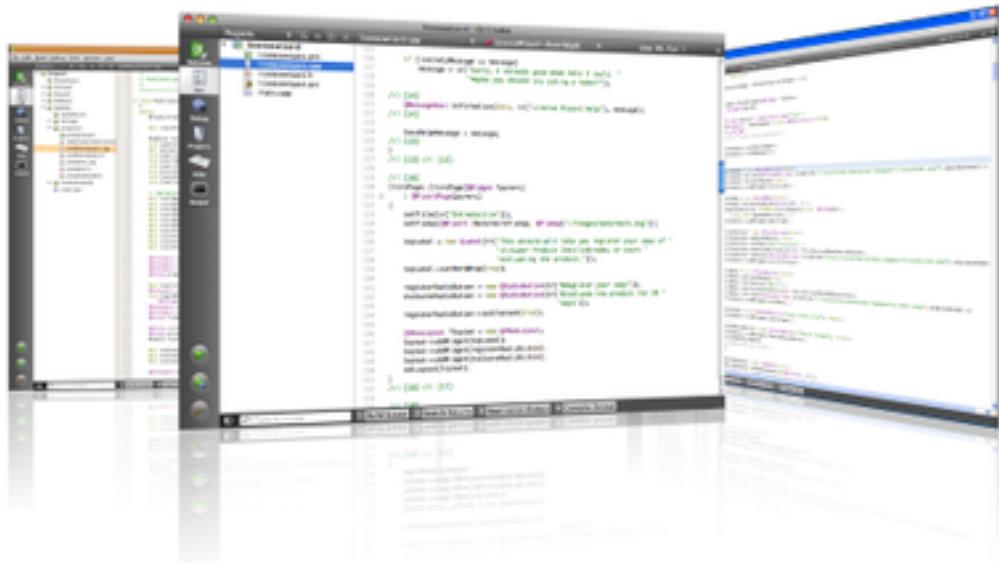
**Szerkesztő** A kódszerkesztő (editor) a kor követelményeinek megfelelően támogatja a szintaxis-kiemelést valamint az automatikus kódkiegészítést (code completion) több programozási nyelvre is. A szerkesztő hátrányának róható fel, hogy a megnyitott fájlokat nem a manapság elterjedt fülek (tab) használatával rendezi, hanem saját fejlesztésű listát használ erre a célra, amely más fejlesztői környezetekből érkezve kissé idegen.

**Qt Designer** A felhasználó felületek kialakításában nélkülözhetetlen segítséget nyújt a beépített Qt Designer eszköz. Használatával WYSIWYG (What You See Is What You Get) módon végezhetjük az alkalmazás grafikus felületének kialakítását. Választhatunk a számtalan beépített widget közül, de saját elemek létrehozását is támogatja a rendszer – a widgetek fejlesztési konvencióit betartva az egyedi elemek is grafikusan jelennek meg a szerkesztőfelületen.

<sup>9</sup>[http://en.wikipedia.org/wiki/Observer\\_pattern](http://en.wikipedia.org/wiki/Observer_pattern)

<sup>10</sup><http://en.wikipedia.org/wiki/QML>

<sup>11</sup><http://qt-project.org/>



**4.3. ábra.** A *Qt Creator* kezelőfelülete.

Forrás: <http://bit.ly/13Re5B8>

A felületek integrálása a kóddal az 4.2. szakaszban említett signal-slot mechanizmus segítségével történhet, amelynek megvalósítása szintén nagyon felhasználóbarát módon sikerült.

**Qt Quick Designer** Ez az eszköz animációk fejlesztésére szolgál, amelyeket QML<sup>12</sup> nyelvű leírásukkal adhatunk meg. A Qt natíván támogatja a nagyfrekvenciás (60 FPS) animációk készítését és lejátszását, ezáltal a 2D vagy 3D animációk „simák”, akadás mentesek lehetnek.

**Verziókezelő-integráció** A fejlesztői környezet képes a legnépszerűbb verziókezelő rendszerek támogatására. A támogatott rendszerek között van a népszerű *Git* valamint *Subversion (SVN)*, de *CVS*, *Bazaar* vagy *Mercurial* kódázisunkat is kezelhetjük közvetlenül a Qt Creatorrel.

**Példakódok** Az alkalmazás nyitóképernyőjén rögtön számtalan példaprogram fogadja a felhasználót. A példák százas nagyságrendben érhetők el, és a legegyszerűbb feladatoktól („Hello World”) az egészen összetettekig próbálják lefedni a fejlesztés során esetlegesen felmerülő problémákat.

Letöltésük nem igényel többet néhány kattintásnál, és integrált megoldás lévéni a kód betöltődik a szerkesztőbe – azonnal futtathatjuk és elkezdhetjük megérteni a működését. Maga az ötlet, és a megvalósítás is példaértékű lehet más fejlesztőkörnyezetek számára, igazi kincsesbányát jelent a Qt működésében elmélyedni kívánó fejlesztők számára.

<sup>12</sup><http://en.wikipedia.org/wiki/QML>

## 4.3. Fejkamera

A tekintetkövető rendszer fejlesztése közben hardveres technológiai kérdések is felmerültek. A legfontosabb ilyen kérdés a pupillát követő kamera kiválasztása és elhelyezése volt.

### 4.3.1. Eszközök

Költséghatékonyiségi okokból – ezzel együtt demonstrálandó, hogy nem feltétlenül kellenek speciális eszközök működőképes tekintetkövető összeállításához – a 4.4. ábrán látható **6 LED-es webkamerát**<sup>13</sup> választottam.



4.4. ábra. A fejlesztés során felhasznált webkamera.

A kamera gyártója ismeretlen (no-name), viszont jelen dolgozat írásának idején is mindössze 5 amerikai dollárért – 2013 májusi árfolyamon nagyjából 1 200 forintos áron – beszerezhető a lábjegyzetben szereplő hivatkozást meglátogatva.

A webkamera főbb paraméterei:

- **videó mód:** színes, 24-bit
- **érzékelő:** 1/4 CMOS

---

<sup>13</sup>megvásárolható: <http://bit.ly/15yoLJ3>

- **látószög:** 56°
- **felbontás:** 640x480 képpont
- **képfrissítés:** 30 FPS

Az átlagos webkamerák között egyedi tulajdonsága, hogy a **lencséje cserélhető**, helyére bármilyen szabvány *M12x0.5* (CCTV biztonsági kamerák körében elterjedt) foglalattal rendelkező lencse beilleszthető. Érdemes is megvásárolni a 4.5. ábrán látható **lencsekészletet**<sup>14</sup>, amely 2,8-tól 16 mm-es fókusztávig tartalmaz lencséket. A cserélhető lencséknek köszönhetően kiválaszthatjuk azt a fókusztávot, amely a kamera elhelyezésének függvényében a legjobb képet adja, azaz a szemrégió a lehető legjobban kitölti a képet.



4.5. ábra. Lencsekészlet M12x0.5 CCTV foglalattal.

A webkamera nagy hátrányaként említhető meg, hogy működőképes meghajtóprogramot (driver) **csak Windows 7 rendszerre** lehetséges találni hozzá, azt is csak rengeteg kerestőkérés után. Az eszközök azonos külső mellett többféle vezérlővel forgalmazzák, ezek meghajtóprogramjai egymással nem kompatibilisek. Ha kamerához csomagolt CD elveszik, megsérül, vagy – mint ahogyan sajnos a szerző esetében is előfordult – egyáltalán nincs is a csomagban, ember legyen a talpán, aki megtalálja az eszközhöz tartozó drivert.

Azonban ha sikerül is, a kizártlagos *Windows 7*-kompatibilitás miatt nem használhatjuk ki a 4.1. szakaszban bemutatott OpenCV, és a 4.2. szakaszban bemutatott Qt cross-platform mivoltát.

### 4.3.2. Módosítások, elhelyezés

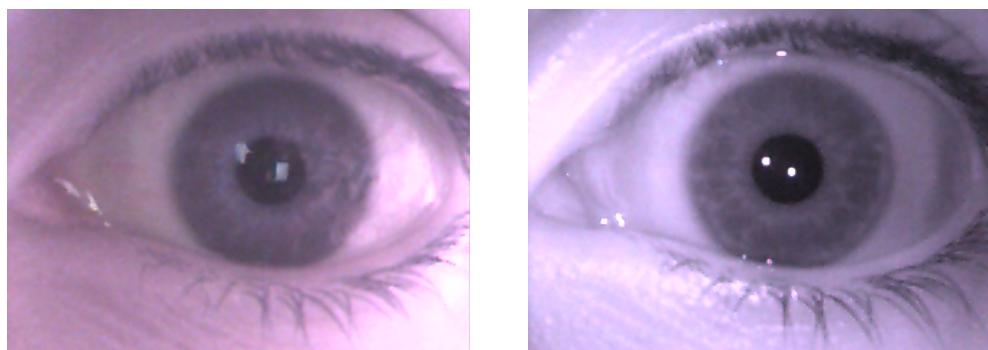
Az előző szakaszban bemutatott webkamera specifikációi között nem tüntetik fel, így szerencsés véletlennek tekinthető, hogy a kamera – valószínűsíthetően a gyártó költséghatékonyiségi törekvései miatt – **nem tartalmaz infratükröt** a CMOS érzékelő elé építve.

<sup>14</sup>megvásárolható: <http://bit.ly/17TtCCg>

Az infratükröt (amely nevéből adódóan visszaveri a beérkező infravörös tartománybeli fényt) a közép- illetve prémiumkategóriás webkamerák mindegyikébe beépítik, mivel használatával robusztusabb képmínőség érhető el infrafényben gazdag megvilágítás esetén is.

A tükr hiányának jelen esetben viszont maximálisan pozitív következménye van: lehetőség nyílik a szemrégió infra-megvilágítására, amelynek előnyeit a 2.3. szakaszban tárgyaltam.

A webkamera műanyag borítását leszerelve tudtam hozzáérni a nyomtatott áramkörhöz. A 6 db. LED-ből kettőt kiforrasztottam a helyéről, majd megfelelő fizikai méretekkel és elektronikai paraméterekkel rendelkező **infra LED-ekkel helyettesítettem** őket. Az új LED-ek beforrasztása és a kamera összeszerelése után a fennmaradó 4 normál LED-et letakartam, hogy fényük ne zavarja az infra-megvilágítást.



**4.6. ábra.** A szemrégió látható- (bal) és infrafényben (jobb)

A művelet elvégzése előtt és után készült képek a 4.6. ábrán láthatók. A várakozásaimnak megfelelően az infrafénnyel megvilágított szemrégió jóval kevésbé (gyakorlatilag elenyészően kis mértékben) érzékeny a látható fény tartományába eső megvilágítási egyenlőtlenségekkel, tükröződésekkel szemben. Igény szerint a kontraszt tovább növelhető a meglévő kettőnél több LED infrára cserélésével.

Végezetül a webkamera elhelyezéséről kellett gondoskodnom. A problémát úgy oldottam meg, hogy a kamerát egy bézból sapka napellenzőjéhez rögzítettem a kamerán található csíptető segítségével, a kamera kábelét pedig több ponton a sapkához rögzítve hátrafelé elvezettem, majd a számítógéphez egy USB hosszabbítón keresztül csatlakoztattam. Az elkészült összeállítás a TODO ábrán látható.

##### TODO sapkas kep

Ebben az összeállításban a szem elé függesztett kamera a látótér meglehetősen nagy százalékát takarja ki, de a rendszer működését ez alapvetően nem befolyásolja, legfeljebb kis megszokást igényel. Bemutató célokra ez a kényelmetlenség elfogadható mértékű, emellett könnyen belátható, hogy speciálisan a feladatra szabott hardverrel a lencse–érzékelő–kamera hármas akár körömvízi területen is elhelyezhető lenne. Ugyanígy léteznek technológiák a jelenlegi kábeles összeköttetés vezeték nélküli kiváltására is.

