Diplomamunka

Képek szegmentálása szín és mozgás alapján

Horváth Péter

V. éves programtervező matematikus hallgató

Témavezető: Dr. Kató Zoltán

Szegedi Tudományegyetem

Természettudományi Kar

Képfeldolgozás és Számítógépes Grafika Tanszék

2004.

Tartalomjegyzék

1. fejezet

Előszó

Szakdolgozatomban egy hosszabb project idáig született eredményeit, további lehetőségeit és problémáit mutatom be.

A project a mozgás meghatározásának feladatával indult. A témában nagyon nagy mennyiségű szakirodalom lelhető fel, igen jó hatékonyságú robosztus módszerek születtek, bár a legtöbbnek megvan egy speciális felhasználási területe, amelyben jól működik, illetve többségük robosztusságából fakadóan nem elég gyors. Néhány tanulmány segítségével kifejlesztettem egy algoritmust, amely megfelelően gyors, bár zajra érzékeny. Nagyon jól használható, amennyiben zaj és rezgésmentes input adatok állnak rendelekezésünkre, ám még ebben az esetben is merülhetnek fel hibák. Ezek kiküszöbölésére elkészítettem egy másik eljárást, mely energiminimalizációs módszerrel rendkívül pontos és precíz eredményt produkál.

A mozgás detektálásával egy időben felmerült az igény az algoritmus során kinyert featureok szegmentálására is. Hosszas vizsgálódás után az aktív kontúrok használata mellett döntüttünk, ehhez szintén megfelelő mennyiségű szakirodalom állt rendelkezésünkre. Találtunk egy a tradicionális módszerek néhány hibáját kiküszöbölő *Gradient Vector Flow* nevű eljárást, mely úgy tűnt megfelelhet a problémáinkra. Ám hamar kiderült a hátránya, nem lehet olyan esetekben használni, ahol több feature áll rendelkezésre. Mivel a mozgás több komponensű, ezen gradiens alapú módszernek elkészítettük egy kiterjesztését több feature esetére, több dimenziós függvények parciális deriválásának segítségével. Így a rendszert alkalmassá tettük mozgás szegmentálására.

A képfeldolgozásban számos hasonló problémát találunk, vagyis a szegmentálást nem egy, hanem több rendelkezésünkre álló feature segítségével kellene végeznünk, tehát ezen esetekben is használható a módszerünk. A legnyilvánvalóbb a különböző színterek esete, ha több színcsatorna is van. Kipróbáltuk az algoritmust RGB, HSL, CMYK és LUV színterekben, meglehetősen jó tapasztalataink vannak. Főleg az LUV színtérben, melyben az érzékelt színek közötti távolság megegyezik a színtér-

beli Euklideszi-távolsággal.

Témavezetőm egyik szakterülete a textúrakinyerő algoritmusok, javaslatára (és mivel a textúrakinyerő algoritmusok is több dimenziós featuret szolgáltatnak) teszteltük az algoritmust ezen esetekre is. Ezek a textúrakinyerő algoritmusok Gabor-filer és MRSAR módszerek voltak, tapasztalataink szerint a módszer és a paraméterek helyes választásával az algoritmus jól működik.

A projectet a jövőben két irányban fogjuk továbbfejleszteni. Az egyik egy valós rendszer megvalósítása, ami a featureok real-time kinyerésére és egy robotizált kameravezérlésre fog irányulni. A másik a jellemzőkivonás után gépi tanulási módszerek segítségével mozgó alakok felismerése lesz.

A dolgozat felépítése

2. fejezet

Aktív Kontúrok

Ebben a fejezetben bemutatom az *aktív kontúr*ok fajtáit, és az azokat mozgató külső erőket, melyek egy vektormezőt határoznak meg, ismertetem ezen vektromezők konstrukcióját, a tradícionális eljárások hibáit, bővebben tárgyalok egy speciális és tapasztalataink szerint igen jól és univerzálisan használható vektormezőt ez a *Gradient Vector Flow*, melynek elkészítettük egy kiterjesztését így már a széleskörűen felhasználható szegmentálási problémák megoldásaira. Most tekintsük át az eddig megjelent publikációkat és egy általános bemutatását az aktív kontúroknak.

2.1. Bevezetés

Az *aktív kontúr* vagy másik talán találóbb megnevezése *snake* azaz kígyó egy kép felületén definiált görbe [17], melyenek alakját és mozgását a görbe formájából számított *belső energia* és a kép színintenzitásaiból számított *külső energia* befolyásol. Ezen belső és külső energiákat úgy definiáljuk, hogy a kontúr segítségükkel rásímuljon a kép éleire vagy más általunk választott jellemzőkre. Az aktív kontúrok széleskörűen használt eszközei a képszegmentálásnak, modellezésnek, éldetektálásnak, alakzatmodellezésnek és mozgáskövetésnek. Bár szakdolgozatomban főleg 2 dimenziós alkalmazások bemutatására koncentrálok de természtesen 3 dimenziós felhasználásra is lehetőségünk van, ekkor *aktív surface*nek vagyis aktív felületnek nevezzük, mely kiváló eszköz szegmentálásra és felületmodellezére.

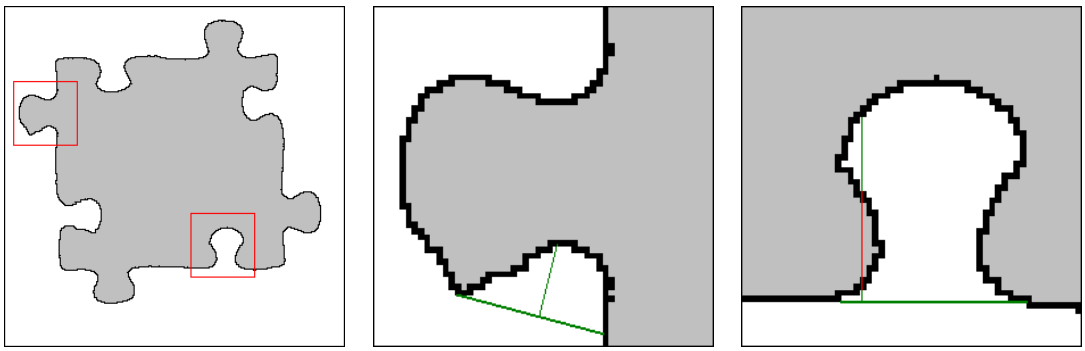
Az eddig megjelent szakirodalom az aktív kontúrok két típusát különbözteti meg: *paraméteres* és *geometriai aktív kontúr* (parametric és geometric active contour). Szakdolgozatomban a paraméteres aktív kontúrokkal foglalkozom, bár a módszer megalkotói szerint a Gradient Vector Flow (továbbiakban GVF) geometriai aktív kotnúrok esetén is használható [30] . A paraméteres aktív kontúrokat paraméteres görbék határozzák meg, melyeket a külső és belső energiák deformálják. Tipikusan a görbe a *potenciális erő*k segítségével az élek irányába mozdul, melyet a potenci-

afüggvény negatív gradienseként definiálunk, a külső erőt alkotó másik komponens a *nyomóerő*, a belső erőket szintén két komponensre bonthatjuk mégpedig a görbe összetartásáért felelős *elasztikus erő*re (elasticity force) és a túlságos elhajlást megakadályozó *elhajlási erő*re (bending force).

A paraméteres aktív kontúrokat használó algoritmusoknak két sarkallatos pontja van, az első probléma a görbe inicializálásakor merül fel, nagyon körültekintőnek kell lennünk különben rossz helyekre konvergálhat a görbe, több javaslat is született a megfelelő inicializálásra, ezek között multirezolúciós eljárások [19], nyomási energiát vagy távolságpotenciált felhasználó módszerek vannak [5]. Az eljárások alapötlete az, hogy egy olyan inicializálást adjunk, mely használata esetén csökkenteni tudjuk a külső energia mező számítási tartományát és a kontúrt a keresett határok felé tudjuk irányítani. Második nagy probléma az aktív kontúrokkal az, hogy a határokon lévő konkáv részekbe nem tud bekonvergálni. Nem publikáltak kielégítő eredményt ezen a probléma megoldására, de a nyomási energiára a kontrol pontokra a terület adaptálásra és vonzóerő irányítására vannak javaslatok. Sajnos a legtöbb módszer csak ezen problémák közül egyre ad megoldást és közben újabb nehézségekbe ütközünk. Bevezetünk egy új fogalmat, melyet a szakirodalomban *capture range* néven emlegetnek és a továbbiakban *elérési terület* néven fogok rá hivatkozni, ez a kép azon területeire mutat melyekről inicializálva a kontúrt sikerül annak a keresett jellemzőre húzódnia. Példaként megemlíteném a multi-layer algoritmusokat, melyek megnövelik az elérési terület nagyságát ám nagyon erős megszorítási vannak a kontúr mozgására az egyes szinteken. Egy másik példa azon eljárások, melyek a nyomási erőn alapulnak, jól használhatók a határvonal konkavitásainak (*boundary concavities*) megkeresésére, ám az erős élek detektálására nem megfelelő.

A határvonal konkavitásaira mutatok be példát a 2.1. ábrán. Az (a) ábra az eredeti 4 darab egymásba illesztett puzzle részecske, a szürke területet szeretnénk szegmentálni az aktív kontúr segítségével, a határvonal egy szakaszát *konkáv*nak nevezzük, ha bármely két pontját kiválasztva az azokat összekötő egyenes szakaszhoz húzható derékszögű egyenes a két pont között lévő görbe szakasz összes pontjából (b). Az 2.1. árba (c) részén *konvex* felület látható, vannak olyan pontok amleyek nem érhetők el a belső rész érintése nélkül.

Az általam használt Gradient Vector Flow egy új módszert biztosít az aktív kontúrok külső energiájának számítására mely a korábban vázolt két problémát az elérési terület méretét és a határvonal konkavitást megoldja. Ez a vektormező a kép színintenzításaiból nyert sűrű mező, melyet bizonyos energiafüggvények minimalizálásával kapunk, mely minimalizálást néhány lineáris egyenletrendszer megoldásával végezhetjük el. Ezen egyenletrendszerek inicializálását a szürkeárnyalatos



(a) (b) (c)

2.1. ábra. Példa határvonal konkavitására (b) és konvexitására (c).

vagy bináris kép gradienseinek számításaival végezhetjük. Azt az aktív kontúrt amely a GVF-en mozog *GVF-snake*-nek nevezzük (GVF-snake), mely abban különbözik majdnem az összes külső energiát használó aktív kontúrtól, hogy a külső energiát nem lehet leírni a potenciálfüggvény külső gradienseként, mivel nem formalizálható sima energiaminimalizálásként, numerikus implemetációja pedig egy iteratív eljárás lesz. A GVF különleges képessége, hogy érzéketlen az inicializálásra, a kezdeti kontúr lehet az objetumon belül, kívül vagy metszheti is a körvonalát. A nyomási energiát felhasználó sznékekkel ellentétben esetünkben nincsen szükség priori tudásra ahhoz hogy ráhúzódjon vagy kifeszüljön az objektum határaira.

A GVF-nek igen nagy az elérési területe, mely azt jelenti, hogy más objektumok elérési területén kívül bárhol lehet inicializálni tetszőlegesen távol a határtól. Az elérési terület megnövelését egy beolvasztási eljárás segítségével érik el, mely nem mossa össze az éleket így nincsen szükség multirezolúciós eljárásokra. A GVF külső energiájának a lelke a Cohen&Cohen féle *distance potencial force* [5]. A GVF energiáit az éltérképből származtatjuk, mely nagy elérési területet szolgáltat, bár a distance potencial force önmagában nem tudja a sznéket a határvonal konkavitásaiba mozgatni. Az aktív kontúrok közül választásom ezen pozitív tulajdonságai miatt esett a GVF-re.