## 5. fejezet

# Megvalósítás I.

Megismerkedve a szükséges elméleti alapokkal, valamint kiválasztva a felhasználandó szoftveres és hardveres technológiákat, elkezdtettem a tekintetkövetés megvalósítását. A rendszer fejlesztésének első fázisában nem kezdtettem bele rögtön komoly architektúra vagy grafikus felhasználói felület tervezésébe. Mielőtt ugyanis ezekre sor kerülhet, fontos döntések meghozatalára, alap algoritmusok fejlesztésére, majd a helyes működés igazolására van szükség. Ezekről a döntésekről, algoritmusokról és validációról szól dolgozatom 5. fejezete.

A fejezet 5.1. szakaszában arra keresem a választ, hogy 3. fejezetben ismertetett módszerek közül melyikkel lehetséges a leginkább pontos és robusztus pupillakövetés megvalósítása. A kiválasztott módszerre alapozva a feldolgozási folyamat részeként két, önmagában is fontos algoritmus kifejlesztésére volt szükség: a 5.2.1. szakaszban a pupillakövetési, míg a 5.2.2. szakaszban a kalibrációs algoritmus részletes bemutatását találhatja az olvasó. Az implementáció első részének lezárása előtt, az 5.3. szakasz a követés validációjáról szól, az 5.4. szakaszban pedig gyakorlati problémákon mutatom be a rendszer működőképességét.

### 5.1. Módszerek összehasonlítása

A 3. fejezetben három módszer elméletét mutattam be, amelyekkel lehetséges a pupilla felismerése a kameraképen, ami az optikai elvű tekintetkövetés alapjául szolgál. Mindhárom követési elvet kipróbáltam működés közben, ebben a szakaszban a módszerek előnyeit és hátrányait foglalom össze.

**Hough-transzformáció** A transzformáció bemenetére egy élképet kell juttatnunk, ezt feldolgozva tudja az algoritmus az ismertetett szavazási eljárással megtalálni a legvalószínűbb körök paramétereit a képen. A módszer használata során a következő előnyökkel és hátrányokkal találkoztam.

*Előnyök:*

- az előfeldolgozási lépések megvalósítása OpenCV környezetben meglehetősen **egyszerű**
- maga a Hough-transzformáció egyetlen függvényhívással (`cv::HoughCircles`) elvégezhető

*Hátrányok:*

- a kép masszív pixel-szintű **előfeldolgozást igényel**, a képet terhelő zajok nagy mértékben hozzájárulhatnak egynél több kör megtalálásához, ezek között nehéz különbséget tenni
- a **módszer sebessége** hagy némi kívánnivalót maga után: minden képkockán feldolgozást végezve a sebesség 6-7 képkocka másodpercenként
- a pupilla formája is gyakran gondot okoz a felismerésben, a Hough-transzformáció OpenCV-beli implementációja csak körök keresésére alkalmas

**Objektumdetektálás és -követés** A Viola–Jones objektumdetektor kipróbálásához jóval nagyobb látószögű objektívet használtam, mint a másik két esetben. A pupillapozíció becsléséhez az OpenCV-ben megtalálható szemrégiókra betanított osztályozót használtam. Az objektumdetektálást csak minden huszadik képkockán végeztem el, a közbülső képkockákon Lucas–Kanade optical flow eljárással követtem az előzőleg felismert pont mozgását.

*Előnyök:*

- az objektumdetektálás és az optikai áramlás eljárás kettősének felhasználásával a Hough-transzformációhoz képest jelentősen **jobb feldolgozási sebességet** – azonos számítógépen átlagosan 15 képkocka másodpercenként – lehetett elérni

*Hátrányok:*

- a Viola–Jones objektumdetektor a használt konfigurációban a szemrégió detektálására volt kihegyezve, a pupilla régióban belüli pozíciójának meghatározása **nem elég pontos**
- igaz, én most előre betanított osztályozót használtam, de egyéb esetekben az **osztályozó betanítása** meglehetősen összetett feladat
- az objektumdetektálás pozíció-hibáját az optical flow legjobb esetben is továbbviszi, de az újabb detektálási pontig tipikusan még több hiba rakódik a detektált pozícióra

**Blob alapú követés** A Hough-transzformáció kipróbálása során azt a következetést tudtam levonni, hogy a pixelszinten végzett előfeldolgozási lépések (pl. bináris morfológia, simítási eljárások) viszik el a szükséges számítási teljesítmény jelentős részét. Minél kevesebb előfeldolgozást kell alkalmazni, annál több kapacitást tudunk a tényleges feldolgozás elvégzésére szálni. Nem is beszélve arról, hogy a kép egészére vonatkozó feldolgozási lépések számításigénye természetesen a kamerakép felbontásának növekedésével együtt növekszik!

*Előnyök:*

- **kevés előfeldolgozást** igényel relatíve zajos képen is a pupilla foltja blob-jellemzők szűrésével meghatározható
- a kevés előfeldolgozási lépés, valamint a kontúrkeresés gyorsasága miatt ez a módszer nyújtott a **legjobb sebességet** – a webkamera maximális 30 képkoc-kás másodpercenkénti frissítését az algoritmus valós időben követni tudta
- az algoritmus végén pupillát egy, a pupillakontúrra nagyon jól illeszkedő **ellipszissel** jellemzhetjük, így nagyon fontos lapultsági, orientációs információkat is kinyerhetünk, ami a tekintetkövetés szempontjából fontos lehet

*Hátrányok:*

- hátrányként említhető a másik két módszerhez képest **bonyolultabb implementáció** – az eljárásból „készen” csak az előfeldolgozási műveletek, valamint a kontúrkeresési eljárás érhető el, a blob-jellemző alapú szűrést egyedileg kell implementálni

Az 5.1. táblázatban foglaltam össze a módszerek összehasonlítása során szerzett tapasztalataimat. A táblázat adataiból látható, hogy a leggyorsabb feldolgozást és a legnagyobb pontosságot a három módszer közül a blob-alapú követés nyújtja, ezért a választásom erre esett.

**5.1. táblázat.** *Pupillakövetési módszerek összehasonlítása.*

	pontosság	sebesség
<b>Hough-transzformáció</b>	zajra érzékeny, csak körök illesztése	~6-7 FPS
<b>objektumdetektálás</b>	pontatlan	~15 FPS
<b>blob alapú követés</b>	pontos ellipszis illesztése	~30 FPS

## 5.2. Feldolgozási folyamat

A tekintetkövetés működéséhez szükséges képfeldolgozási illetve számítási folyamat kidolgozása során két fő feladatra kellett robusztus megoldást találnom: a pupilla követésére, valamint a pupilla-pozíció képernyő-pozícióba történő leképezésére. Mindkét feladat önmagában van annyira összetett, hogy érdemes őket kitüntetett figyelemmel kezelni.

A pupillakövetést az előző szakaszban írtaknak megfelelően több módszerrel is teszteltem, a végül kiválasztott blob alapú követési algoritmust azonban szeretném részletesebben is bemutatni a 5.2.1. szakaszban. A kalibrációt már a stabilan működő pupillakövetésre építettem fel, a felhasznált módszer bemutatására szenteltem dolgozatom 5.2.2. szakaszát.

### 5.2.1. Pupillakövetés

A pupillakövetés megbízható működéséhez meg kellett találnom azon elemi képfeldolgozási műveletek megfelelő sorrendjét és paraméterezését, amelyekkel lehetővé

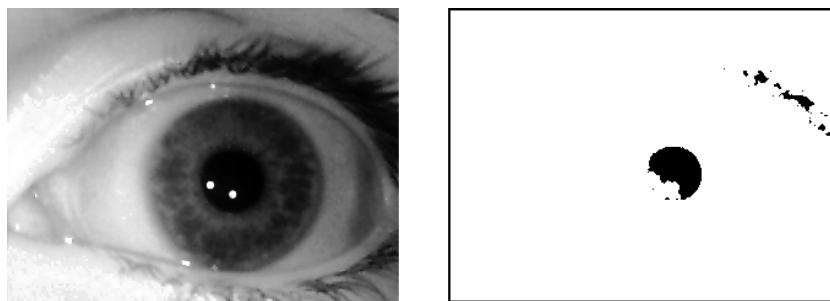
vált a pupilla azonosítása. A pontos folyamat kialakítása nagyrészt intuíciók alapján, a részmegoldások folyamatos tesztelésével történt.

Az előfeldolgozási műveleteket az OpenCV készten kínálja, azonban a függvények száma és paraméterezési lehetőségei miatt rengeteg alternatíva jöhetett szóba. A szakasz további részében a kiválasztott képfeldolgozási műveleteket szeretném dokumentálni, a folyamat egészének tekintetében lényegtelen részletek mellőzésével. A felhasznált függvények pontos paraméterezése megtalálható az F.1. függelékben.

A szemrégió képét közvetítő webkamera a gyári orientációjához képest fejjel lefelé került felfüggesztésre. A könnyebben értelmezhető megjelenítés érdekében ezért *első lépésként megtükröztem* a képet a vízszintes középtengely mentén a `cv::flip` függvény használatával. Jegyezzük meg azonban, hogy ez a lépés csak azért szükséges, hogy a felhasználó számára emészthetőbb, „álló” képet tudjunk megjeleníteni az alkalmazás felületén. A kép tükrözött voltát a feldolgozás során végig figyelembe véve ez a lépés akár el is hagyható!

A pupillakövetés szempontjából a színinformációnak nincs jelentősége, ezért a *második lépés* a kép **szürkeárnyalatossá konvertálása** a `cv::cvtColor` metódus használatával.

A *harmadik lépésben hisztogram-kiegyenlítést* végeztem a képen az OpenCV `cv::equalizeHist` függvénye használatával. A kiegyenlítés eredményeképp a kép minden megvilágítási körfülmény között kellően kontrasztos lesz, megkönnyítve ezzel a pupilla detektálását.



**5.1. ábra.** Hisztogram-kiegyenlítés és küszöböözés

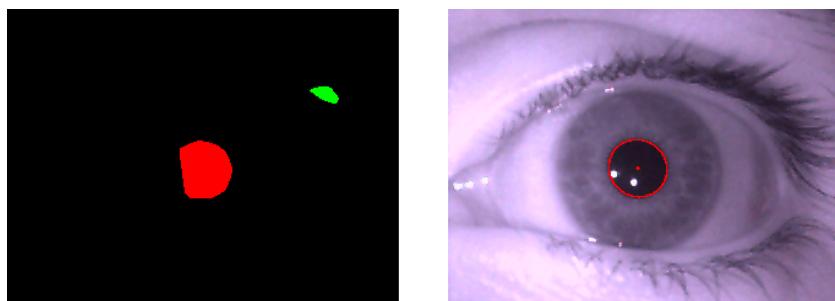
A szürkeárnyalatos képet ezek után a *negyedik lépésben küszöböözni* kellett, ehez a `cv::threshold` függvény használatára volt szükség. A pontos küszöbérték beállítása általában nehéz feladat, minden megvilágításra működő univerzális érték megtalálása pedig sokszor nem is lehetséges. A 4.3.2. szakaszban részletezett infráfényforrás megvalósításával azonban abba a szerencsés helyzetbe kerülttem, hogy a külső megvilágítás hatását gyakorlatilag sikerült teljesen kikiüszöbölni: a tesztelés során végül meghatározott ideális küszöbérték a verőfényes napsütéstől kezdve a vaksötét szobáig minden esetben megfelelően binarizálja a szürkeárnyalatos képet.