Gyorsan kiderült azonban, hogy a szegmentálási problémák között sokszor nem csak egyetelen szürkeárnyalatos kép hanem több is, azaz több feature mátrix áll rendelkezésünkre, sajnos nem található a szakirodalomban olyan modell mely ezen problémára megoldást ad, ezért kifejlesztettük a GVF egy kiterjesztését több dimenzióra, így már használható színes képek, mozgás és textúrázott képek szegmentálására [16] .

2.2. Paraméteres sznék modell

A tardícionális sznék egy olyan paraméteres görbe $\mathbf{x}(s) = [x(s), y(s)], s \in [0, 1]$, mely a kép területén mozog és a következő energiafüggvényt minimalizálja:

$$E = \int_0^1 \frac{1}{2}[\alpha|\mathbf{x}'(s)|^2 + \beta|\mathbf{x}''(s)|^2] + E_{ext}(\mathbf{x}(s))ds, \tag{2.1}$$

ahol α és β súlyparaméterek melyek felügyelik a görbe nyúlékonyságát és merevségét, $\mathbf{x}'(s)$ és $\mathbf{x}''(s)$ jelöli az s szerinti első és második deriváltat. A külső energia az $\mathbf{E}_{ext}$ a képből származik úgy, hogy alacsony értékeket vesz fel az értékes jellemzők helyén és magasat a homogén területeken. Adott egy szürkeárnyalatos $I(x, y)$ kép[1], a külső energiákat általában úgy tervezik, hogy a kotnúrt lépésről lépésre a kontúr felé mozdítsák:

$$E_{ext}^{(1)}(x, y) = -|\nabla I(x, y)|^2 \tag{2.2}$$

$$E_{ext}^{(2)}(x, y) = -|\nabla(G_\sigma(x, y) \star I(x, y))|^2, \tag{2.3}$$

ahol G_σ a két dimenziós Gauss sűrűségfüggvény σ standard eloszlással, ∇ a gradiens operátor mely számítására több lehetőségünk van például a Sobel-operátor, melyet a (2.4)-en láthatunk..

$$S_1 = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}, S_2 = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, \tag{2.4}$$

Abban az esetben, ha a kép bináris[2] a helyes külső energia a következő:

$$E_{ext}^{(3)}(x, y) = I(x, y) \tag{2.5}$$

$$E_{ext}^{(4)}(x, y) = G_\sigma(x, y) \star I(x, y). \tag{2.6}$$

A definícióból látható, hogy σ nagy értékei esetén az élek elmosódnak nehezen megtalálhatóvá válnak, mégis bizonyos esetekben érdemes nagyra választani, mert így az aktív kontúr elérési területe megnő.

Azon sznékeknek, melyek minimalizálják (2.1)-et ki kell elégíteniük a következő Euler-egyenletet:

$$\alpha\mathbf{x}''(s) - \beta\mathbf{x}''''(s) - \nabla E_{ext} = 0. \tag{2.7}$$

Ezt úgy is felfoghatjuk, mint a következő erő egyensúly egyenltet:

[1] Tekintsük a kép (x, y) változóit folytonosnak.

[2] Csak fekete és fehér színeket tartalmaz.

$$F_{int} + F_{ext}^{(p)} = 0, \tag{2.8}$$

ahol $F_{int} = \alpha \mathbf{x}''(s) - \beta \mathbf{x}''''(s)$ és $F_{ext}^{(p)} = -\nabla E_{ext}$. A belső energia F_{int} megakadályozza az aktív kontúr túlságos elhajlását és nyúlását, míg a külső helyzeti energia $F_{ext}^{(p)}$ rávezeti a snakeet a keresett élekre. A snake a (2.7) megoldásához az $\mathbf{x}$-et dinamikusan s és t[3] függvényeként kezeli. Ekkor az $\mathbf{x}$ parciális deriváltja t-függvényében a (2.7) bal oldalából kifejezve a következő:

$$\mathbf{x}_t(s,t) = \alpha \mathbf{x}''(s,t) - \beta \mathbf{x}''''(s,t) - \nabla E_{ext}. \tag{2.9}$$

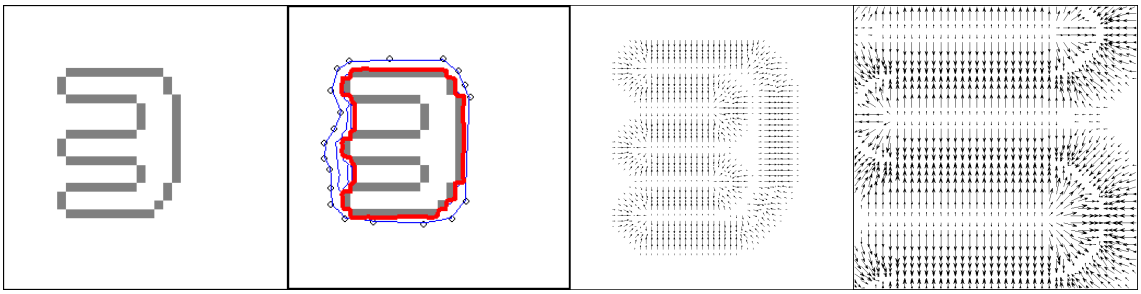
Így ha a (2.9)-ban $\mathbf{x}_t(s,t)$ stabillá válik (bekonvergál), t paraméter eltűnik és megkapjuk a (2.7) egyenlet eredményét. A (2.9) megoldását úgy kapjuk, hogy diszkrétté tesszük az egyenletet és iteratívan megoldjuk. Megjegyzem a legtöbb snake implemetáció használ valamilyen súlyparamétert a az $\mathbf{x}_t$ vagy a ∇E_{ext} módosítására, így vezérli a kontúrt a megfelelő területre és elkülöníti a külső energiát a vezérléstől. A Gradient Vector Flow erre egy normalizálást használ, mely segítségével a külső energia vektorait egysényire állítja.

2.3. A tradicionális snake-ek viselkedése

Az 2.2. ábrán láthatunk példát a tradicionális snake viselkedésére bináris képen, melynek két "U" alakú bemélyedése határvonal konkavitást tartalmaz (a). Az ábra (b) részén látható, hogyan simul rá a kezdeti snake az objektumra, az inicializálás az objektumon kívül, de az elérési területen belül történt, így a potenciális erők segítségével a kontúr rá tudott símulni az élkre, kék színnel az inicializáló görbe a kontrollpontokkal és a konvergálás lépései láthatók, pirossal a végleges szegmentálás. A (c) ábrán ábrázoltam a potenciális erők mezőjét, $\mathbf{F}_{ext}^{(p)} = -\nabla E_{ext}^{(4)}$ és a $\sigma = 25.0$ értékkel. Látható, hogy a tradicionális snake nem konvergált be a területi konkavitások azon részeibe, melyek párhuzamos határoló éleket tartalmaznak, ezen tulajdonságra a magyarázat a (d) ábrán látható, párhuzamos egyenesek esetén a gradiens vektorok egymással éppen ellentétes irányúak, így az oldalról érkező kontúrt azonnal kiirányítják a két párhuzamos élhez azaz meggátolják abban, hogy beljebb haladhasson, sajnos a tradicionális snake esetén nem tudjuk helyesen beállítani az α és β paramétereket, hogy a kontúr a határvonal konkavitásaiba is bekonvergáljon.

A tradicionális snake formulák másik kulcs problémája a limitált elérési terület, melyet a (2.2 c) ábra tanulmányozásával érthetünk meg. Láthatjuk, hogy a külső erők nagzsága az alakzat határától távolodva meglehetősen gyorsan csökken. A σ

[3] t-vel az időt mint paramétert jelöljük.

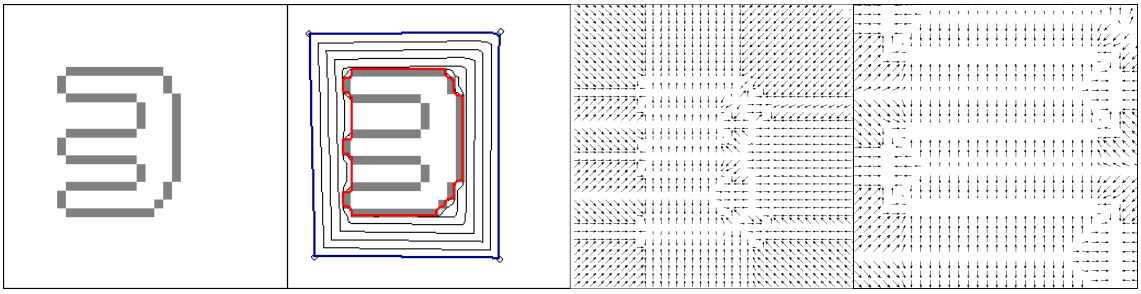


(a) (b) (c) (d)

2.2. ábra. Példa tradicionális snake konvergenciára (b), a potenciális erőkre (c) és a határvonal konkavitásnál kinagyítva a képet (d).

növelésével kiterjeszthető ez a terület, de a határok lokalizálása kevésbé lesz pontos, végül magukat a konkavitásokat is eltünteti a túl nagy σ érték.

Cohen&Cohen [5] mutatták be azt a külső energiamodellt, mely nagyon nagy mértékben megnövelte a tradicionális snake elérési trületét. Ezen külső energiát egy olyan helyzeti energiamező gradienseként számoljuk, melyet Euklideszi vagy Chamfer [3] távolságdefiníció segítségével készítünk. A (2.3) ábrán láthtjuk a módszer végeredményeként kapott szegmentálást. A (b) ábrán a megnövelt elérési területet, kékkel az inicializálást és a kontroll pontokat pirossal pedig a kontúr nyugalmi állpotát láthatjuk. A (c) ábrán láthatjuk a helyzeti energiamezőt, ebben az esetben már az objektumtól távoli vektorok is nagyok, rámutatva miért nagy az elérési területe a modellnek.



(a) (b) (c) (d)

2.3. ábra. Távolságtranszformáción alapuló energiamező, a kontúr ebben az esetben sem konvergált be, de az elérési terület jelentősen megnőtt.

Mint a (b) ábra mutatja a snake szintén nem konvergál be a határvonal konkavitásokba, erre a (d) ábrán látható kinagyított részen található a magyarázat. Látható, hogy a tradicionális helyzeti energiához hasonlóan itt is egymással ellentétes irányba mutatnak a vektorok a határvonal konkavitásaiban, melyek így nem tudják a kontúrt belevezetni a kérdéses helyekre. Bár Cohen&Cohen megadtak egy nemlineáris transzformációt a módszer kiterjesztésére, de ez csak a vektorok nagyságát

változtatja az irányát nem. Vagyis a határvonal konkavitásainak problémáját nem tudjuk megoldani a tradicionális helyzeti energiamezőket felhasználva.

2.4. Gradient Vector Flow snake

Új fajta megközelítése az aktív kontúroknak, 1998-ban mutatta be C. Xu és J. L. Prince [30]. A problémát alapvetően úgy közelítik meg, hogy a snake tervezésekor kiindulópont az erőegyensúlyi feltétel legyen (2.8). Definiálnak egy később bemutatásra kerülő új statikus külső energiamezőt $\mathbf{F}_{ext}^{(g)} = \mathbf{v}(x, y)$, melyet *gradient vector flow*-nak nevezünk. Azért, hogy a dinamikus snake egyenlet továbbra is teljesüljön (2.9) -ban $-\nabla E_{ext}$-et $\mathbf{v}(x, y)$-nal helyettesítjük:

$$\mathbf{x}_t(s,t) = \alpha \mathbf{x}''(s,t) - \beta \mathbf{x}''''(s,t) + \mathbf{v}. \tag{2.10}$$

Ezen egyenlet paraméteres görbe segítségével történő megoldását *GVF snake-nek* nevezzük. Ezt numerikusan diszkretizálással és iterációs lépesekkel oldjuk meg – hasonlóan a tradicionális snake-ekhez. Habár a GVF snake végső konfigurációja kielégíti a (2.8) erőegyensúlyi egyenletet, általában nem reprezentálja az Euler egyenletet (2.7), mint energiaminimalizációs problémát. Ezen optimális tulajdonság elvesztését kompenzálja a GVF snake jelentősen javított teljesítménye.

2.4.1. Éltérkép

Először is az $I(x, y)$ képből származtatott $f(x, y)$ éltérképet definiálunk, melynek megvan az a tulajdonsága, hogy a képen az élek közelében nagyobb értékeket vesz fel. A képfeldolgozás szakirodalmában bármelyik szürkeárnyalatos vagy bináris képekre definiált éldetektáló algoritmust felhasználhatjuk, például a (2.3, 2.6)-ban bemutatott:

$$f(x, y) = -E_{ext}^{(i)}(x, y), \tag{2.11}$$

ahol $i = 1, 2, 3$ vagy 4. A GVF esetében az éltérkép tulajdonságai közül három fontos, tekintsük át most ezeket. Az első nagyon fontos tulajdonság, hogy a ∇f mindig az élek felé mutat, a második, hogy ezen gradiensvektorok csak az élek közvetlen közelében vesznek fel magas értékeket, a harmadik pedig, hogy a homogén területeken, ahol $I(x, y)$ majdnem konstans ∇f majdnem 0. Vizsgáljuk meg, ezen tulajdonságok hogyan befolyásolják a tradicionális snake viselkedését abban az esetben ha a gradienst használjuk éltérkép gyanánt mint külső erő. Az első számunkra kedvező tulajdonság miatt a snake a kezdeti inicializálásból egy stabil végállapotba fog

konvergálni az élek közelében. A második tulajdonság az ami miatt az elérési terület kicsi lesz, és a harmadik tulajdonság miatt a homogén területeknek egyáltalán nem lesz külső energiájuk. A két utolsó tulajdonság sajnos igen előnytelen számunkra, a módszert úgy oldották meg, hogy az éltérkép kedvező tulajdonságai megmaradjanak és kiterjesztették a gradinens térképet az élektől távoli homogén régiókra is, egy iteratívan terjeszkedő módszer segítségével, melynek egy ennél nagyobb előnye, hogy olyan vektorokat készít melyek belemutatnak a határvonal konkavitásaiba.

2.4.2. Gradient Vector Flow

A gradient vector flow mezőt egy olyan $\mathbf{v}(x,y) = [u(x,y), v(x,y)]$ vektormezőként definiáljuk, mely minimalizálja a következő energiafüggvényt:

$$\varepsilon = \int\int \mu(u_x^2 + u_y^2 + v_x^2 + v_y^2) + |\nabla f|^2 |\mathbf{v} - \nabla f|^2 dxdy. \tag{2.12}$$

A formula azt az alapelvet követi, ha nincsen adat egy ponton, akkor végezzünk egy simítást. Különösképpen megfigyelhető ez, ha $|\nabla f|$ kicsi, ekkor az energiának a domináns részét a parciális deriváltak négyzetösszegei adják, így egy simított lassan változó mezőt kapunk. A másik eset, ha a $|\nabla f|$ nagy, ebben az esetben az integrandus második része a domináns, így $\mathbf{v}$ majdnem teljesen ∇f értékéből származik, a homogén régiók esetében pedig lassú változtatással beállítja a megfelelő erőket. A μ paraméter egy szabályzó paraméter mely az integrandus első és második része közötti arányt szabályozza, ezt a paramétert a képen található zaj nagysága alapján kell beállítani (sok zaj esetén μ értékét növelni). Érdekességképpen megjegyzem, hogy az itt alkalmazott simítás – a (2.12) integrandus első fele – ugyanaz a művelet, melyet Horn és Schunck [14] használ optical flow detektáló algoritmusukban[4].

Courant és Hilbert [6] 1953-ban bemutatott értekezése alapján a GVF mezőt a következő Euler egyenletrendszer megoldásával kaphatjuk meg:

$$\mu\nabla^2 u - (u - f_x)(f_x^2 + f_x^2) = 0 \tag{2.13}$$

$$\mu\nabla^2 v - (v - f_y)(f_x^2 + f_x^2) = 0, \tag{2.14}$$

ahol ∇^2 a Laplace operátor. Megjegyzem, hogy a homogén régiókban – ahol az $I(x,y)$ konstans – mindkét egyenlet második része 0, ezért az $f(x,y)$ gradiens szintén 0, következésképpen ezen régiókban u-t és v-t a Laplace operátor határozza meg. Ezekből következik, hogy a GVF mező a régiók határaitól interpolálódik. Ez a magyarázata annak, hogy miért mutatnak a vektorok a határvonal konkavitásaiba.

[4] Horn és Schunck 1981-ben publikálták optical flow detektáló algoritmusukat, mely a klasszikus algoritmuok közöl a mai napig az egyik legközkedveltebb.

2.4.3. Numerikus Implemetáció

A (2.13) és (2.14) egyenleteket úgy oldhatjuk meg, hogy u-t és v-t az idő függvényeként tekintjük és megoldjuk a következő egyenleteket:

$$\begin{aligned} u_t(x,y,t) &= \mu\nabla^2 u(x,y,t) - [u(x,y,t) - f_x(x,y)][f_x(x,y)^2 + f_y(x,y)^2] \quad (2.15) \\ v_t(x,y,t) &= \mu\nabla^2 v(x,y,t) - [v(x,y,t) - f_y(x,y)][f_x(x,y)^2 + f_y(x,y)^2]. \quad (2.16) \end{aligned}$$

Ezeket az egyenleteket átírhatjuk a következő formába:

$$\begin{aligned} u_t(x,y,t) &= \mu\nabla^2 u(x,y,t) - b(x,y)u(x,y,t) + c^1(x,y) \quad (2.17) \\ v_t(x,y,t) &= \mu\nabla^2 v(x,y,t) - b(x,y)v(x,y,t) + c^1(x,y), \quad (2.18) \end{aligned}$$

ahol

$$\begin{aligned} b(x,y) &= f_x(x,y)^2 + f_y(x,y)^2 \\ c^1(x,y) &= b(x,y)f_x(x,y) \\ c^2(x,y) &= b(x,y)f_y(x,y). \end{aligned}$$

Bármelyik képekre alkalmazható gradiens operátort használhatjuk az f_x és f_y kiszámítására. A $b(x,y)$, $c^1(x,y)$ és $c^2(x,y)$ együtthatókat az iteratív eljárás előtt kiszámolhatjuk és fixálhatjuk. Legyen Δx és Δy a pixelek közötti távolság, és Δt két iteráció között eltelt idő. Ekkor a parciális deriváltak a következőképp közelíthetők:

$$\begin{aligned} u_t &= \frac{1}{\Delta t}(u_{i,j}^{n+1} - u_{i,j}^n) \\ v_t &= \frac{1}{\Delta t}(v_{i,j}^{n+1} - v_{i,j}^n) \\ \nabla^2 u &= \frac{1}{\Delta x \Delta y}(u_{i+1,j} + u_{i,j+1} + u_{i-1,j} + u_{i,j-1} - 4u_{i,j}) \\ \nabla^2 v &= \frac{1}{\Delta x \Delta y}(v_{i+1,j} + v_{i,j+1} + v_{i-1,j} + v_{i,j-1} - 4v_{i,j}). \end{aligned}$$

Ezt a (2.17) és (2.18)-ba helyettesítve a GVF mező a következő iteratív egyenletekből adódik:

$$\begin{aligned} u_{i,j}^{n+1} &= (1 - b_{i,j}\triangle t)u_{i,j}^n + r(u_{i+1,j}^n + u_{i,j+1}^n + u_{i-1,j}^n + u_{i,j-1}^n - 4u_{i,j}^n) + c_{i,j}^1 \triangle t \\ v_{i,j}^{n+1} &= (1 - b_{i,j}\triangle t)v_{i,j}^n + r(v_{i+1,j}^n + v_{i,j+1}^n + v_{i-1,j}^n + v_{i,j-1}^n - 4v_{i,j}^n) + c_{i,j}^2 \triangle t, \end{aligned}$$

ahol

$$r = \frac{\mu \Delta t}{\Delta x \Delta y}. \tag{2.19}$$

A Δt értékének módosításával tudjuk szabályozni a konvergenciát:

$$\Delta t \leq \frac{\Delta x \Delta y}{4\mu}. \tag{2.20}$$

Az implementációm C++-ban készült, egy 256×256 mértű képre 128 iteráció esetén 8,12 másodperc alatt készíti el a GVF mezőt.

2.4.4. Gradient Vector Flow kiterjesztése magasabb dimenziókra

Az eddig ismertetett módszer $\Re^2 \rightarrow \Re$-be történő leképezésekre, azaz szürkeárnyalatos képekre lett definiálva. A színes képek vagy például az optical flow $\Re^2 \rightarrow \Re^n, (n \geq 1)$ képző függvények, azaz a GVF-et nem tudjuk közvetlenül fehasználni. Kézenfekvő megoldás lenne egy bijektív leképezést adni ami ezen függvények értékkészletének halmazát egy dimenzióba képezi le, de ez adattorzulással járna. Ezért inkább az algoritmuson változtattunk egy kicsit [16], hogy több dimenzióban is megfelelően működjön. Tehát a feladatunk a következő: adott $f(x,y) : \Re^2 \rightarrow \Re^n$ és ehhez keressünk egy korrekt GVF-t. A több dimenziós függvények parciális deriváltja egy Jacobbi mátrix, mely a következőképpen írhatunk fel $\Re^2 \rightarrow \Re^n$-ben:

$$Jac(f(x,y)) = \begin{pmatrix} \frac{\partial f_1(x,y)}{dx} & \cdots & \frac{\partial f_n(x,y)}{dx} \\ \frac{\partial f_1(x,y)}{dy} & \cdots & \frac{\partial f_n(x,y)}{dy} \end{pmatrix}_{2 \times n}, \tag{2.21}$$

legyen $M \in \Re^{(2 \times n)}$, és $M = Jac(f(x,y))$. Bontsuk fel az M-et 2 sorvektorára, ezek legyenek $M_x, M_y \in \Re^n$:

$$M_x = (M_{1,1}, \ldots, M_{1,n}), M_y = (M_{2,1}, \ldots, M_{2,n}), \tag{2.22}$$

így a parciális deriváltak koordinátánként a két vektorban vannak. Mivel lehetséges, hogy bizonyos jellemzőket nem ugyanakkora súllyal akarunk figyelembe venni, szorozzuk meg egy α súlyvektorral ezeket. Tehát legyen $\alpha \in \Re^n$, ekkor

$$M'_x = \alpha \bullet M_x, M'_y = \alpha \bullet M_y \tag{2.23}$$

az új súlyozott irányok. Ebből a GVF-nél használt $f_x(x,y), f_y(x,y)$ iránykomponenseket úgy kapjuk meg, hogy vesszük az M'_x, M'_y vektorok normáit, azaz:

$$f_x(x,y) = \|M_x\|_k, f_y(x,y) = \|M_y\|_k, k = 1, 2, \infty, max. \tag{2.24}$$

2.4.5. Gradient Vector Flow általánosítása

Az előző alfejezetben megnéztük azt a két dimenziós esetet, mikor több mint egy jellemzőnk van. Most ezt általánosítjuk több dimenzióra. Adott egy $f(x_1, \ldots, x_m) : \Re^m \rightarrow \Re^n$. Ekkor $f(x)$ parciális deriváltja:

$$Jac(f(x_1, \ldots, x_m)) = \begin{pmatrix} \frac{\partial f_1(x_1,\ldots,x_m)}{dx_1} & \cdots & \frac{\partial f_n(x_1,\ldots,x_m)}{dx_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_1(x_1,\ldots,x_m)}{dx_m} & \cdots & \frac{\partial f_n(x_1,\ldots,x_m)}{dx_m} \end{pmatrix}_{m \times n}, \tag{2.25}$$

majd hasonlóan az előzőekhez vesszük a sorvektorokat: $M_1, \ldots, M_m \in \Re^m$, ahol:

$$\begin{aligned} M_1 &= (M_{1,1}, \ldots, M_{1,n}) \\ &\vdots \\ M_m &= (M_{m,1}, \ldots, M_{m,n}), \end{aligned}$$

ezután súlyozzuk $\alpha \in \Re^m$ vektorral a sorvektorokat, és végezetül képezzük a nomrákat, melyek a gradiens irányokat fogják adni:

$$\begin{aligned} f_{x_1}(x_1, \ldots, x_n) &= \|M_{x_1}\|_k \\ &\vdots \\ f_{x_m}(x_1, \ldots, x_n) &= \|M_{x_m}\|_k. \end{aligned}$$

Az itt kiszámított irányokat felhasználva tudjuk elkészíteni a GFV-et n dimenzióra.