A fenti négy lépés elég a kamerakép előfeldolgozásához, az *ötödik lépéstől* kezdve a tényleges felismerés megvalósítása következhet. Az előző, 5.1. szakaszban kifejtettem, hogy a feldolgozás sebességét döntően befolyásolja a pixelszinten végzett feldolgozási lépések komplexitása és száma. A előfeldolgozás végén a kép még meglehetősen nagy zájjal terhelt: látható a pupilla viszonylag nagy, ovális foltja, azonban például a szempillák foltja zavaró hatásként jelentkezik.

Sebesség szempontjából azonban sokkal jobban járunk, ha nem is próbáljuk az előfeldolgozási lépések után hátramaradó zajt pixelszintű eljárásokkal kiszűrni (pl. bináris morfológia). Ehelyett az előfeldolgozott képre rögtön ráereszthetünk egy hierarchikus **kontúrkeresési eljárást** a `cv::findContours` függvény meghívásával. A további lépések során elegendő a hierarchia legfelső szintjét használnunk: mivel a megtalálni kívánt pupilla konvex, a blobok belsejében található lyukak elhelyezkedése nem szolgál többletinformációval.

*Hatodik lépésben* meg is szabadulhatunk a **túl kis méretű objektumoktól**, mivel ezek nagy valószínűséggel csak mint zaj vannak jelen a képen. Egy kontúrpontjaival megadott objektum területét a `cv::contourArea` függvény segítségével határozhatjuk meg, ez alapján már kiszűrhetők a „túl kicsi” objektumok.

A kisméretű zajok eltávolítása után jó eséllyel csak néhány, nagyobb területű blobunk marad, köztük a megtalálni kívánt pupilla. A fennmaradó blobok között jó szűrési feltételeként adja magát a blobok **körkörössége** vizsgálata. A *hetedik lépésben* ezért a 3.3.2. szakaszban ismertetett módon kiszámolom a foltok körkörösségi faktorát, majd ezek közül a legerősebbet kiválasztva nagyon jó százalékban sikerül a pupilla azonosítása.



**5.2. ábra.** Blobok és a pupilla ellipszise

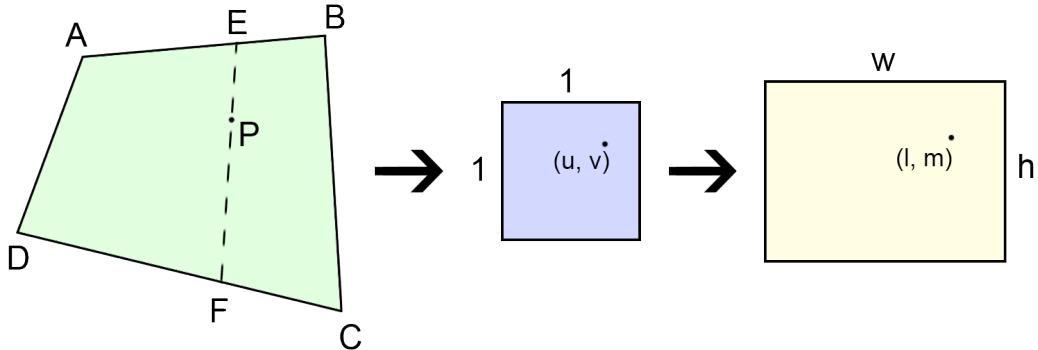
Nincs más hátra, mint a *nyolcadik lépésben* **ellipszist illeszteni** a pupilla kontúrára, ehhez használható a `cv::fitEllipse` függvény. A pupillát a ráillesztett ellipszis paramétereivel lehet jellemzni: középpontjával, kis- és nagytengelyének hosszával, valamint nagytengelyének irányszögével.

A pupillakövetésnél gondolni kell azokra az esetekre is, amikor a követés fizikailag nem megvalósítható. Csukott szemnél a követés nyilvánvalóan lehetetlen, de az pislogási reflex okozta anomáliákat jó lenne kiszűrni. A detektálás közben felismerhető, ha egyáltalán nincs értékes pupilla-jelölt a blobok között. Abban az esetben, ha a fent vázolt algoritmus nem találja a pupillát, **öt egymást követő sikertelen felismerésig** még megtartja az utoljára detektált pupillapozíciót. Ezzel a módszerrel a pislogás okozta rövid kiesések szinte teljesen kiküszöbölhetők, és a valóban csukott szem detektálása is csak néhány századmásodperces késleltetést szenved.

### 5.2.2. Kalibráció, leképezés

A pupillakövetés mellett a másik, összetettebb algoritmus használatát igénylő feladat a **tekintet kalibrációja** volt, annak érdekében, hogy a kameraképen detektált pupillapozíció és a képernyón nézett pont között kapcsolatot teremthessek.

A kalibráció során a képernyő négy sarkában rögzítem a pupilla pozícióját, majd minden képkockán ezek között a pontok között interpolálva számítom ki a képernyőn nézett pont koordinátait.



**5.3. ábra.** A  $P$  pupilla-középpont leképezése képernyő-koordinátákba.

A klasszikus **bilineáris interpoláció** estén az egységnégyzet  $u = [0, 1]$  és  $v = [0, 1]$  koordinátáit szeretnénk egy tetszőleges négyszögbe leképezni. A 5.3. ábrán látható módon definiáljuk a négyszöget  $A, B, C$  és  $D$  sarkaival. Belátható, hogy a  $P$  pont koordinátáit a következő módon kaphatjuk meg:

- $E$  pont lineáris interpolációval adódik  $A$  és  $B$  között
- $F$  pont lineáris interpolációval adódik  $C$  és  $D$  között
- a  $P$  pont szintén lineáris interpolációval adódik  $E$  és  $F$  között

A leképezési feladat megoldása során a fenti számítást kell „visszafelé” elvégezni: a sarokpontok (pupillapozíciók a képernyő sarkain mérve) mellett szintén adott  $P$  ponthoz (a pupilla aktuális képkockán vett pozíciója) kell meghatározni annak  $u$  és  $v$  koordinátáit. Az  $u$  és  $v$  koordináták, valamint a képernyő szélességének és magasságának ( $w$  és  $h$ ) ismeretében meghatározhatók az éppen fókusztált pont képernyő-koordinátái,  $l$  és  $m$ .

A felírandó rendszer ugyan nem lineáris, de egyszerűen, analitikusan megoldható. Az  $u$  koordináta az

$$u = \frac{-b - \sqrt{b^2 - 4 \cdot a \cdot c}}{2 \cdot a} \quad (5.1)$$

másodfokú egyenlet megoldásaként számítható, ahol az együtthatókat a következő kifejezések adják meg:

$$\begin{aligned} a &= (B_y - A_y) \cdot (C_x - D_x) - (B_x - A_x) \cdot (C_y - D_y) \\ b &= (A_y - P_y) \cdot (C_x - D_x) + (B_y - A_y) \cdot (D_x - P_x) - \\ &\quad - (A_x - P_x) \cdot (C_y - D_y) - (B_x - A_x) \cdot (D_y - P_y) \\ c &= (A_y - P_y) \cdot (D_x - P_x) - (A_x - P_x) \cdot (D_y - P_y) \end{aligned} \quad (5.2)$$

Az  $u$  koordináta ismeretében már meghatározhatók  $E$  és  $F$  pontok, a

$$\begin{aligned} E &= A + (B - A) \cdot u \\ F &= D + (C - D) \cdot u \end{aligned} \quad (5.3)$$

kifejezések kiszámításával. A hiányzó  $v$  koordinátát ezek után a

$$v = (P_x - E_x) / (F_x - E_x) \quad (5.4)$$

képlettel határozhatjuk meg. Az egységnégyzetet már csak a megfelelő méretre kell skálázni: az  $l$  és  $m$  képernyő-koordináták a következő triviális összefüggésekkel számíthatók.

$$\begin{aligned} l &= u \cdot w \\ m &= v \cdot h \end{aligned} \quad (5.5)$$

### 5.3. Validáció

Az alapvető algoritmusok önálló implementációját követően, de még az architektúra kialakítása, valamint a felhasználói felületek fejlesztésének megkezdése előtt szerettem volna igazolni, hogy a kifejlesztett pupillakövetési módszer valóban alkalmas a rá szabott feladat elvégzésére.

A validáció során méréseket végeztem a pupillakövető algoritmussal: a kijelző teljes területén sorban egy adott oldalhosszúságú négyzetháló pontjaira fókuszálva vettem fel a felismert ellipszisek paramétereit. A méréseket egy  $1280 \times 1024$  képpont felbonású monitorral végeztem el, a fej pozícióját álltámasszzal rögzítve (a mérési elrendezés pontos paraméterei a F.1. függelékben megtalálhatóak). A mérés során használt négyzetháló méretét és az ebből adódóan rögzített képpontok számát a 5.2. táblázat tartalmazza.

**5.2. táblázat.** A validáció során rögzített mérések adatai.

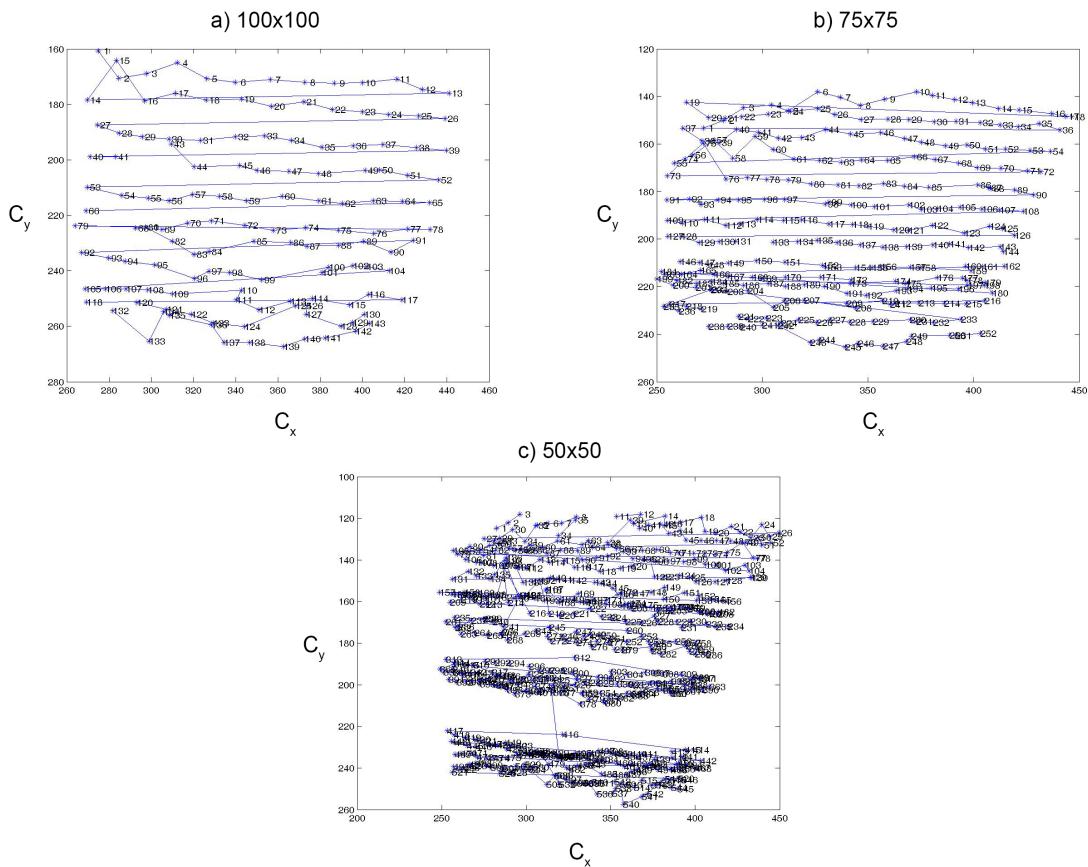
rács mérete	rögzített pontok száma
$100 \times 100$ képpont	143 db
$75 \times 75$ képpont	252 db
$50 \times 50$ képpont	546 db

Mindhárom rácsméret esetén a rácspontokon balról jobbra, fentről lefelé haladva minden mérési pontban a következő adatokat rögzítettem a tesztpontra és a felismert ellipszisre vonatkozóan:

- a tesztpont **képernyő-koordinátái** –  $T_x$  és  $T_y$
- az ellipszis **súlypontjának** pozíciója –  $C_x$  és  $C_y$
- az ellipszis **kis- és nagytengelyhossza** –  $k$  és  $l$

- az ellipszis irányszöge –  $\varphi$

Első lépésként ábrázoltam a tesztpontok koordinátáit a felismert ellipszis súlypontjának (középpontjának) koordinátái függvényében. Az eredmény a 5.4. ábrán látható, és a többnyire szabályos mintázat megjelenése jó kilátásokkal kecsegtetett a későbbi mérések, valamint a remélhetőleg helyes működés tekintetében is.