2.4.6. Pontok felvétele és elhagyása a GFV snake-ből

Új pontok felvétele

Amikor incialzáltuk a snake-et és elkezdjük iteratív lépésekkel a GVF irányába mozgatni, előfrodulhat, hogy két pont nagyon eltávolodik egymástól és ekkor a közöttük lévő területre nem jut olyan pont amely ráhúzódhatna a korrekt pozícióra. Ez főleg konkáv részeken és szakadások közelében figyelhető meg. Ekkor jó ötletnek tűnik beszúrni egy újabb pontot a kontúrba, úgy, hogy az a két eltávolodott pontot összekötő egyenesen a pontok között középen legyen. Ezzel nem rontunk a snake állapotán, és az új pont elindulhat a konkáv részek irányába. A módszer egy lehetséges implemetálása, ha megkeressük a k legnagyobb távolságot a snake mentén és oda beszúrunk egy-egy pontot. Tapasztalataim szerint egy n pontot tartalmazó kontúrnál $k = n/10 \ldots n/100$ választása a legjobb. A (2.4)b. ábrán látható a beszúrás nélküli

eredmény. Megfigyelhető, hogy a szakadásba nem jutott pont és a konvex területekre nem konvergált be a snake.

(a) (b) (c)

2.4. ábra. Konkáv alakzat szakadással, az (a) ábrán az eredeti alakzat, (b) ábrán a GVF snake beszúrás és törlés nélkül ($n = 200$), (c) ábrán beszúrás és törléses GVF snake látható($n = 200, k = 2$).

Pontok törlése

Megismertük a pontok beszúrását, ám ha minden iterációban beszúrunk k pontot, akkor az i. iterációban már $n + ik$ db pontunk lesz, mely nem rontja el a snake jellemzőit, de a számítási időt jelentősen befolyásolhatja. Ezért érdemes minden iterációban kitörölni a feleslegesnek tűnő pontokat. Ezek olyan pontok amelyek túl közel esnek egymáshoz. Azaz minden (p_{i-1}, p_{i+1}) párosra vizsgáljuk meg milyen távol vannak egymástól, és a k legkisebbre töröljük ki a p_i pontot.

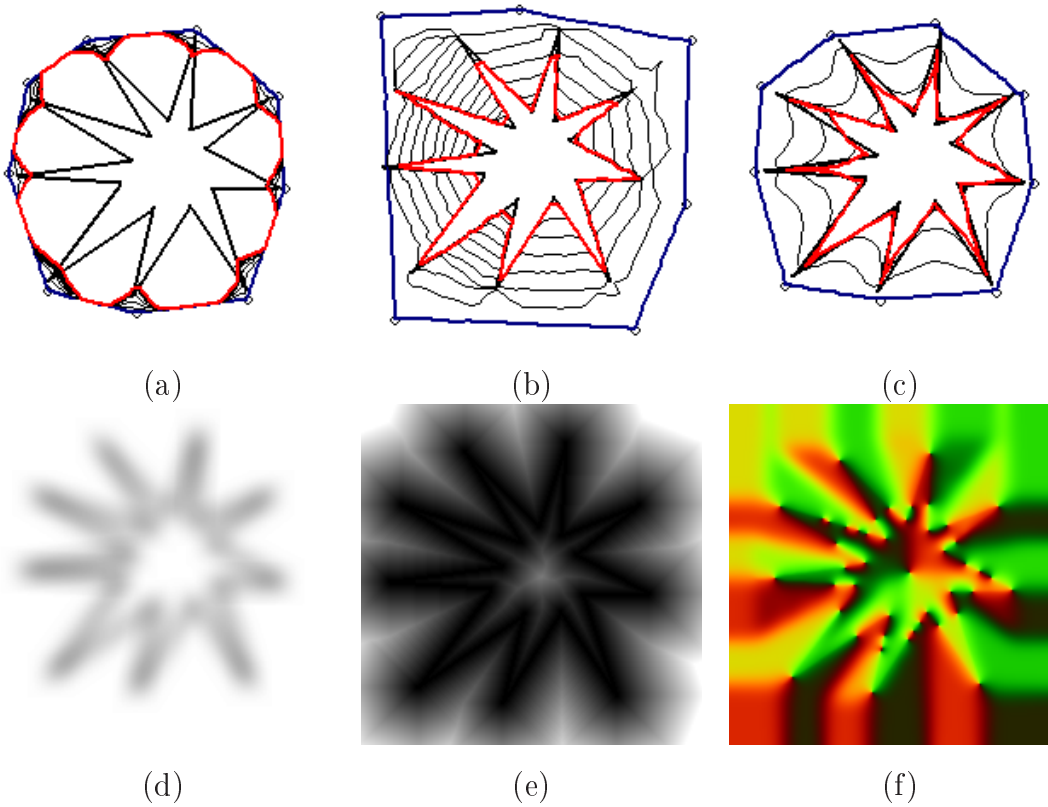
2.5. Kísérleti eredmények

A GVF tesztelését négy fázisban végeztem. Az első részben összehasonlítom a tradícionális módszerekkel. Ezután bemutatok olyan teszteseteket melyekre a GVF snake nagyon jól használható és olyanokat ahol nem javasolt az alkalmazása. Megvizsgálom hogyan működik színes képek esetén végül pedig mozgó objektumok detektálására használom optical flow segítségével.

2.5.1. A tradícionális módszerek és a GVF összehasonlítása

Elsőként egy olyan bináris tesztképet láthatunk (2.5), melyen egy csillag található. Kipróbáltam mindhárom bemutatott módszert a képen. Az (a) és (d) ábrákon a tradicionális módszerrel kapott eredmény, a simításnál $\sigma = 25$ értéket használtam, láthatjuk, hogy a görbe csak azon területekre húzódott rá, melyek az inicializálás

közelében voltak. A (b) és (e) ábrákon a távolságtérképet felhasználva készítettem a vektormezőt, láthatjuk, hogy a snake megfelelően jó szegmentálást készített, köszönhetjük ezt annak, hogy nincsenenk konkáv részek a képen. Végül a GVF snake, mely megfelelő pontossággal találta meg a keresett alakzatot (c) és (f) ábrák.



(a) (b) (c)

(d) (e) (f)

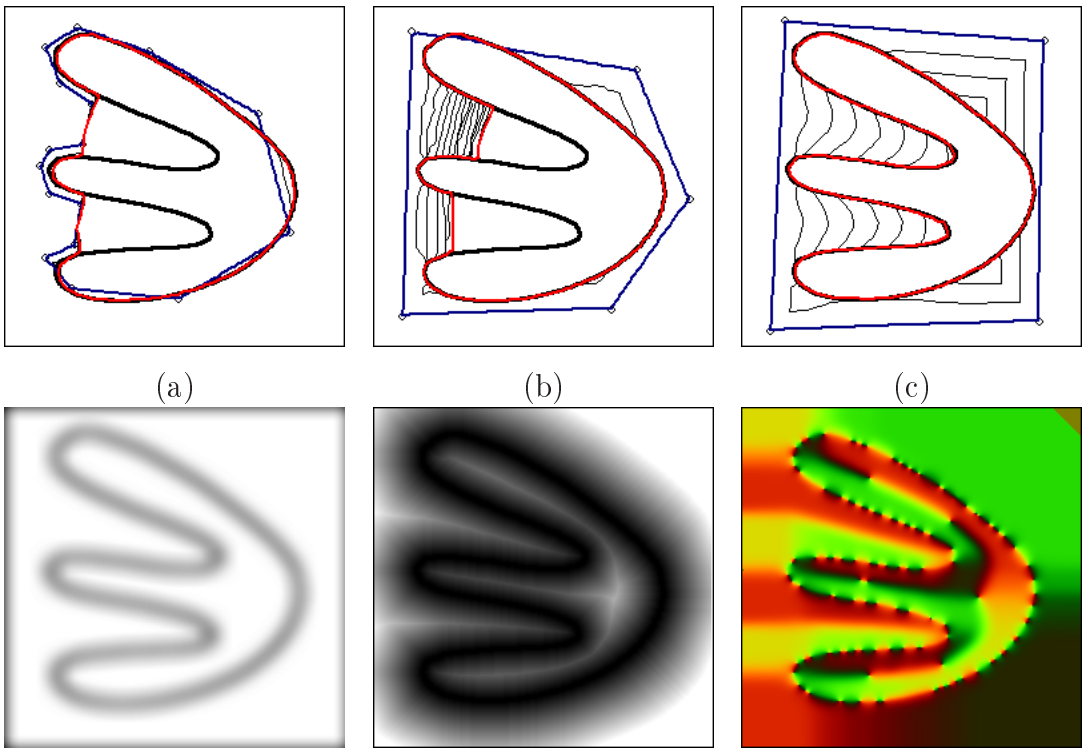
2.5. ábra. Példa a három tárgyalt eljárásra, az (a) és (c) ábrákon a simítás után számított gradiensen alapuló módszer, a (b) és (e) ábrákon a Chamfer távolság felhasználásával kapott eredmény, míg a (c) és (f) ábrákon a GVF látható.

A (2.6). ábrán arra az esetre láthatunk példát amikor egyik tradícionális módszer sem képes a kontúrt a megfelelő helyre eljuttatni, csak a GVF-mező.

2.5.2. A Gradient Vektor Flow előnyei és hibái

Az előző alfejezetben láthattuk, hogy a GVF bináris képekre minden olyan esetben jól működik, ahol nincsen konvex része az alakzatnak, most olyan esetet mutatok, melyet GVF-snake-kel nem lehet szegmentálni. A (2.7) ábrán egy olyan alakzatot látunk, melynek van konvex része is, az (a) mutatja mi történik ha tetszőleges helyen inicializáljuk a snake-et, ekkor nem képes a megfelelő szegmentálást elvégezni, a (b) ábrán megfelelő inicializálás mellett láthatjuk a végerdményt, ekkor már megfelelő lett a szegmentálás.

Természetesen az igazi kihívást a szürkeányalatos képek jelentik. A (2.8). ábrán egy orvosi képet láthatunk, a belső ellipszis alakú részt akartuk szegmentálni, és mint



(a) (b) (c)

(d) (e) (f)

2.6. ábra. Példa arra az esetre, amikor a távolságtranszformáción alapuló vektormező sem képes a snake-et a megfelelő helyre irányítani. Az (a) és (d) képeken a tradícionális gradiens segítségével nyert eredmény ($\sigma = 25$), a (b) és (e) mutatja a távolságtranszformáció segítségével készített szegmentálást, a (c) és (f) ábrákon pedig a GVF látható.

ahogy az a (b) ábrán jól látható megfelelően sikerült, a GVF mező elkészítéséhez 128 iterációs lépést végztem az éltérkép meghatározásakor Gauss-szűrőt használtam, melyet $\sigma = 13$ szórással inicializáltam. A nagy σ érékekre gyakran lehet szükségünk szürkeárnyalatos képek esetében, mivel a módszer meglehetősen zajérzékeny.

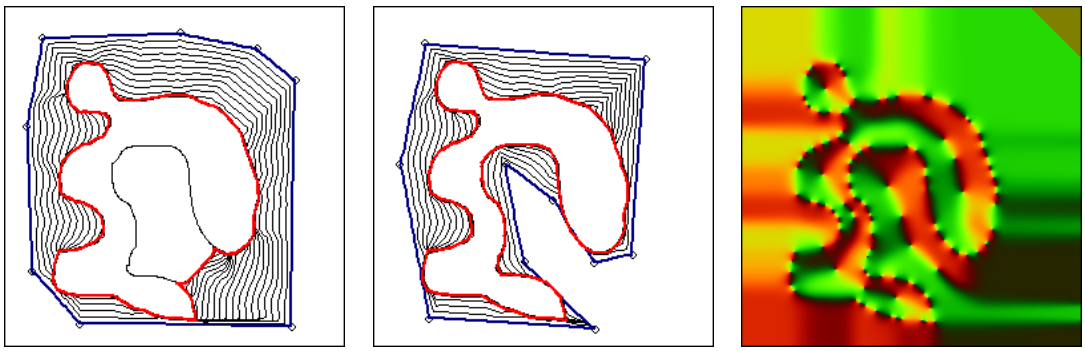
A (2.9). ábrán egy kézfejre ráhúzódó snake látható, mivel a képen kevés a zaj, így $\sigma = 7$ nagyságú szűrő elegendőnek bizonyult, a kontúr 200 pontból áll, és k értéke 2 volt[5]. Az ábra (b) részén a GVF vektormezőt láthatjuk.

2.5.3. Színes képek és mozgás szegmentálása GVF segítségével

Szegmentálás mozgás alapján

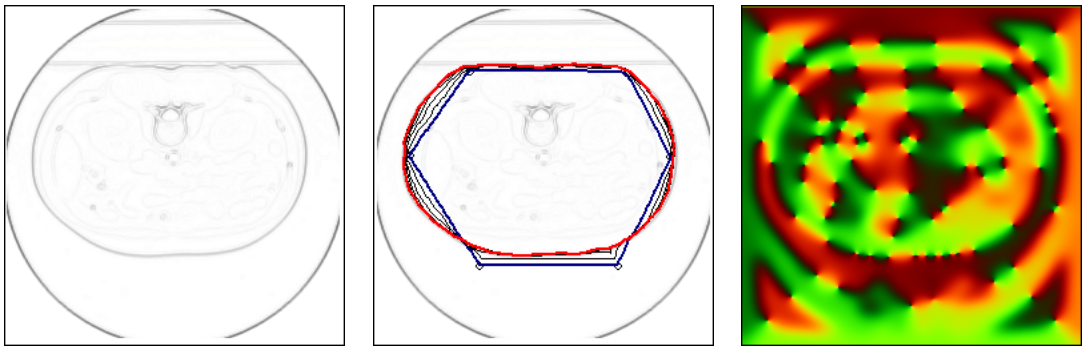
A (2.10 a) ábrán egy asztaliteniszező játékos látható, mely egy képsorozat egy szekvenciája, a (b) ábrán a kép és a rákövetkező kép közötti mozgások vektorai láthatók.

[5] A 2.4.6 részben bemutatott iterációnkénti beszúrt és törölt pontok száma.



(a) (b) (c)

2.7. ábra. Az (a) ábrán példát láthatunk arra az esetre amikor a GVF-snake sem képes a konvex területekre rásímulni, bár megfelelő inicializálással ez a probléma kiküszöbölhető (b).



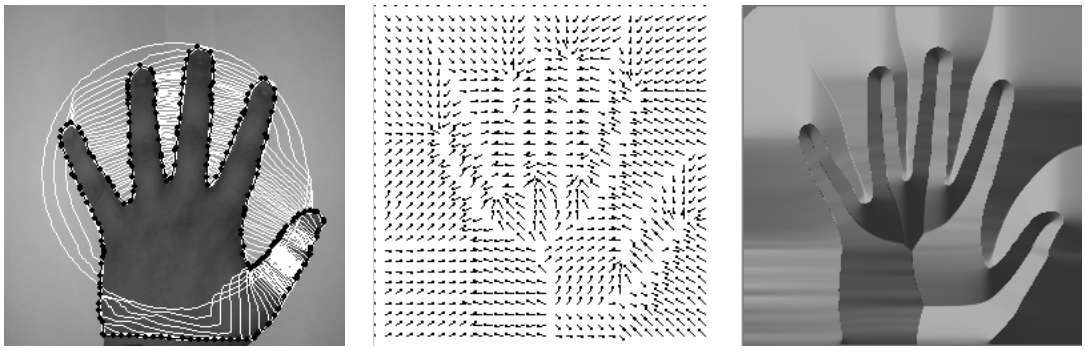
(a) (b) (c)

2.8. ábra. Példa egy orvosi alkalmazásra, az (a) ábrán látható az eredeti kép, a (b) a szegmetált képet míg a (c) a GVF mezőt mutatja.

Végül a szegmentálás eredménye a (c) ábrán. A Gradient Vector Flow paraméterei: $n = 200, k = 3$. A kontúrt inicializáláskor kör alakú volt, és a játékos kezét akartuk szegmentálni az optical flow segítségével, így a két jellemző az optical flow vektorainak x és y koordinátái az α súlyvektor komponensei egyenlők voltak. A (c) ábrán látható a snake mozgása, a fekete kontúrokat minden 5. iteráció után rajzoltuk ki. A snake kb. 100 iteráció után bekovergált.

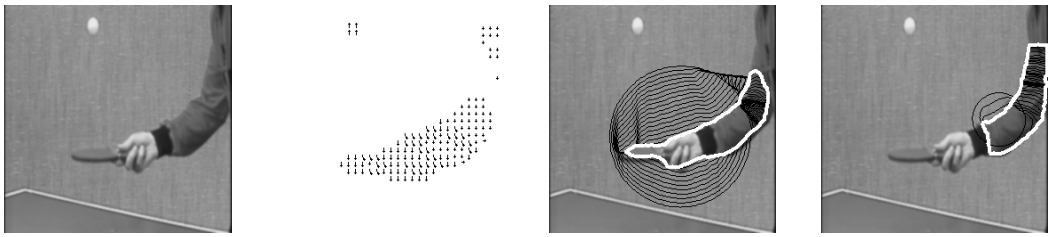
Szegmentálás szín alapján

A (d) ábrán az asztaniteniszező karjának szegmentálását találjuk meg, látható, hogy nem sikerült az egész karra ráhúzódnia a kontúrnak, mert felül az alakzat konvex, azaz szélesedik az átmérője. A színek CIE L*u*v* színtérbe lettek konvertálva. A kontúr inicializálása egy körrel történt, a kör a játékos karján helyezkedik el.



(a) (b) (c)

2.9. ábra. Gradient Vector Flow segítségével előállított képek. Az (a) ábrán egy kézfejre ráhúzódott snake, (b)-n a vektormező és a c-n a vektorok színskálán ábrázolva.



(a) (b) (c) (a)

2.10. ábra. Kép szegmentálása mozgás alapján, az (a) ábrán az eredeti kép látható a (b)-n az optical flow, a (c)-n pedig a szegmentálás eredménye a kontúr mozgása, a (d) ábrán a játékos karjának szegmentálása szín alapján.

3. fejezet

Mozgás detektálása

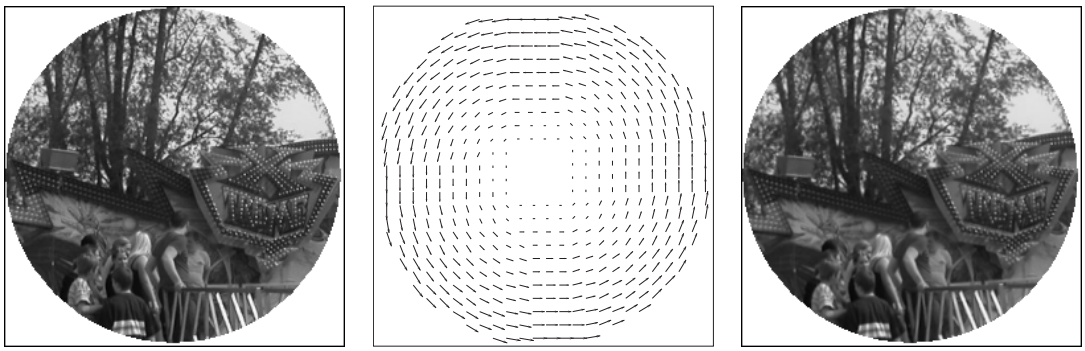
3.1. Bevezetés

Mozgás detektálására sok robosztus és pontos algoritmus született. Dolgozatomban a detektálást *optical flow* meghatározásával végzem. Bemutatom a téma eddig megjelent fontosabb publikációit, ismertetem az általunk készített és publikált eljárásokat, melyek közül az elsőként bemutatásra kerülő egy real-time algoritmus. Bemutatok egy *expectation-maximization* módszeren alapuló objektumkövetésre használható eljárást, végül egy általunk készített robotot, melyet egy PC vezérel és mozgó kamera segítségével lehetőséget biztosít a bemutatásra kerülő algoritmusok széleskörű felhasználására.

A mozgás detektálásának egyik közkedvelt és széleskörűen használható eszköze az *optical flow*, mely egy képsorozat két egymást követő képkockája között végbemenő mozgások vektorait tartalmazó mező. Az optical flowra láthatunk példát a (3.1). ábrán, az (a) és (c) mutatja a két kép-szekvenciát, a második az óra járásával ellentétes irányban 3°-kal el van forgatva az elsőhöz képest, a (b) ábrán a képekhez tartozó optical flow látható.

Az optical flow detektálásról elsők között K. Prazdny 1979-ben megjelent írásában olvashatunk [26]. Az első még ma is méltán közkedvelt és sokszor alkalmazott mószert B. D. Lucas és T. Kanade 1981-ben publikált cikkükben mutatták be [21], az eljárás egy simítással kezdődik Gauss-normál eloszlást felhasználva ($\sigma = 1,5$ paramétert javasolnak), majd egy idő és tér szerinti differnciálást végeznek 4-pontos központi differencia segítségével. A teljes globális mozgást a kép minden pixelére súlyozott legkisebb négyzetek módszerével keresik meg. A lokális mozgások segítségével kiszámolják a mozgások normálvektorait, majd adnak egy biztonsági mértéket a becslésre sajátvektor számítással.

A másik, a témakörben alapműnek számító publikáció szintén 1981-ben született, alkotói B. K. P. Horn és B. G. Schunk [14]. A módszerük hasonló az előző



(a) (b) (c)

3.1. ábra. Példa optical flow-ra, az (a) és (c) ábrákon az erdeti képsorzat két képkockája, a második az órajárásával ellentétes irányban 3°-kal lett elforgatva az elsőhöz képest. A (b) ábrán az optical flow látható.

dolgozatban tárgyalt eljáráshoz, de a vektorok finomítására egy itérációs eljárást használnak. Általam a probléma megoldására legjobbnak tartott algoritmust 1989-ben P. Anandan publikálta [1]. A módszer az eddgi bemutatottaknál gyorsabb, de nem megfelelően gyors real-time implementációra. Az eljárás egy képpiramis építésével kezdődik, mely során mindkét bemeneti képből kisebbeket készít, majd ezen kisebb képeken a mozgásra durva becslést ad, folyamatosan haladva a finomabb felé, opcionálisan egy simító szűrést is javasol. 1990-ben Singh [29] egy egészen új, két lépésből álló korreláción alapuló mozgásdetektáló módszert mutat be. Az első lépésben egyszerű korreláció segítségével egy a különbségek négyzetösszegéből (SSD) álló felületet hoz létre. A második lépésben iteratív egyenletek felhasználásával simítja az SSD-t, bemutat egy lehetőséget gyors mozgások kezelésére Gauss-piramis felhasználásával. A Gauss-piramissal később még bővebben foglalkozom [4]. D. J. Fleet és A. D Jepson 1990-ben egy gradiensen alapuló módszert mutattak be [11], Gabor-filterek felhasználásával.