**5.4. ábra.** A tesztpontokon mért eredmények a súlypontok függvényében függvényében a) 100 b) 75 c) 50 képpontos rácstávolság esetén

A második lépésben a volt a célom, hogy meghatározzam, az ellipszis paraméterei közül melyek azok, amelyek a legszignifikánsabban vesznek részt a pupillapozíció és a képernyő-koordináta közötti kapcsolatban. A paraméterek „fontossági” értékeit korrelációs együtthatók számításával vizsgáltam a 5.3. táblázatban megadott bemenetekkel és eredményekkel.

Nem teljesen váratlan módon a legnagyobb korreláció minden esetben az ellipszis súlypontjának koordinátái és a tesztpont koordinátái között mutatkozott. Az együtthatók alapján bátran kijelenthető, hogy lehetséges a súlypont koordinátáira alapozni a leképezési algoritmust.

A leképzés továbbfejlesztése esetén a többi paramétert esetleg járulékos információként felhasználva (pl. a szemgolyó középpontjától kifelé haladva a pupillát reprezentáló ellipszis lapultságának mértéke és iránya determinisztikusan változik) a tekintetkövetés pontossága növelhető lehet.

**5.3. táblázat.** Korrelációs együtthatók,  $D = \sqrt{T_x^2 + T_y^2}$

	100×100	75×75	50×50
$R(T_x, C_x)$	0,9789	0,9587	0,9464
$R(T_y, C_y)$	0,9898	0,9893	0,9854
$R(D, k)$	-0,4386	-0,2427	-0,1193
$R(D, l)$	0,1603	0,1098	0,0776
$R(D, \varphi)$	-0,1537	-0,2431	-0,1092

## 5.4. Demonstráció

A számszerűsített validáció mellett – még mindig csak a lehető legegyszerűbben implementált pupillakövetési és leképezési algoritmusok felhasználásával – szükségét éreztem annak is, hogy megmutassam: a rendszer maximálisan valós problémák megoldására is alkalmas lehet.

Munkám 1. fejezetében többek között a pszichológiai és a webergonómiai felhasználási lehetőségekről ejtettem szót. A szakaszban további részében lássunk tehát e két témában egy-egy egyszerű kísérletet, a rendszer gyakorlati használhatóságát igazolandó.

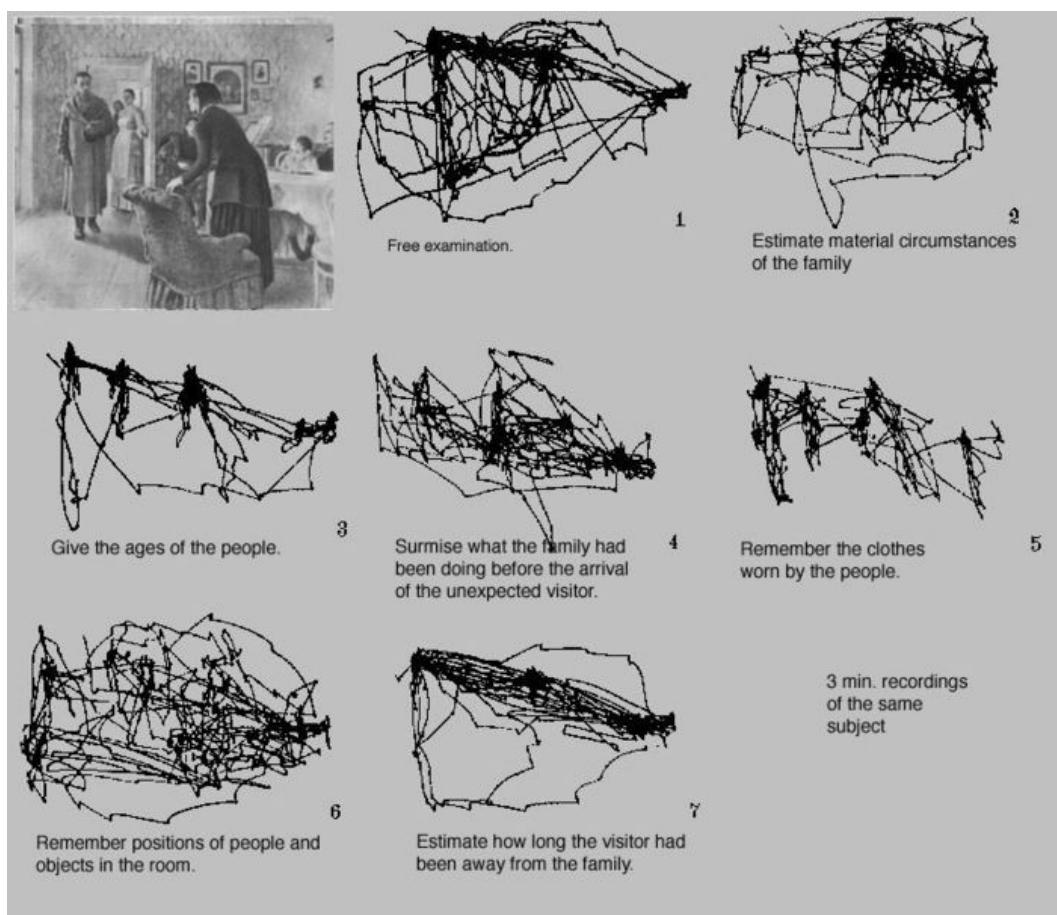
### 5.4.1. Pszichológiai bemutató

A rendszer magasabb szintű működését első esetben úgy próbáltam demonstrálni, hogy végrehajtottam Alfred L. Yarbus orosz pszichológus 1967-es tanulmánya egy részletét. A kísérletben a kutatók azt bizonyították be, hogy a tesztalanyoknak előzetesen különböző kérdéseket feltéve, a kérdések jelentősen befolyásolták egy kép részleteinek vizsgálatát ahhoz képest, ha csak „szabadon” nézegették azt. A teszteredményeket összefoglaló kép – mint az egyik első a témával foglalkozó eredmény – jól ismert a tekintetkövetéssel foglalkozók körében, ez látható a 5.5. ábrán.

Látható, hogy a kísérlet véghajtásához szükség volt a tekintet követésére. Ekkor még a kornak megfelelően nem álltak rendelkezésre kifinomult módszerek: az alanyokat egy meglehetősen kényelmetlen acélszerkezethez rögzítve vizsgálták. A bemutató alkalmazásomban arra kerestem a választ, hogy lehetséges-e hasonló méréseket elvégezni az általam fejlesztett rendszerrel.

Az vizsgálatban résztvevő alanyoknak a következő kérdésekre kellett válaszolniuk a tesztkép (Repin: Váratlan utazó) rövid vizsgálata után. A vizsgálatot minden kérdésben 200 beérkezett érvényes mérési pontig folytattam, ez a felhasznált webkamera, illetve a feldolgozási sebesség mellett nagyjából kérdésenként 15–20 másodpercret vett igénybe.

1. szabad nézelődés
2. „Milyen anyagi körülmények között él a család?”
3. „Adja meg az egyes szereplők életkorát!”
4. „Mit csinálhattak a szereplők, mielőtt az utazó betoppant?”



5.5. ábra. Yarbus '67-es kísérletének eredménye.

5. „Milyen ruhát viselnek a kép szereplői?”
6. „Próbáljon megjegyezni minél több strukturális részletet (személyek, tárgyak pozíciója)!”
7. „Mennyi ideig lehetett távol az utazó a családtól?”

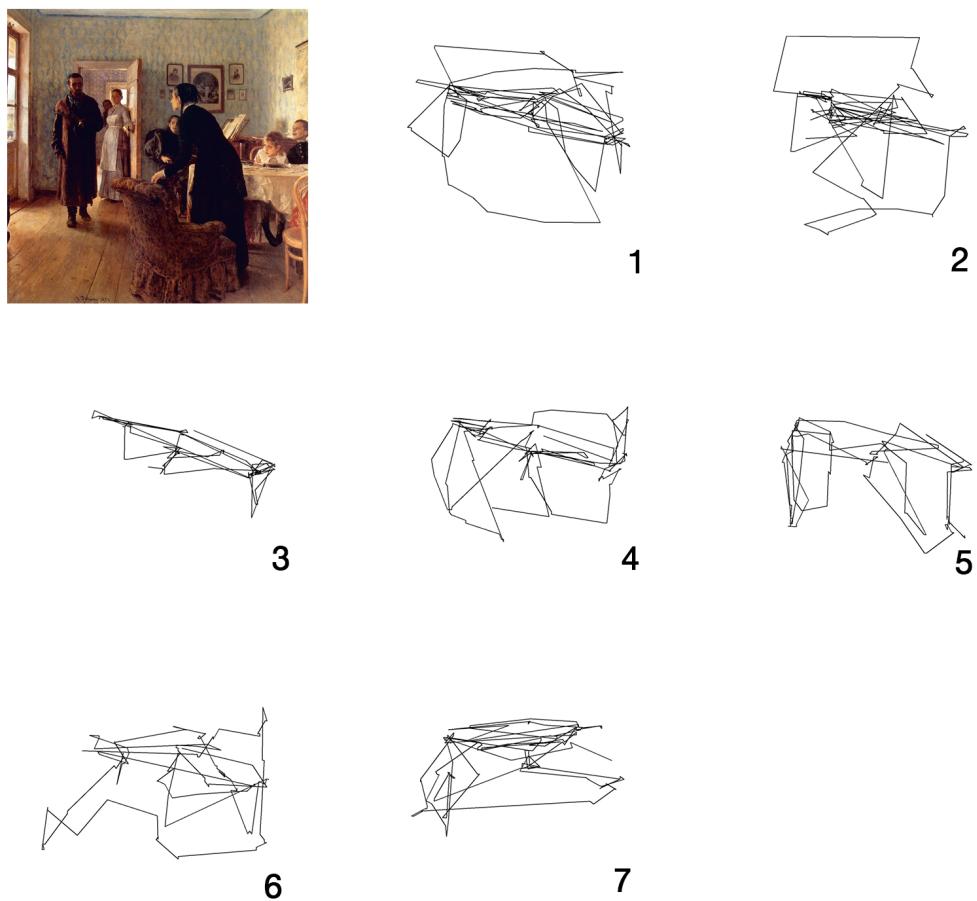
A kérdésekre nem volt jó, vagy rossz válasz, a kísérlet minden alanyánál csak a feladatnak megfelelő figyelmi területek változását vizsgálja. A vizsgálatom eredménye a 5.6. ábrán követhető.