Az eddigiekben áttekintettem milyen tradicionális módszerek születtek a témában, most pedig bemutatok néhány az elmúlt tíz évben készült jelentősebb vagy valamilyen szempontból érdekesebb munkát. A francia IRSIA/INRIA-n dolgozó P. Bouthemy és F. Heitz 1993-ban [13] publikálták optical flow detektáló algoritmusukat, melyben véletlen Markov-mezőket (MRF) használnak. Először az élek mentén keresik meg a mozgásokat, majd MRF segítségével korrigálnak. Bouthemy és E. Francois [2] bemutat egy szintén MRF-et használó optical flow detektáló módszert képszegmentálásra. Az előző módszert továbbfejlesztve enerigaminimalizációval mozgás alapú képszegmentálást mutat tanulmányában 1998-ban É. Mémim és P. Pérez [22].

Daniel Cremers PhD értekezésében mozgás alapú szegmentálást mutat be aktív

kontúrok segítségével [7], [8]. A mozgás mellett a priori tudást is visz a rendszerbe a szegmentálandó objektum alakjáról.

Végül a mozgásdetektálás egy más irányú megközelítéséről, az ipari és gyakorlati alkalmazásokról szeretnék szólni. 1998-ban jelent meg Y. Raja, J. McKenna és S. Gong cikke az ACCV-n [27]. Ők mozgáskövetést az Expectation-Maximalisation módszerrel alkalmaznak, kihasználva azt, hogy léteznek olyan színterek, melyek esetében az objektumok színei különböző fényviszonyok esetén is hasonló távol vannak egymástól. Algoritmusukban HSV színteret használnak. R. Cucchiara, M. Piccardi, A. Prati és N. Scarabottolo egy hardvert építettek (PCI-kártya ami egy normál PC-be helyezhető) [9], mely mozgó járművek real-time megfigyelésére használható.

Munkáim során igen sokszor felmerült a kérdés hogyan lehetne mérni az optical flow pontosságát. Még, ha ismert az eltolási mező, akkor sem sikerült egy minden esetben korrekt hibamértéket adni, bár T. Lin és J. L. Barron [20] készítettek egy értekezést, melyben a különböző tradicionális módszereket hasonlítják össze. Cikkükben adnak egy hibamérőszámot, de ez tapasztalataim szerint nem használható két képsorozat, csak két módszer egyazon sorozaton produkált eredményeinek összehasonlítására.

Bemutatok egy triviális mozgásdetektáló algoritmust, annak javításait, kiterjesztéseit, majd megmutatom miért nem használható gyakorlati alkalmazásokban. Ezután egy olyan módszert ismertetek, mely $O(n \lg n)$ futásidejű, így már valós időben történő optical flow meghatározásra is felhasználhatjuk. Az algoritmus során a két bemeneti képből egy piramist készítünk. A piramis magasabb szintjein meghatározzuk a durva mozgásokat, majd ezeket finomítgatjuk az alacsonyabban lévő szinteken, egy simítást is alkalmazunk, így az esetleges rosszul detektált vektorokat a környezetükhöz igazíthatjuk. Megvizsgálunk néhány hibamérésre kidolgozott technikát, és bemutatom az elért eredményeket.

D. Mumford és J. Shah 1989-ben publikált cikkében [23] egy egyszerű képszegmentálási egyenletet fogalmaznak meg, mely kiterjeszthető sok speciális esetre. Az egyenlet meg- oldására szolgáló módszerekről nagy mennyiségű irodalom áll rendelkezésünkre. Ezek közül szeretném kiemelni J. A. Sethian: *Level set methods* című könyvét [28], amiben a *level set* módszereket tárgyalja, melyek megoldást szolgáltatnak a szegmentálási egyenletekre. A könyvében a képfeldologzás igen sok területére kiterjeszti a módszert, többek között egy egyszerű optical flow detektálására használható algoritmust is bemutat.

A dolgozat végén részletesebben bemutatok egy általam készített és programozott robotkamerát, mely mozgó objektumok követésére alkalmas, a megvalósítás során két nehézségbe ütköztem, először meg kellett oldani a kamerából származó

adatok valós-idejű feldolgozását, másodszor pedig a robot kommunikációját meg kellett valósítani és össze kellett hangolni a vezérlést a számítógéppel.

Végül két érdekes jelenséget –egyben nehézséget– szeretnék bemutatni mely gyakran tapasztalható optical flow detektálása közben. Az első homogén területek mozgásának meghatározáskor figyelhető meg, mivel az optical flow detektálás szín alapján történik a homogén területek mozgását nem tudjuk meghatározni, ugyanez a helyzet abban az esetben is, ha egy homogén háttér előtt a háttérrel megegyező színű tárgyak mozognak, illetve azonos színű tárgyak mozognak egymás alatt/felett. Másik érdekes jelenség az ún. körlap effektus, mely kör-szerű objektumok mozgásának detektálása során lép fel, a mozgásvektorok nem a helyes irányba, hanem a körvonal legközelebbi pontjába mutatnak.

3.2. Gyors algoritmus optical flow meghatározására

Most bemutatok egy gyors algoritmust optical flow meghatározására de előtte nézzük meg a triviális megoldást illetve néhány javítását. Legyen a két bemeneti kép $I_1, I_2 \in \mathbb{R}^{M \times N}$, a kép x, y koordinátájú pontját jelöljük $I_1[x, y]$-nal. $\overline{OF[i,j]}$ (az *optical flow*) jelölje azt, hogy az $I_1[i, j]$ pixelén lévő objektum milyen irányban és mennyit mozgott, azaz hol található az I_2 képszekvencián ($[i, j] + \overline{OF[i, j]}$).

3.2.1. Triviális algoritmus a mozgás meghatározására

A triviális algoritmus az első képkocka minden pixelére megvizsgálja a második képen levő összes pixelt és közülük a leghasonlóbbat választja ki:

$$\overline{OF[i,j]} = \arg \min_{\substack{k=1\ldots N-1 \\ l=1\ldots N-1}} |I_1[i,j] - I_2[k,l]| - [i,j]. \tag{3.1}$$

$I_1[i, j]$ és $I_2[i, j]$ az első és második képszekvencia pixelei. Ezen algoritmus javított változata, amikor egy adott pixel valamilyen alakú és nagyságú körmyezetét vizsgáljuk. Lehetséges a környezet pixeleit különböző súlyokkal figyelembe venni és a különbségek négyzetösszegeinek minimumát (RMSE - Root Mean-Square Error) venni, ekkor:

$$\overline{OF[i,j]} = \arg \min_{\substack{k=1\ldots N-1 \\ l=1\ldots M-1}} \left(\sqrt{\frac{1}{E} \sum_{\forall (q,r) \in S} c_{(q,r)} (I_1[i+q, j+r] - I_2[k+q, l+r])^2} \right), \tag{3.2}$$

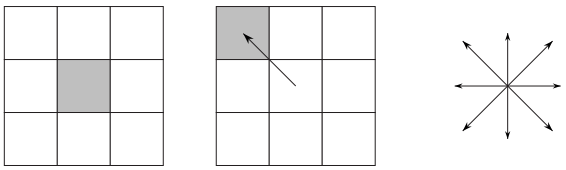
ahol S a vizsgált környezet pontjainak halmaza, $E = |S|$ és $c_{(q,r)}$ a minta (q, r). elemének súlya. Ezen algorimtus futása ugyan sokkal lassabb mint az egyszerű változaté de alaposabb vizsgálatot végez és pontosabb megoldást szolgáltat. A triviális algoritmus futásideje a kép pixelszámának másodfokú polinomiális függvénye,

$\Theta((M \times N)^2)$, ahol $M \times N$ a kép mérete. Ez gyakorlatban nem használható, ezért a probléma megoldására most bemutatunk egy $\Theta((M \times N)lg^2(M \times N))$ futásidejű algoritmust.

3.2.2. Maximálisan egy pixel nagyságú mozgások meghatározása

Ha feltételezzük, hogy a képen egy pixel nagyságúnál nagyobb mozgások nem lehetnek, akkor könnyebb dolgunk van. Ebben az esetben az előző részben bemutatott mószerek valamelyikét használva számíthatjuk a mozgásvektorokat, de csak egy pixel nagyságú környezetet vizsgálva (3.2).

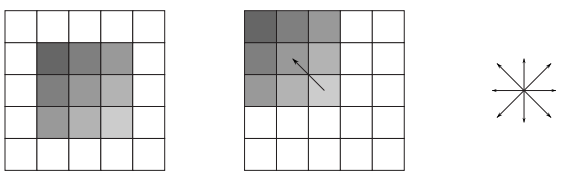
$$\overline{OF[i,j]} = \arg \min_{\substack{-1 \le x \le 1 \\ -1 \le y \le 1}} |I_1[i,j] - I_2[i+x, j+y]| \tag{3.3}$$



3.2. ábra. Maximálisan egy pixel nagyságú mozgás meghatározása egyszerű abszolút különbség felhasználásával.

Így két dimenziós mozgásvektorokat kapunk, melyek értékei $(-1 \ldots 1)$ közöttiek lesznek. Ez a módszer nagyon gyors eredményt szolgáltat, de nem minden esetben korrekt. Ennek javított változata amikor a keresett pixel valamilyen környezetében vizsgáljuk az RMSE-t, célszerű ezt súlyozni a fent bemutatott módon. Nézzünk néhány kipróbált módszert maximum egy pixel nagyságú mozgások meghatározására. Először egy egyszerű de a gyakorlatban jól bevált eljárás, a pixel 3×3-as környezetét vizsgáljuk és a legkisebbet feltételezzük korrekt megoldásnak (3.3).

$$\overline{OF[i,j]} = \arg \min_{\substack{-1 \le x \le 1 \\ -1 \le y \le 1}} \left| \sum_{u=-1}^{1} \sum_{v=-1}^{1} (I_1[i+u, j+v] - I_2[i+u+x, j+v+y]) \right| \tag{3.4}$$



3.3. ábra. Javított eljárás maximálisan egy pixel nagyságú mozgás meghatározására.

Ennek a módszernek egy javított változata, ha nem 3×3-as ablakot használunk, hanem változó méretűt és súlyozzuk az elemeket egy Gauss normális eloszlású mátrixszal, mely mátrix elemeinek összege 1. Ekkor:

$$f_{\mu,\Sigma}(\overline{x}) = \frac{1}{2\pi|\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(\overline{x}-\mu)^T\Sigma^{-1}(\overline{x}-\mu)} \tag{3.5}$$

normális eloszlás sűrűségfüggvénye segítségével számoljuk ki a Gauss mátrixot a következőképpen:

$$G'[i,j] = G[i,j] / \sum_{k,l=-\lfloor R/2 \rfloor}^{\lfloor R/2 \rfloor} G[i,j], \tag{3.6}$$

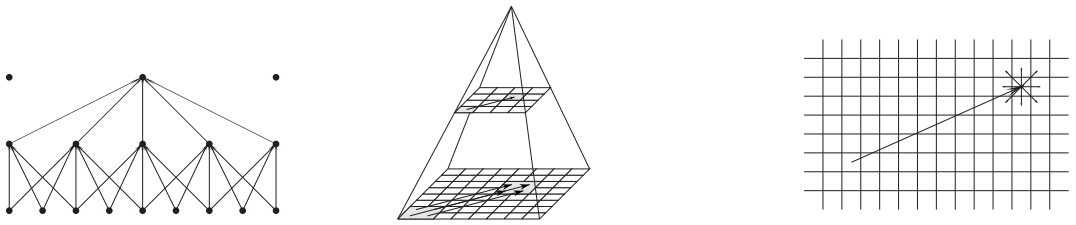
ahol R a környezet nagysága, így a mozgásvektorok keresése a következőképp zajlik.

$$\overline{OF[i,j]} = \arg \min_{\substack{-1 \le x \le 1 \\ -1 \le y \le 1}} \left(\sqrt{\sum_{u=-\lfloor R/2 \rfloor}^{\lfloor R/2 \rfloor} \sum_{v=-\lfloor R/2 \rfloor}^{\lfloor R/2 \rfloor} G'[u,v](I_1[i+u, j+v] - I_2[i+u+x, j+v+y])^2} \right), \tag{3.7}$$

Az eddigiek során megismertük hogyan lehet maximum egy pixel nagyságú mozgásokat meghatározni, most megnézünk egy eljárást, mely segítségével a nagyobb mozgásokat egy pixel nagyságúra redukálhatjuk.

3.2.3. Piramis nagyságának meghatározása és felépítése

Tekintsünk egy képet, amelyen egy x nagyságú mozgást találhatunk, ha ezt a képet n-szeresére nagyítjuk, akkor a mozgás is n-szeres lesz. Ezt az állítást felhasználva, ha felére csökkentjük a képet akkor a mozgások is felére csökkenek. Így ha a képet x-ed részére csökkentjük, akkor a mozgás pontosan 1 egység nagyságú lesz. Itt bemutatunk egy eljárást, amely segítségével igen gyorsan felére csökkenthetjük a kép méretét, az eljárás a Gauss piramis [4] felépítésének része.



(a) (b) (c)

3.4. ábra. A Gauss piramis kiszámításának menete (a) ábra, a visszafelé dolgozó stratégia (b) ábra, a becslés finomítása (c) ábra.

Az elkészítéshez egy normális eloszlású $N(0,1)$ paraméterű fentebb bemutatott Gauss mátrixot használunk. Egy $2n+1 \times 2n+1$ nagyságú képből indulunk és egy

$n+1 \times n+1$ mértűt készítünk. Konvolúciót alkalmazunk, a (3.4) a. ábrán látható az 1 dimenziós megvalósítás.

Az eljárást alkalmazva elértük, hogy a képet lineáris futásidejű algoritmussal felére csökkentettük. Itt jegyezzük meg, hogy több más lehetőségünk is van a kép átméretezésére, például a lefedett pixelek átlaga, esetleg súlyozott átlaga. Így elértük azt, hogy a mozgások a képen a felére csökkentek. A gondolatmenetet tovább folytatva a kapott képet ismét felére csökkentjük és így a mozgások negyedére, nyolcadára,... csökkennek. Így az x nagyságú mozgás a k. lekicsinyítés után $x/2^k$ nagyságú lesz. Már megismertük a maximum 1 pixel nagyságú mozgások detektálását. Ha tudjuk, hogy maximum k nagyságú mozgások vannak a képen akkor $\log_2 k$ darab képből kell állnia a piramisnak, így a $\log_2 k$.-on már csak maximum 1 pixel nagyságú mozgások lesznek.

3.2.4. Visszafelé dolgozó stratégia

A piramis felépítése után a legfelső képen meghatározzuk a maximum 1 pixel nagyságú mozgásokat, ezután átugrunk az alatta lévő szintre, itt az imént megállapított mozgások kétszer akkorák lesznek ám ez nyilván csak egy durva becslés. Tehát az $i+1$. szinten lévő Optical Flow-ból úgy készítjük a nagyobb i. szinten lévőt, hogy az $OF_i[2k, 2l]$, $OF_i[2k+1, 2l]$, $OF_i[2k, 2l+1]$ és $OF_i[2k+1, 2l+1]$ elemei megkapják az $OF_{i+1}[k, l]$ vektor hosszának kétszeresét (3.4) b. ábra. Ezek után jön az algoritmus egyik kulcslépése, ez a finomítás, amley során a vektorokról lévő durva becslésünket finomítjuk. A finomítás a durva becslés 3×3-as környezetében megkeresi pontosan melyik pixel is lehet a második képen a mozgás korrekt helye. Ezt pedig a 3.2.2 fejezetben ismertetett maximálisan egy pixel nagyságú mozgások detektálására szolgáló módszer segítségével végezzük (3.4) c. ábra).

3.2.5. Simítás

Az algoritmus végrehajása során keletkeznek olyan vektorok az optical flow-ban, amelyek hibás vagy részben hibás megoldások. Ezen hibák kiküszöbölésére alkalmazzuk a lágyítást. A szűrő egy konvolúciós maszk alkalmazása a vektorfolyamon.

Többféle maszkot is használhatunk a módszerhez, ezek egyike például a következő:

$$\begin{pmatrix} \frac{1}{16} & \frac{2}{16} & \frac{1}{16} \\ \frac{2}{16} & \frac{4}{16} & \frac{2}{16} \\ \frac{1}{16} & \frac{2}{16} & \frac{1}{16} \end{pmatrix}, \tag{3.8}$$

ennél jobb megoldás a normális eloszlású Gauss mátrix segítségével elvégezni a simítást. Tapasztalatok szerint a legjobb megoldást az $N(0.0, 1.5)$ paraméterekkel

érhető el. A lágyítást az első három szinten nem ajánljuk, mert ezeken a szinteken a mozgások igen kicsik és ezzel elronthatjuk az eredményt. A 6a ábrán látható az eredeti kép, a 6b ábra a simítás nélküli a 6c ábra pedig a simított mozgásokat ábrázolja.

(a) (b) (c)

3.5. ábra. Asztaliteniszező első szekvencia (a), simítás nélküli mozgásvektorok (b) és simított vektorok (c).

3.2.6. Hiba mérése, tradicionális tesztsorozatok

Először az optical flow hibájának mérésére mutatunk be módszereket. Az első Lin és Barron értekezésében olvasható [20]. Az első szekvencia és az optical flow segítségével megpróbáljuk előállítani a következő szekvenciát. Majd az előállított kép és a második szekvencia közötti RMS_{error}-t definiáljuk hibának. Formálisan a kép előállítása:

$$\forall i,j : I_{shynt}[i + OF[i,j].x, j + OF[i,j].y] = I_1[i,j], \tag{3.9}$$

a hiba számításának módja:

$$RMS_{err} = \sqrt{\frac{\sum_x \sum_y (I_2[x,y] - I_{shynt}[x,y])^2}{M \times N}}. \tag{3.10}$$

Kísérleteim alapján azt mondhatom a bemutatott mérték nem használható jól egy módszer különböző tesztképeken adott eredményeinek összehasonlítására, viszont két módszer összehasonlítására megfelelő. A problémák akkor léptek fel amikor meg akartuk mondani, hogy egy bizonyos módszer milyen típusú mozgások meghatározására jó vagy nem jó, például: forgatás, eltolás, átméretezés, kisebb vagy nagyobb mozgások... . Egy mértéket szerettünk volna adni két vektor hasonlóságára, mely 0, ha a két vektor azonos és 1 ha a vektorok teljesen különbözőek, de már azt is nagyon nehéz megfogalmazni mikor hasonlít vagy nem hasonlít egymásra két vektor, az összehasonlítás közben két művelettel is gond lehet, az első amikor a két vektor különbségét szeretnénk képezni, torz eredményt kapunk – gondoljunk csak bele egy 2 hosszúságú vektor helyett 3 hosszúságút kapunk a különbség 1, mely

50% hibát jelent, ugyanez egy 10 hosszú 11-nek detektált vektor esetében csak 10%, holott mindkét esetben 1 pixelnyi volt az eltérés. A másik az osztásnál jelentkező probléma, mivel igen sok $\overline{0}$ vektorunk van sok 0-val való osztás lenne. Megoldja a problémát a vektorok ε-nal való eltolása, de ez nem túl elegáns megoldás és a $-\varepsilon$ vektoroknál az eredeti problémába ütközünk.

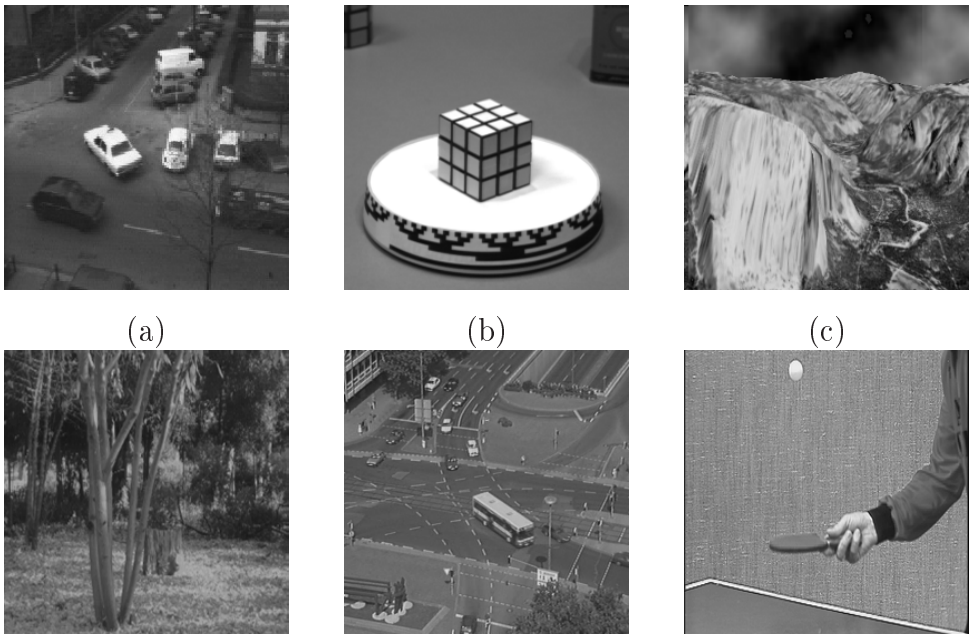
Több különböző módszert próbáltunk de úgy gondolom még nem sikerült megtalálni a megfelelő hibamértéket két optical flow összehasonítására. Most bemutatom az eddigi próbálkozásaimat:

- Külön kezeljük a vektorok hossz és szögbeli eltérését, majd a két érték súlyozott összegét képezzük (sajnos a $\overline{0}$ szögeltérését itt sem tudjuk kezelni).
- Különböző olyan jellegű próbálkozások, hogy a vektorok hosszának különbségét vagy hányadosát veszem, de a fent leírt okok miatt ez sem vált be.
- A két vektor hányadosának vagy különbségének egy függvényét venni, mely függvény kis eltérés esetén alig változik, míg nagyobb eltérés esetén gyorsabban. Egy érdekes megközelítés, ha egy tangens függvényt elnyújtunk az "x" tengely mentén majd a vektorok vektoriális távolságának hosszát behelyettesítjük, természetesen a szükséges normalizálásokkal.

Ha azonban nem ismerjük a korrekt eltolási mezőt mint általában valós képek esetében akkor nem tudunk ilyen módszereket alkalmazni. Ilyenkor tapasztalataim szerint a 3.9 és a 3.10 egyenleteket felhasználva megfelelő mértéket kaphatunk hiba mérésére.

Szükségesnek tartom még bemutatni a mozgásdetektálási algoritmusok tesztelésére használt tradicionális tesztsorozatokat.

A (3.6) ábra (a) részén a legismertebb Hamburgi taxi sorozat, melyről a későbbiekben még bővebben szólok, a (b) ábrán egy Rubik-kocka látható mely a sorozat folyamán körbe mozog, a (c)-n az un. Yosemite képsorozat, melyen egy sziklás hegyet rögzítve mozog a kamera. A (d) ábrán az SRI-tree sorozat látható, mely egy tisztást ábrázol előtérben egy fával. Az (e) sorozat egy közlekedési kép, melyről szintén szó lesz a későbbiekben. Az utolsó képen egy asztalitenniszező játékost látunk, a sorozat nem csak a mozgásdetektáló de a különböző szín alapú szegmentáló algoritmusok tesztelésére is használható.