Látható – bár ez nem volt feltétel – hogy a mérési eredmények ezen alany esetén meglehetősen jól fedik Yarbus tesztalanyának eredményeit. Az viszont mindenképpen kijelenthető, hogy szignifikáns különbség van az ábra első (szabad nézelődés), valamint többi része között.

Természetesen nem tudom, és nem is célom a kísérlet pszichológiai eredményeit értékelni, az viszont a kapott eredményekből leszűrhető, hogy a rendszer hasonló kísérletek elvégzését nagy valószínűséggel támogatja.

#### 5.4.2. Webergonómiai bemutató

Ahogyan dolgozatom 1. fejezetében már említettem, a tekintet követése a webergonómia területén is fontos kísérletek elvégzésére ad lehetőséget. A rendszer ilyen



**5.6. ábra.** Eredmények a saját rendszerrel

irányú képességeit demonstrálandó, elkészítettem az pszichológiai kísérlet adatait hőterképen (heatmap) megjelenítő funkciót is, ennek eredménye egy tesztképre a 5.7. ábrán látható.

A webergonómia terén sokszor az értékes információ nem a tekintet dinamikus változásában keresendő (bár természetesen az is érdekes vizsgálatok tárgya lehet). A teljes vizsgálati periódus alatt aggregált értékek alapján például a kép egyes területeinek „érdekességét” tudjuk meghatározni,

A hőterképeken más-más színekkel jelöli a kép egyre érdekesebb régióit. A tesztképen én a legelterjedtebb (valóban hőmérsékleti adatok reprezentálásánál megszokott) megjelenítési módot választottam, azaz egyre „forróbb”, pirosabb színnel jelöltem a kép frekventáltan vizsgált területeit.

A tesztképet nézve leszűrhető az eredmény: a rendszerrel viszonylag kevés erőfeszítés árán kinyerhető a nyers tekintetkövetési adatok olyan reprezentációja, amely az aktuális feladat szempontjából a leginkább látványos és hasznos megjelenítést biztosítja.



**5.7. ábra.** Hőterképes megjelenítés a pszichológiai teszt egy adathalmázára („Adja meg a szereplők életkorát!”)

## 5.5. Összefoglalás

A megvalósítás első fázisában – ahogy a fejezetben olvashattuk – sikerült lefektetnem a tekintetkövető rendszer alapjait. Választ kaptam a kérdésekre, hogy

- melyik pupillakövetési módszert célszerű használnom?
- milyen módon végezzem a rendszer kalibrációját?
- alkalmas-e a rendszer mérések, és kísérletek alapján a kiszabott feladat elvégzésére?

A pupillakövetési és kalibrációs algoritmusok kidolgozása a fent említett kérdések megválaszolásán kívül sem volt hiábavaló. A tekintetkövetés alapját jelentő módszerek implementációja szinte egy az egyben felhasználható a végleges alkalmazásban.

## 6. fejezet

# Megvalósítás II.

A megvalósítás második szakasza során csak az **objektumorientált programozás** (object-oriented programming – OOP) merült fel mint felhasználható módszertan. Az általánosan elterjedt koncepció lehetővé teszi egy komplex probléma kellően intuitív megfogalmazását és átlátható leírását. Erre a fejlesztés során szükség is volt, hiszen a meglehetősen összetetté váló programot csak gondos tervezéssel és kivitelezéssel lehetett megfelelő minőségben előállítani.

Az alkalmazás fejlesztését tehát az OOP paradigmáinak szem előtt tartásával végeztem. Az ismert alapelvek néhány szóban:

- **absztrakció (abstraction):** a probléma valós világbeli objektumok mintájára történő modellezése, a lényegtelen részletek elhagyásával, a lényeges tulajdonságok kiemelésével
- **egységezés (encapsulation):** az egyes osztályok és objektumpéldányok saját maguk rendelkezzenek a futások során szükséges adatok és metódusok felett
- **öröklődés (inheritance):** lehetséges egy általános ősosztály tetszőleges kiengészítése, specializálása a közös elemek megtartásával
- **polimorfizmus (polymorphism):** az egyes műveleteket képesek legyünk bemeneti paraméterek széles skáláján végrehajtani (metódus- vagy operátor-felüldefiniálás)

A fejezet 6.1. szakaszában összefoglalom a feladat specifikációját, majd a 6.2. szakaszban részletesen dokumentálom a rendszer architektúrájával és implementációs részleteivel kapcsolatos tudnivalókat. A 6.3. szakaszban a grafikus felhasználói felületet mutatom be, a 6.4. szakasz pedig a felhasználói dokumentációnak ad helyet. A fejezet lezárásaként a 6.5. szakaszban a rendszer kipróbálása során megfigyelt eredményeket és általános tapasztalatokat foglalom össze.

### 6.1. Követelmények

Mielőtt a fejezet későbbi szakaszaiban részletekbe menőben bemutatom az elkészült alkalmazás architektúráját, illetve osztályainak pontos működését, szeretném dokumentálni a kész programra vonatkozó követelményeket, azaz specifikációt.

Az alkalmazás legyen képes:

- a) valósidejű **videofolyamot** olvasni a számítógépre kötött webkameráról (lásd 4.3. szakasz)
- b) a videofolyam feldolgozásával a **pupillapozíció meghatározására** a 5.2.1. szakaszban leírtak szerint.
- c) a tekintet **kalibrációjára** a 5.2.2. szakaszban ismertetett módszer felhasználásával.
- d) **munkamenetek** (session) rögzítésére a képernyő tartalmának mentésével együtt.
- e) a korábbi munkamenetek listázására, törlésére.
- f) munkamenetek **visszajátszására**.
- g) a munkamenetek alapján **hőterképek** generálására (lásd 5.4.2. szakasz)

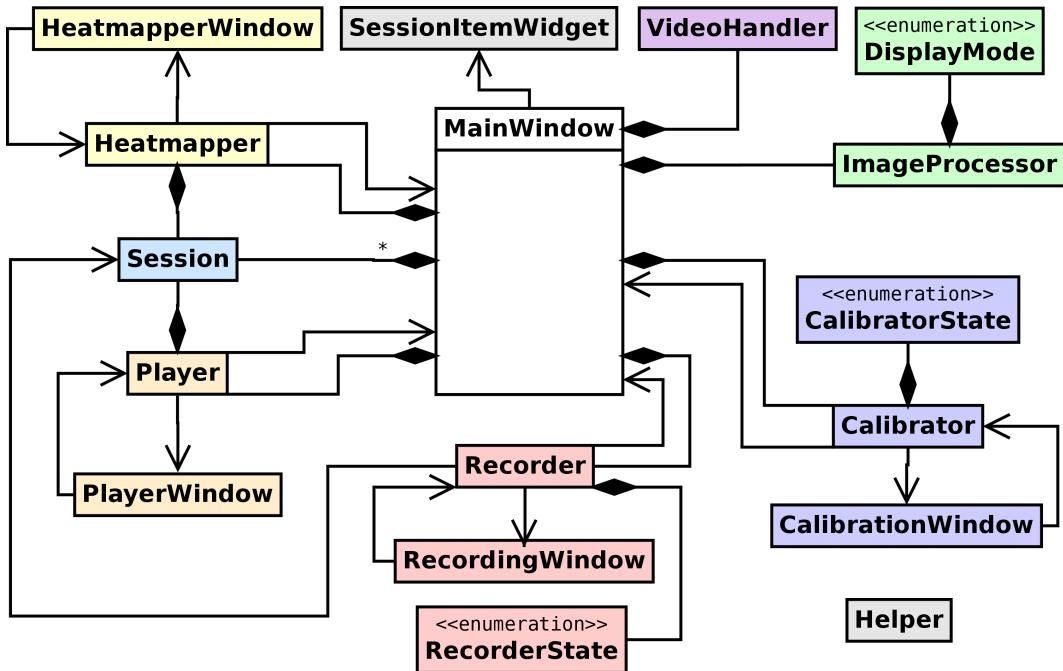
## 6.2. Architektúra

Az implementációt a 4. fejezetben bemutatott technológiák (OpenCV, Qt, Qt Creator) felhasználásával **C++ nyelven** készítettem el a Model–View–Controller (MVC) tervezési minta szem előtt tartásával. A kialakított osztályhierarchia a 6.1. ábrán látható módon épül fel. A kibővített osztálydiagram – a fontos változók és funkciók feltüntetésével – megtalálható az F.2. függelékben.

Az átláthatóság kedvéért még a kibővített diagramról is lehagytam a kevésbé fontos tagváltozókat, és a nyilvánvaló funkcióval rendelkező függvényeket (pl. „getter” vagy „setter” metódusok). Emellett természetesen az információrejtés (information hiding) alapelveinek megfelelően a tagváltozók amikor csak szükséges, privát változóként deklártak, és publikus interfész áll rendelkezésre az értékük lekérdezésére, vagy az értékkadásra.

Az alkalmazás egyes funkcióit a videofolyam feldolgozásától egészen a hőterképek generálásáig osztályok egy-egy csoportja végzi. A 6.1. ábrán színkódokkal jelöltetem a logikailag összetartozó osztályokat. A specifikáció egyes pontjait kielégítő osztályok az ábráról leolvasva, felsorolás szintjén:

- **MainWindow** – az alkalmazás főablakát tartalmazó osztály, valamint ez szolgál általános kontrollerként is, lévén a felhasználói interakciók döntő része ide fut be
- **VideoHandler** – a webkameráról beérkező videofolyam olvasásáért felelős
- **ImageProcessor** – a képkockák feldolgozását – a pupillakeresést – végzi
- **Calibrator** – a kalibrációt, majd kalibrált állapotban a *pupilla-pozíció*  $\Rightarrow$  *képernyő-pozíció* átszámítást végzi
- **Recorder** – a munkamenetek rögzítését végzi
- **Player** – a felvett munkamenetek visszajátszását teszi lehetővé ez az osztály



6.1. ábra. Az alkalmazás osztályhierarchiája

- **Heatmapper** – a hőterképek generálása történik ebben az osztályban
- **Session** – konténerosztály egy-egy munkamenet futásidőjű tárolása számára
- kisegítő osztályok

A továbbiakban sorra veszem az egyes osztályok interfészét, valamint belső működésüket, kicsit mélyebb betekintést engedve a felépítésükbe, egymással való összeköttetésükbe, valamint az esetleg érdekesnek ítélezhető implementációs részletekbe.

### 6.2.1. A MainWindow osztály

MainWindow
<pre>+PROCESS_TIMEOUT: int = 50 -processTimer: QTimer -videoHandler: VideoHandler -imageProcessor: ImageProcessor -calibrator: Calibrator -recorder: Recorder -player: Player -heatmapper: Heatmapper -sessions: std::vector&lt;Session&gt;  +startToggled(state:bool): void +calibrateToggled(state:bool): void +recordToggled(state:bool): void +equalizedClicked(): void +thresholdedClicked(): void +blobsClicked(): void +pupilClicked(): void +listItemSelected(): void +playClicked(): void +heatMapClicked(): void +deleteClicked(): void +findSessions(): void +refreshSessionList(): void</pre>

A MainWindow osztály az alkalmazás általános kontrollereként működik. A nevéből is látszik, hogy ehhez az osztályhoz tartozik a program fő interfészablaka, ennek

megfelelően a felhasználói interakciók lekezelése ezen osztály slotjaiban történik. Az osztályban tényleges feldolgozás nem történik, csak összefogja az egyes lépések elvégzni hivatott objektumokat, valamit biztosítja, hogy a felhasználói felület állapota mindenkorban mindenkorban a program tényleges belső állapotát reprezentálja, és csak engedélyezett állapotátmenetek történhessenek.

Az `int PROCESS_TIMEOUT` attribútum a `QTimer processTimer` időzítővel együtt a feldolgozás rögzített sebességgel történő elvégzését vezérli. Az előbbi ezredmásodpercben megadott értéke alapján hívódik a feldolgozást vezérlő `processTimeout` függvény.

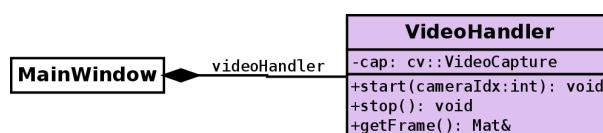
A `sessions` lista az elmentett munkamenetek futásidejű tárolására szolgál, amely listát a `refreshSessionList` függvény állítja elő.

A `videoHandler`, `imageProcessor`, `calibrator`, `recorder`, `player` és `heatmapper` attribútumok tárolják a névből adódó típusú objektumpéldányokat, amelyek az alkalmazás egyes funkciót valósítják meg.

A metódusok közül a `startToggled`, `calibrateToggled` és `recordToggled` metódusok sorra a videofolyam, kalibráció és a felvétel ki/bekapcsolását kezelik le.

A lejátszás során megjelenítési módok váltását kezelik le az `equalizedClicked` és a `pupilClicked` között felsorolt függvények.

### 6.2.2. A VideoHandler osztály

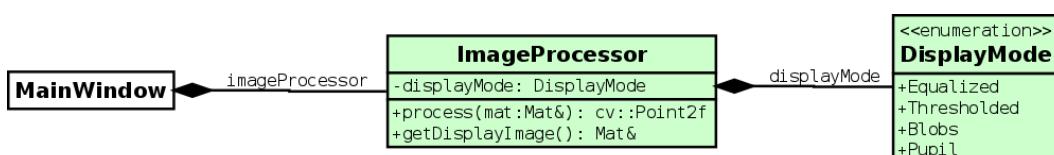


A `VideoHandler` az „alacsony szintű” videokezelést végzi. A gyakorlatban ez úgy valósul meg, hogy `cv::VideoCapture` `cap` attribútumaként tartalmazza az OpenCV kamerakép-feldolgozásra szolgáló objektumának egy példányát.

A `start` függvényét egy kameraindexszel paraméterezve inicializálhatjuk a fent említett objektumot, a `stop` függvény pedig a kamera szabályos leállítására szolgál.

Elindított állapotban a `getFrame` függvénnnyel kérhetjük le az adott pillanatban a kamera által érzékelt képet. A képet tartalmazó `cv::Mat` objektumpéldány szintén az osztály egy tagváltozójában kerül tárolásra, a `getFrame` függvény erre az objektumra mutató referenciát ad vissza. A megoldásnak nyilvánvalóan performanciabeli okai vannak: a tényleges objektumpéldány átadása felesleges számításokat róna a feldolgozási sorra.

### 6.2.3. Az ImageProcessor osztály

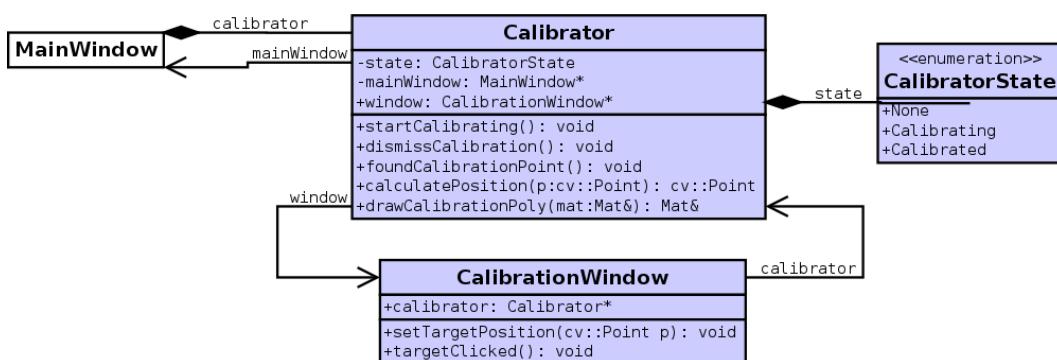


Az `ImageProcessor` osztály végzi a képkockák feldolgozását, és a pupilla azonosítását. A pupillakeresést az alkalmazás előfeldolgozás után blobok detektálásával és változatos szűrésével végzi, a 5.2.1. szakaszban részletesen bemutatott módon.

A `process` függvény végzi a felismerés, és a képen megtalált pupillaközépponttal tér vissza egy `cv::Point2f` változó használatával. A visszaadott középpont az érvénytelen ( $-1, -1$ ) koordinátákat kapja, ha nem sikerült a pupilla azonosítása.

Szükség van még ezen kívül a feldolgozás valamely közbülső állapotának képét lekérésére, ezt lehetjük meg a `getDisplayImage` metódus használatával. A függvény az `ImageProcessor` belső `displayMode` állapotának megfelelően adja vissza a kívánt képet. Fontos megjegyezni, hogy mindenekkel csak a megjelenítést befolyásolja. A `displayMode` állapottól függetlenül a pupillafelismerés folyamata minden esetben elejtől a végig lefut, majd visszatér a megtalált pupillaközéppont koordinátáival.

#### 6.2.4. A Calibrator osztály



A `Calibrator` osztály az alkalmazás kalibrálását, majd a pupilla-koordináta alapján a képernyőn éppen nézett pont koordinátájának kiszámítását végzi. A kalibráció folyamatát részletesen a 5.2.2. szakaszban ismertettem.

Az osztályhoz tartozik egy `CalibrationWindow` objektum is, amely a futás során dinamikusan kerül lefoglalásra (ezért csak a rámutató pointert tároljuk). Emellett a kalibrálás állapotát is nyilvántartjuk, erre szolgálnak a `CalibratorState` felsorolás (enumeration) értékei: nincs kalibrálva (`None`), kalibrálás alatt (`Calibrating`), vagy kalibrálva (`Calibrated`).

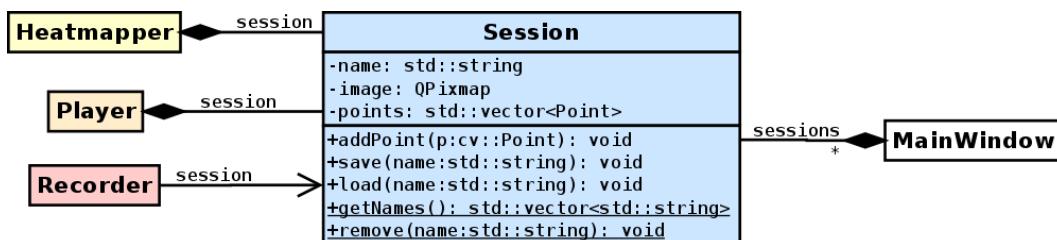
A `startCalibrating` és `dismissCalibration` függvények elnevezései magukért beszélnek. A `dismissCalibration` függvény elveti a meglévő kalibrációs paramétereit, a `startCalibrating` metódus esetében pedig új kalibrációt indítunk: megnyílik a `CalibrationWindow` osztály felhasználói felülete (teljes képernyős kalibrációs ablak), rajta az első kalibrációs ponttal.

A kalibrációs ablakban tekintetünket az aktuális pontra irányítva, majd kattintva (`targetClicked`) a `foundCalibrationPoint` függvény hívása történik meg, amelyben utasításra kerül a következő kalibrációs pont kirajzolása `setTargetPosition`, vagy az utolsó pont esetén a kalibráció lezárása.

Kalibrált állapotban a `calculatePosition` függvény a `cv::Point` formában paraméterül adott pupillakoordinátát képzi le a kalibrációs paraméterek felhasználásával képernyő-koordinátákba.

Az `ImageProcessor` osztályhoz hasonlóan itt is az interfész részét képezik olyan metódusok, amelyek nem feltétlenül a tekintetkövetés szempontjából elengedhetetlen számítási feladatokat végeznek, csak a megjelenítés miatt van szükség rájuk. A `Calibrator` osztály esetében ilyen a `drawCalibrationPoly` függvény, amely a paraméteréül kapott képre egyszerűen rárajzolja a kalibrációs pontok négyszögét.

### 6.2.5. A Session osztály



A `Session` osztály az éppen felvétel alatt lévő, és már felvett munkamenetek futás közbeni tárolására szolgál.

Tagváltozóként tartalmazza a munkameneteket jellemző tulajdonságokat: a munkamenet nevét (azonosítóját, tetszőleges `std::string name`, a képet, amelyet a felvétel alatt az alany vizsgál (`QPixmap` objektumként), valamint a vizsgálat alatt felvett képernyő-koordinátákat (`std::vector<cv::Point> points` listaként).

A publikus interfészen keresztül lehetőség van a munkamenet-objektum lementésére a `save` metódus használatával. A paraméterül adott azonosítóval kerül lementésre a munkamenet, a futtatási könyvtár `session_<név>` formátumú alkönyvtárába. A munkamenet alkönyvtárán belül létrejön a `bg.png` képfájl, ami értelemszerűen a vizsgálat alatt használt képet tartalmazza, valamint a `points.txt` szöveges fájl, amely soronként tartalmazza a rögzített pontok képernyő-koordinátáit.

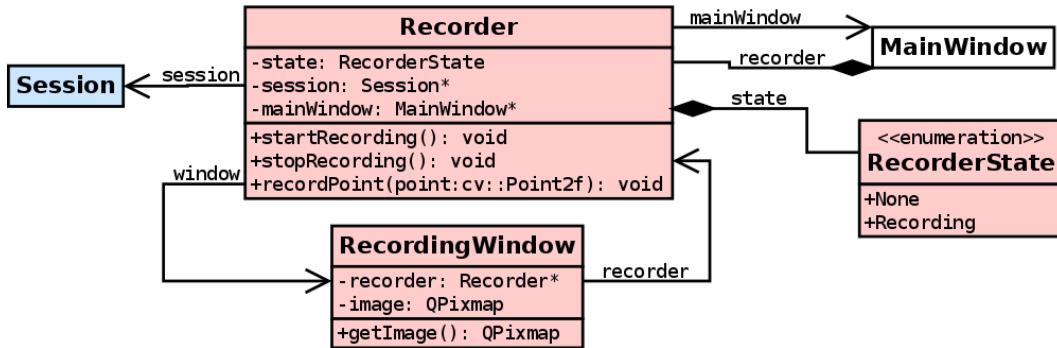
A mentés pájaként a `load` függvény tölti be a paraméterül kapott névvel lementett munkamenetet.

A fentieken kívül két statikus metódus is helyet kapott a `Session` osztályban. A `getNames` függvény a már lementett munkamenetek neveinek tömbjét adja vissza, a `remove` osztálymetódus pedig visszavonhatatlanul törli a háttértárról paraméterként megadott névvel rendelkező munkamenetet.

### 6.2.6. A Recorder osztály

A `Recorder` osztály felelős az új munkamenetek létrehozásáért. Tartozik hozzá egy, a megjelenítést végző ablak, a `RecordingWindow` osztály egy példánya. Emellett a belső állapot is nyilvántartásra kerül a `RecorderState` felsorolás értékei által.

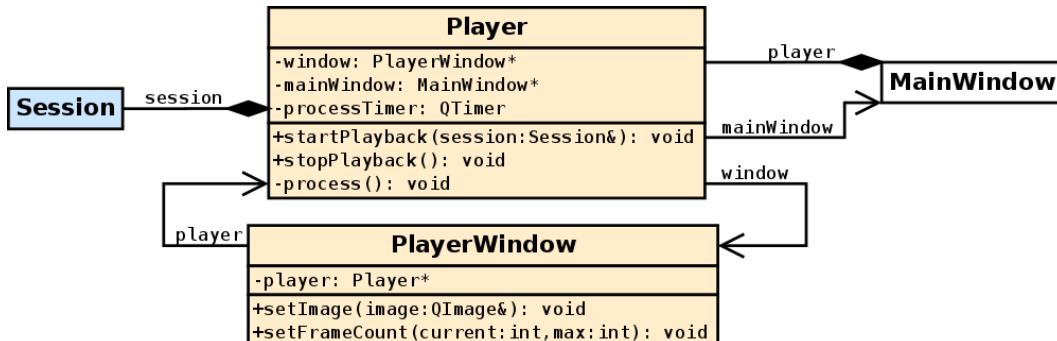
Az osztály publikus interfésze mondhatni magától értetődő. A `startRecording` függvény a meghívását követően azonnal képernyőképet (screenshot) készít az elsődleges kijelző tartalmáról, és elkezdi egy új munkamenet felvételét. A munkamenet azonosítója az alkalmazás jelenlegi állapotában automatikusan a felvétel indításának pillanatában érvényes UNIX időbélyeg (timestamp) értéke lesz.



A **recordPoint** metódussal új mérési pontot lehet a munkamenethez fűzni. Az alkalmazás használata során a **recordPoint** függvény felvétel során minden képkocka feldolgozása után meghívódik, vagyis képkockánként egy mérési pont kerül tárolásra. Ennél sűrűbb mintavételnek nyilvánvalóan nincs értelme, de a ritkább pontregisztrálás lehetősége adott.

A **stopRecording** függvény a felvétel befejeztével hívódik meg. A futása során eltünteti a felvételi ablakot, valamint le is menti az elkészült munkamenetet a háttértárra.

### 6.2.7. A Player osztály



A **Player** osztály a felvett munkamenetek lejátszásáért felelős. A lejátszást a dinamikusan foglalt **PlayerWindow** példány grafikus felületén végzi.

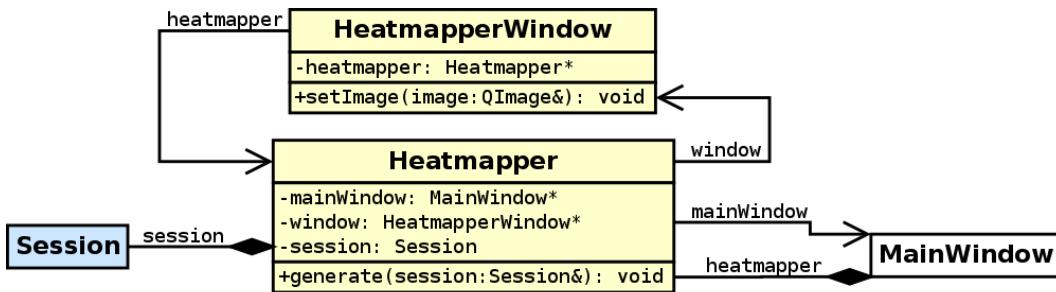
A **startPlayback** metódus egy munkamenet-objektumra (**Session**) mutató referenciát vár paraméterként, ennek lejátszását kezdi el meghívása után azonnal.

A visszajátszás futása közben a **processTimer** időzítő és a **process** függvény gondoskodik a megfelelő ütemben történő visszajátszásról, felhasználva a **MainWindow** osztály **PROCESS\_TIMEOUT** értékét. A feldolgozott képet a **PlayerWindow** osztály **setImage** metódusának meghívásával jeleníti meg.

A **stopPlayback** függvény egyszerűen bezárja a használt (teljes képernyős) ablakot, ezzel egy időben leállítja a munkamenet visszajátszását.

### 6.2.8. A Heatmapper osztály

A **Heatmapper** osztály a munkamenetekhez kapcsolódó hőtérfépek (heatmap) generálására és megjelenítésére szolgál.



Az osztály `generate` függvénye a paraméterül kapott `Sesssion` referencia alapján kezdi meg a működését. A generálás meglehetősen hosszú folyamat, ennek végeztével a `HeatmapperWindow` osztály grafikus felületén jelenik meg a munkamenet hőterképes reprezentációja.

### 6.2.9. Kisegítő osztályok

Röviden szeretnék szót ejteni az osztálydiagram eddig mellőzött elemeiről is. Ezek az osztályok nem közvetlenül a specifikáció valamely pontjára nyújtanak megoldást, inkább ismétlődő feladatok megoldására adnak helyet.

**A SessionWidgetItem osztály** Az osztály a megjelenítésben jut szerephez. A munkamenetek egy listában kerülnek megjelenítésre a főablak bal oldalán. A lista elemeinek egyedi megjelenítésük van, ennek a grafikus elemnek (widget) a felületét tartalmazza a `SessionWidgetItem` osztály.

**A Helper osztály** A kódolás során több helyen is szükség volt különböző átalakítások elvégzésére. A `Helper` osztály statikus metódusai időbényegek formázására, valamint a Qt és az OpenCV belső kép-reprezentációi közötti oda-vissza átalakításra szolgálnak.

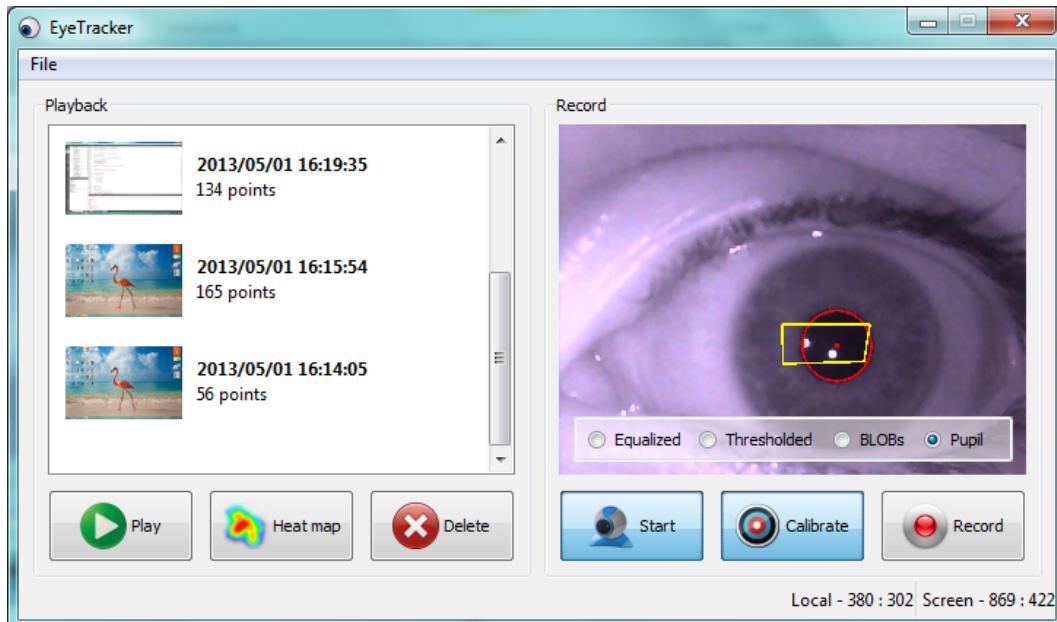
## 6.3. Grafikus felhasználói felület

Az alkalmazás felhasználói felülete az előző szakaszban említetteknek megfelelően több ablakot is felhasznál (pl. kalibrációkor, felvételkor, visszajátszáskor), de ezek többnyire csak teljes képernyős képek megjelenítésére használatosak, felhasználói interakció nem történik rajtuk.

A felhasználó – legyen az a vizsgáló vagy az alany – leginkább az alkalmazás 6.2. ábrán látható **főablakával** találkozik.

Az főablak két részre oszlik. Az ablak bal oldalán a felvett **munkamenetek menedzszerére** nyílik lehetőség, míg a jobb oldalon a kalibrációt és az új munkamenetek felvételét végezhetjük.

A bal oldal nagy részét elfoglaló listanézet a rögzített munkameneteket tartalmazza időrendben visszafelé. Ahogy az ábrán látható, az egyes munkamenetekhez tartozó sorokban az összes munkamenet szempontjából fontos információ – a felvétel időpontja, a felvett pontok száma, valamint a felvétel során vizsgált kép előnézete – kijelzésre kerül. A lista alatt található három gombbal nyílik lehetőség sorrendben a



6.2. ábra. Az alkalmazás főablaka

visszajátszásra, a hőterkép generálására, illetve a munkamenet törlésére (a törléshez megerősítés szükséges).

A jobb oldal jelentős részét az **élő kamerakép** foglalja el. A kép alatti három gomb a kamera inicializálására, a kalibráció indítására, valamint új munkamenet rögzítésére szolgál. A kamerát inicializálva az élőképen elhelyezett rádiógombokkal tudjuk vezérelni, hogy a pupillakeresés melyik feldolgozási fázisa jelenjen meg az előnézeti képen.

A kalibrációt elindítva teljes képernyős, homogén fehér hátterű ablak nyílik meg. A kalibrációs pontok rögzítését a sarkokban megjelenő gombokra kattintva érhetjük el. A kalibráció végeztével a rögzített sarokpontok kijelzésre kerülnek az élőképen (sárga négyzetök). Új munkamenet rögzítése esetén szintén új, teljes képernyős ablak nyílik, a képernyő aktuális tartalmát mintegy „befagyasztva”.

A jobb oldalon található gombok mindegyike **két állapotú**. Bekapcsolt állapotukban újra megnyomva őket tudjuk a felvételt leállítani, az eddigi kalibrációt semmisé tenni, valamint a kamerát kikapcsolni. A jobb oldali gombok értelemszerűen csak meghatározott sorrendben használhatók: a funkciók balról jobbra haladva aktiválhatók.

## 6.4. Felhasználói dokumentáció

Az alkalmazás indítása után a felhasználót a főablak (lásd 6.2. ábra) fogadja. Az ablak bal oldalán a felvett munkamenetek megtekintésére, törlésére, lejátszására, illetve hőterkép generálására van lehetőség. A főablak jobb oldalán történhet az új munkamenet rögzítése.

### Munkamenet felvétele

1. Adjuk fel a vizsgálandó alanyra a sapkát, amelyhez a webkamera rögzítve van!
2. Végezzük el a kamerakép bekapcsolását a **Start** gomb megnyomásával (gyorsbillentyű: **S**)!
3. Győződjünk meg róla, hogy a kamera orientációja megfelelő: ha nem, irányítunk a szemrégióra!
4. Szükség esetén állítsuk be a kamera fókuszát a lencse tekerésével úgy, hogy a pupilla a lehető legélesebben rajzolódjon ki!
5. Kérjük meg az alanyt, hogy a vizsgálat során csak a szemét mozgassa, a fejét ne! A pontosabb mérés érdekében használhatunk álltámaszt is.
6. Indítsuk a kalibrációt a **Calibrate** gomb megnyomásával (gyorsbillentyű: **C**)!
7. A kalibráció során kérjük meg az alanyt, hogy tekintetét irányítsa a képernyő sarkain sorban megjelenő céltáblákra! A kalibrációs pont rögzítését a céltáblára kattintva, vagy az **Enter** gyorsbillentyű segítségével végezhetjük.
8. A háttérben jelenítsük meg a vizsgálni kívánt képet (pl. weboldal részlete).
9. A **Record** gomb megnyomásával indítsuk el a munkamenet felvételét (gyorsbillentyű: **R**)
10. A felvételt a **Record** gomb újból megnyomásával, vagy az **Escape** gyorsbillentyűvel állíthatjuk le.

### Munkamenet visszajátszása

1. Válasszuk ki a bal oldali listából a visszajátszani kívánt munkamenetet!
2. Kattintsunk a **Play** gombra! A visszajátszás megkezdődik, és folyamatosan ismétlődik, amíg le nem állítjuk.
3. A visszajátszás leállításához nyomjuk meg az **Escape** billentyűt!

### Hőterkép generálása

1. Válasszuk ki a bal oldali listából azt a munkamenetet, amelyhez hőterképet szeretnénk generálni!
2. Kattintsunk a **Heatmap** gombra! A generálás megkezdődik, a folyamat állása a státuszszávon visszajelzésre kerül. A generálás végeztével a hőterkép megjelenik a képernyőn.
3. Használjuk az **Escape** billentyűt a hőterkép bezárásához.

### Munkamenet törlése

1. Válasszuk ki a bal oldali listából a törölni kívánt munkamenetet!
2. Kattintsunk a **Delete** gombra! A törlés megerősítését kérő ablak ugrik fel.
3. Erősítsük meg törlési szándékunkat az **OK** gombra kattintva!

**Többmonitoros használat** A program fel lett készítve a több monitorral történő használatra. Amennyiben nem a saját tekintetünket követjük, hanem egy másik felhasználóét (alany), úgy a kétmonitoros elrendezés használata a legkényelmesebb.

Az alkalmazás főablakát húzzuk a másodlagos monitorra, ez lesz a vizsgáló képernyője. Az elsődleges monitort használjuk az alany képernyőjeként, azon jelenítsük meg a vizsgálni kívánt ábrákat, képeket, weboldal-részleteket stb. A másodlagos monitoron a vizsgáló a mérés teljes időtartama alatt szemmel tarthatja az alkalmazás főablakát. Ennek köszönhetően például azt is figyelemmel kísérheti, hogy a kamera vagy a fej elmozdulásából kifolyólag a kalibráció a mérés során (vagy két mérés között) helytelenné válik-e. Amennyiben igen, kérheti új kalibráció elvégzését és a mérés megismétlését.

## 6.5. Tesztelés, eredmények

+++ szubjektív tesztek + kepek! +++

---

# Értékelés

+++ általános értékelés ide +++

---

# Köszönetnyilvánítás

+++ koszonetnyilvanitas ide +++

## Irodalomjegyzék

- [1] Ragó Anett Csépe Valéria, Győri Miklós. *Általános pszichológia I. – Észlelés és figyelem.* Osiris, Budapest, 2007.
- [2] E. O. Andreeva, P. Aarabi, M. G. Philiastides, K. Mohajer, and M. Emami. Driver drowsiness detection using multimodal sensor fusion. *Proc. SPIE, Vol. 5434*, 2004.
- [3] Arne John Glenstrup and Theo Engell-Nielsen. Eye Controlled Media: Present and Future State. Master's thesis, University of Copenhagen, 1995.
- [4] D. Scott and J. M. Findley. Visual search, eye movements and display units. Technical report, University of Durham, South Road, Durham DH1 3LE, UK, 1993.
- [5] P. E. Hallett. *Handbook of Perception and Human Performance I*, chapter Eye movements, pp. 10.25-10.28. 1986.
- [6] S. Baluja and D. Pomerlau. Non-intrusive gaze tracking using artificial neural network. Technical report, School of Computer Science, Carnegie Mellon University, Pittsburgh PA, USA, 1993.
- [7] G. d'Ydewalle P. U. Muller, D. Cavegn and R. Groner. *Perception and Cognition*, chapter A comparison of a new limbus tracker, corneal reflection technique, purkinje eye tracking and electro-oculography, pp. 393-401. Elsevier Science Publishers, B.V, 1993.
- [8] Shebilske W. L. and D.F. Fisher. Understanding Extended Discourse Through the Eyes: How and Why. *Eye Movements and Psychological Functions: International Views (pp. 303?314).*, 1983.
- [9] R. H. S. Carpenter. *Movements of the Eyes*. London: Pion, 1977.
- [10] J. D. Cook D. A. Robinson D. S. Zee, L. M. Optican and W. K. Engel. Slow Saccades in Spinocerebellar Degeneration. *Archives of Neurology, 33, 243?251.*, 1976.
- [11] Kovács Balázs. Pupillakövetés képfeldolgozási módszerei, 2010.
- [12] P. V. C. Hough. Machine Analysis of Bubble Chamber Pictures. *Proc. Int. Conf. High Energy Accelerators and Instrumentation*, 1959.

- [13] R. O. Duda and P. E. Hart. Use of the Hough Transformation to Detect Lines and curves in pictures. *Comm. ACM*, Vol. 15, pp. 11-15, January, 1972.
- [14] D. H. Ballard. Generalizing the Hough Transform to Detect Arbitrary Shapes. *Pattern Recognition*, Vol. 13, No. 2, 1981.
- [15] H. K. Yuen, J. Princen, J. Illingworth, and J. Kittler. Comparative study of Hough transform methods for circle finding. *Image and Vision Computing*, Vol. 8, Issue 1, February, 1990.
- [16] Davies E. R. A modified Hough scheme for general circle location. *Pattern Recognition Letters*, Vol. 7, No. 1, 1988.
- [17] Illingworth J. and Kittler J. The adaptive Hough Transform. *IEEE Trans. Pattern Analysis & Machine Intelligence*, Vol 9, No. 7, 1987.
- [18] L. A. F. Fernandes and Oliveira M. M. Real-time line detection through an improved Hough transform voting scheme. *Pattern Recognition*, Elsevier, Vol. 41, Issue 1, pp. 299-314, January, 2008.
- [19] P. Viola and M. Jones. Robust Real-time Object Detection. *Second International Workshop on Statistical and Computational Theories of Vision-Modeling, Learning, Computing, and Sampling*, 2001.
- [20] Y. Freund and R. E. Schapire. A Decision-Theoretic Generalization of On-line Learning and an Application to Boosting, 1995.
- [21] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. *Proceedings of Imaging Understanding Workshop*, pages 121–130, 1981.
- [22] J. L. Pfalz A. Rosenfeld. Sequential Operations in Digital Picture Processing. *Journal of the ACM* Vol. 13, No. 4, 1966.
- [23] K. Suzuki Lifeng He, Yuyan Chao. A Run-Based Two-Scan Labeling Algorithm. *IEEE Transactions on Image Processing* 17, 2008.
- [24] K. Suzuki, I. Horiba, and N. Sugie. Linear-time connected-component labeling based on sequential local operations. *Computer Vision and Image Understanding* 89, 2003.
- [25] Y. Han and R. A. Wagner. An efficient and fast parallel-connected component algorithm. *Journal of the ACM*, 1990.
- [26] OpenCV – Wikipedia szócikk. <http://en.wikipedia.org/wiki/OpenCV>.
- [27] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, 2008.
- [28] Qt (framework) – Wikipedia szócikk. [http://en.wikipedia.org/wiki/Qt\\_\(framework\)](http://en.wikipedia.org/wiki/Qt_(framework)).
- [29] Qt Creator – Wikipedia szócikk. [http://en.wikipedia.org/wiki/Qt\\_Creator](http://en.wikipedia.org/wiki/Qt_Creator).

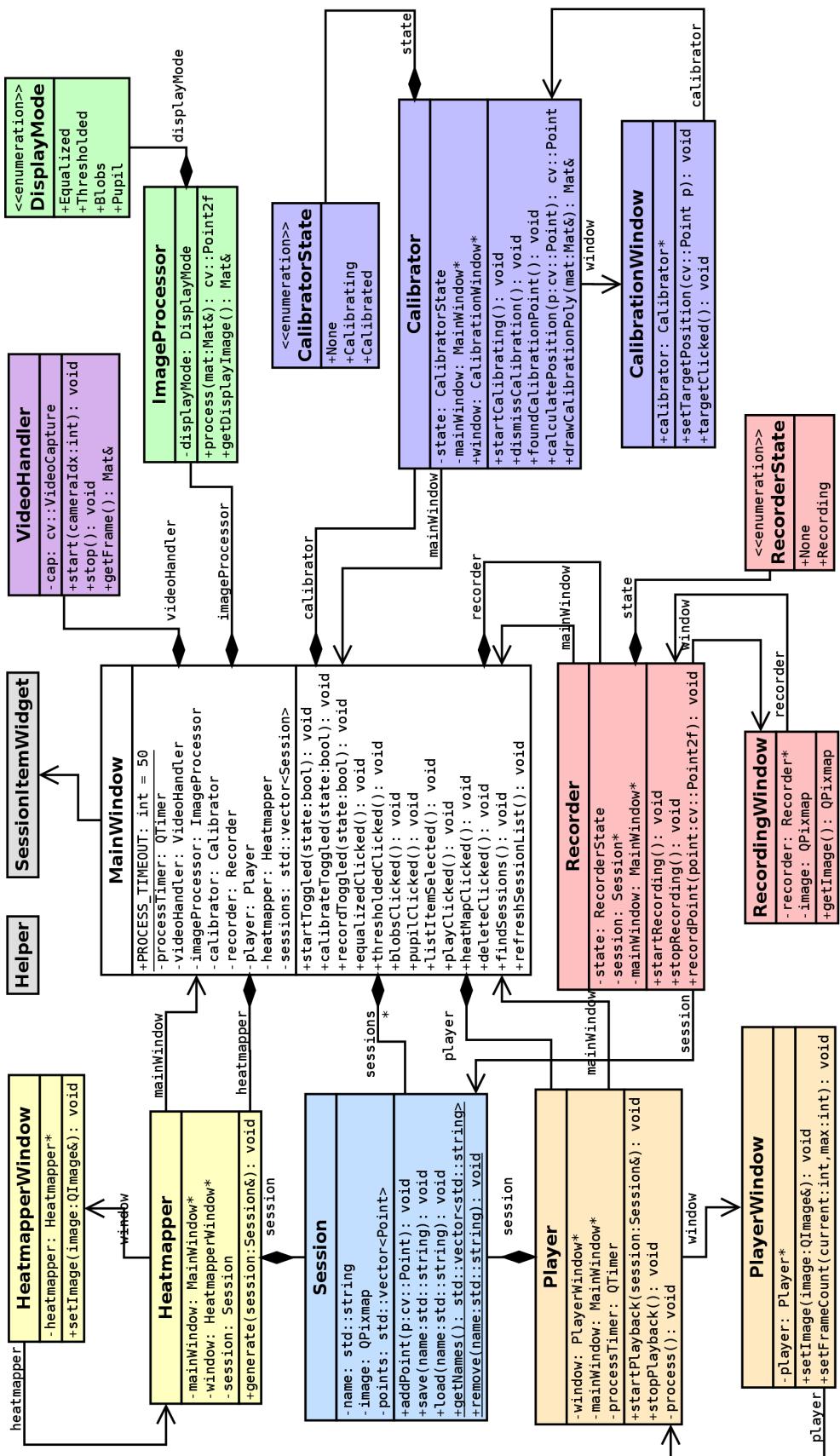
---

# Függelék

## F.1. Paraméterezés

blabla fv hogy van parameterzve

## F.2. Kibővített osztálydiagram



---

### F.3. Telepítés

+++ forditas menete, szukseges szoftverek +++