(a) (b) (c)

(d) (e) (f)

3.6. ábra. A mozgásdetektálási algoritmusok tesztelésére leggyakrabban használt tesztsorozatok.

3.3. Az Optical Flow simítása a Mumford-Shah energiaminimalizációs eljárás segítségével

Mumford és Shah 1989-ben publikált cikkükben [23] bemutattak egy általános szegmentálási módszert, melyben a szegmentálást egy *energia-funkcionál* segítségével írták fel:

$$E(f,\Gamma) = \mu^2 \int\int_R (f-g)^2 dxdy + \int\int_{R-\Gamma} \|\nabla f\|^2 dxdy + \nu|\Gamma|, \tag{3.11}$$

ahol f a szegmentálás, g az eredeti kép, Γ a határ a régiók között és R a régiók összessége. Az első rész kicsi, ha a g hasonlít f-re, a második a simaságot biztosítja és a harmadik a kontúr hosszának minimalizálásáról gondoskodik. Ezt a következőképpen alkalmaztuk optical flow számítására:

$$e = \alpha \int\int_\Omega (I_2(x,y) - I_{shynt}(x,y))^2 dxdy + \beta \int\int_{\Omega\setminus\Gamma} E_{OF}(x,y)dxdy + |\Gamma|, \tag{3.12}$$

ahol α és β súlyparaméterek, I_{shynt} egy olyan kép, melyet az első képszekvencia és az optical flow segítségével készítünk el a következőképpen $\forall i,j$:

$$I_{synt}(i + OF(i,j).x, j + OF(i,j).y) = I_1(i,j). \tag{3.13}$$

$$s_1 = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}, s_2 = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

3.7. ábra. A Sobel-gradiens operátorok

Legyen E_{OF} az optical flow gradiensvektora, ez a homogén területeken kis értékeket vesz fel viszont a határokon az értéke magas. Közelítsük a következőképpen:

$$E_{OF} = \sqrt{S_1(OF.x)^2 + S_2(OF.x)^2 + S_1(OF.y)^2 + S_2(OF.y)^2}, \tag{3.14}$$

ahol S_1 és S_2 a Sobel gradiens operátorok (3.14). Az egyenlet harmadik része $|\Gamma|$, ahol Γ a határpontok halmaza és $|\Gamma|$ ezen pontok száma, azaz a kontúr hossza.

A megoldást szimulált hűtéssel végeztük, eredményeket a dolgozat végén láthatunk, a módszerről bővebben a [15]-ben olvashatunk.

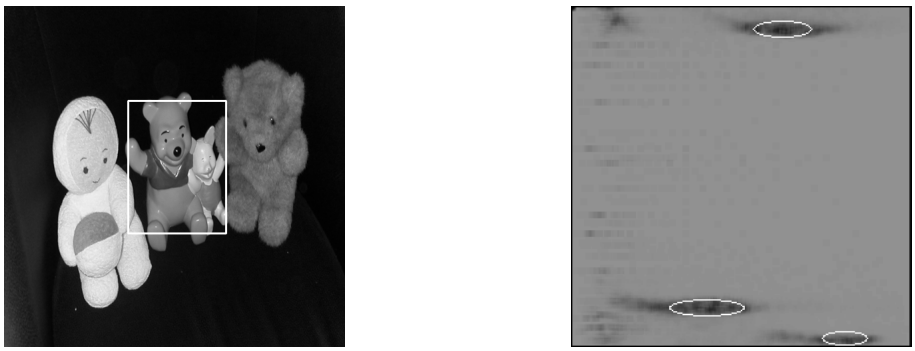
3.4. Objektumok követése statisztikai módszerrel

Y. Raja, J. McKenna és S. Gong publikálták 1998-ban az eljárást [27] mely segítségével színes objektumokat követhetünk és kereshetünk meg a képsorozaton. A módszer alapja az, hogy vannak bizonyos színterek, ahol a fényerő egy külön színkomponens, ilyenek például az LUV és HSV. Mindkettőben megtalálható a komponensek között a fényerő. Az objektumok követésénél ez a jellemző szükségtelen, sőt fontos, hogy különböző fényerő esetén is megtaláljuk a referenciaobjektumot. Így az említett komponenst elhagyva a maradék két jellemzőből egy kétdimenziós hisztogramot készíthetünk. A felismerés a következőképpen zajlik: a kiválasztott terület hisztogramjának eloszlását Gauss felületekkel közelítjük, majd megkeressük a képen azt a részt amelynek a referenciához leghasonlóbb a becslése. A (3.8). ábrán egy képet és a hisztogrammját láthatjuk.

A Gauss-görbék illesztését Expectation Maximalisation (továbbiakban EM) eljárás segítségével végeztük [18], [10]. Jelölje $\mathcal{O}$ a követendő objektumot és ξ egy pixelt, ha az EM m darab eloszlással közelít, akkor annak a valószínűsége, hogy a ξ pixel az $\mathcal{O}$-hoz tartozik:

$$p(\xi|\mathcal{O}) = \sum_{j=1}^{m} p(\xi|j)P(j), \tag{3.15}$$

ahol $P(j)$ a j. Gauss-eloszlásba tartozás valószínűsége és természetesen $\sum_{j=1}^{m} P(j) = 1$. Ha pedig már tudjuk, hogy a ξ pixel az objektum pixele, akkor annak a valószínűsége, hogy a ξ a j. komponensbe tartozik:



(a) (b)

3.8. ábra. Mesefigurák, és a kijelölt rész hisztogramja a megtalált Gauss-eloszlásokkal.

$$p(\xi|j) = \frac{1}{2\pi|\Sigma_j|^{\frac{1}{2}}} e^{-\frac{1}{2}(\xi-\mu_j)^T\Sigma_j^{-1}(\xi-\mu_j)}, \tag{3.16}$$

μ_j és Σ_j a j. eloszlás várható értéke és kovariancia mátrixa.

Az EM segítségével tetszőleges objektumra meghatározhatjuk $(\Sigma_1, \mu_1), ..., (\Sigma_m, \mu_m)$ értékeket, így csak meg kell keresni a referenciaobjektumhoz leghasonlóbbat és készen vagyunk, ám ezt a kép összes pixelére alkalmazni nagyon lassú lenne. Ennek megoldására a szerzők azt javasolják, ha ismert az objektum helye a t_{k-1} időpillanatban, akkor ebből a következőképpen határozzuk meg a t_k időpillanatban a képen való elhelyezkedését: tegyül fel, hogy az objektumot *befoglaló téglalap* (bounding box) középpontja $\mu^t = (\mu_x, \mu_y)$ és a nagysága $\sigma^t = (\sigma_x, \sigma_y)$. A szerzők azt javasolják, hogy az előző képkockán talált doboz $\frac{3}{2}$-szeresét tekintsük és itt keressük az objektumot. Ekkor a μ^{t-1} ismeretében a következőt mondhatjuk:

$$\mu^t = \mu^{t-1} + \frac{\sum_x p(\xi_x)(x - \mu^{t-1})}{\sum_x p(\xi_x)} \tag{3.17}$$

ebből pedig a doboz méretének kiszámítása:

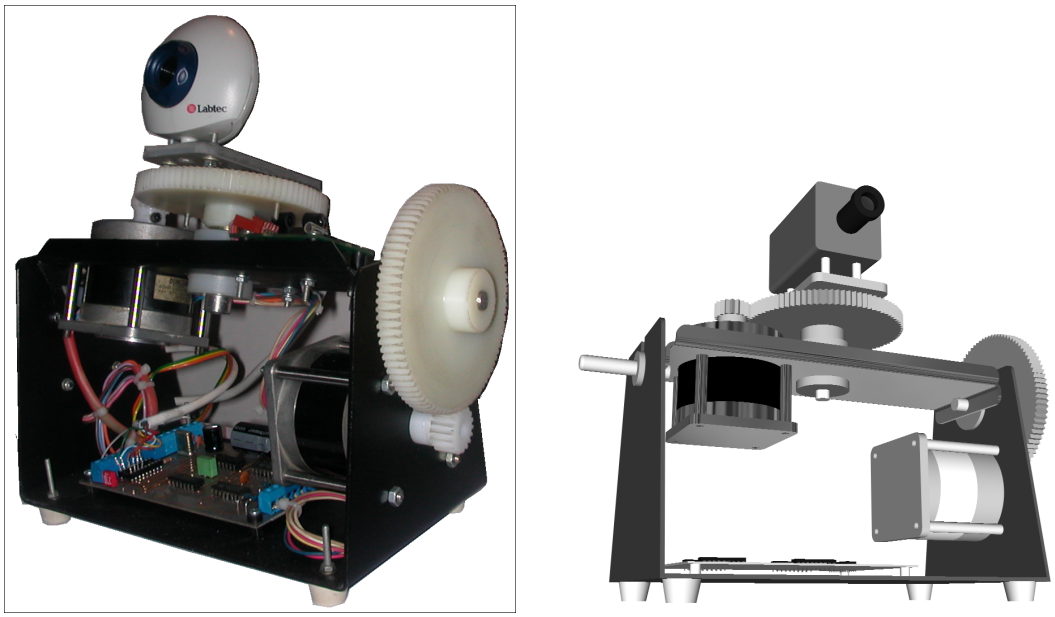
$$\sigma^t = \sqrt{\frac{\sum_x p(\xi_x)((x - \mu^{t-1}) - \mu^t)^2}{\sum_x p(\xi_x)}}. \tag{3.18}$$

A módszerrel elért kísérleti eredményeket a dolgozat végén láthatjuk a 3.6.5. fejezetben.

3.5. Mozgás követése robottal

Az optical flow-t sikerült real-time-ban implementálni, ám felmerült az igény, hogy nem csak detektálni, hanem követni is kellene a kamera környezetében található

mozgó objektumokat, itt azonban abba a problémába ütköztünk, hogy ezen objektumok hamar kiérnek a kamera látómezejéből. Kézenfekvőnek tűnt egy olyan hardver építése, mely mozgatja a kamerát és az egyszerűen vezérelhető egy számítógép segítségével. Megépítettük ezt a hardvert (továbbiakban robot[1]) mely alkalmas a tér majdnem teljes egészének szemmel tartására, és az itt mozgó objektumok nyomonkövetésére. Az elkészült szerkezet fényképe és AutoCAD-ben megtervezett és renderelt modellje a (3.9) ábrán látható.



(a) (b)

3.9. ábra. Az általunk épített robot (a) és annak AutoCAD-ben elkészített modellje (b).

A következő két alfejezetben pedig bemutatom a robot mechanikai és elektronikai felépítését, majd a teljes operatív szoftverrendszert mely a vezérlésért felel.

3.5.1. Mechanikai felépítés

A robot mozgatásáról két léptetőmotor gondoskodik, melyek 1.8°-os lépésközzel vezérelhetők[2], mivel jelen probléma nem igényel kifejezetten gyors mozgást, viszont a precíz pozícionálás és a kis lépésköz is fontos, fogaskerékhajtást készítettünk, mellyel tovább növeltük a lépésközt és így a mostani konstrukcióban egy teljes körbefordulás

[1] A robot szót valójában egy intelligens autonóm rendszerre használják, mely rendszerbe esetünkben a számítógép is beletartozna, de a dolgozatban robotnak csak a megépített hardvert és a vezérlő elektronikáját fogom nevezni.

[2] Egy speciális eljárás segítségével a lépésközt felére csökkentettük, így bár a sebesség csökkent a pontosság megduplázódott.

2000 lépésben végezhető. A motorokon lévő kis fogaskerekek 18 fogúak, míg a forgástengelyeken lévők 90 fogat kaptak.

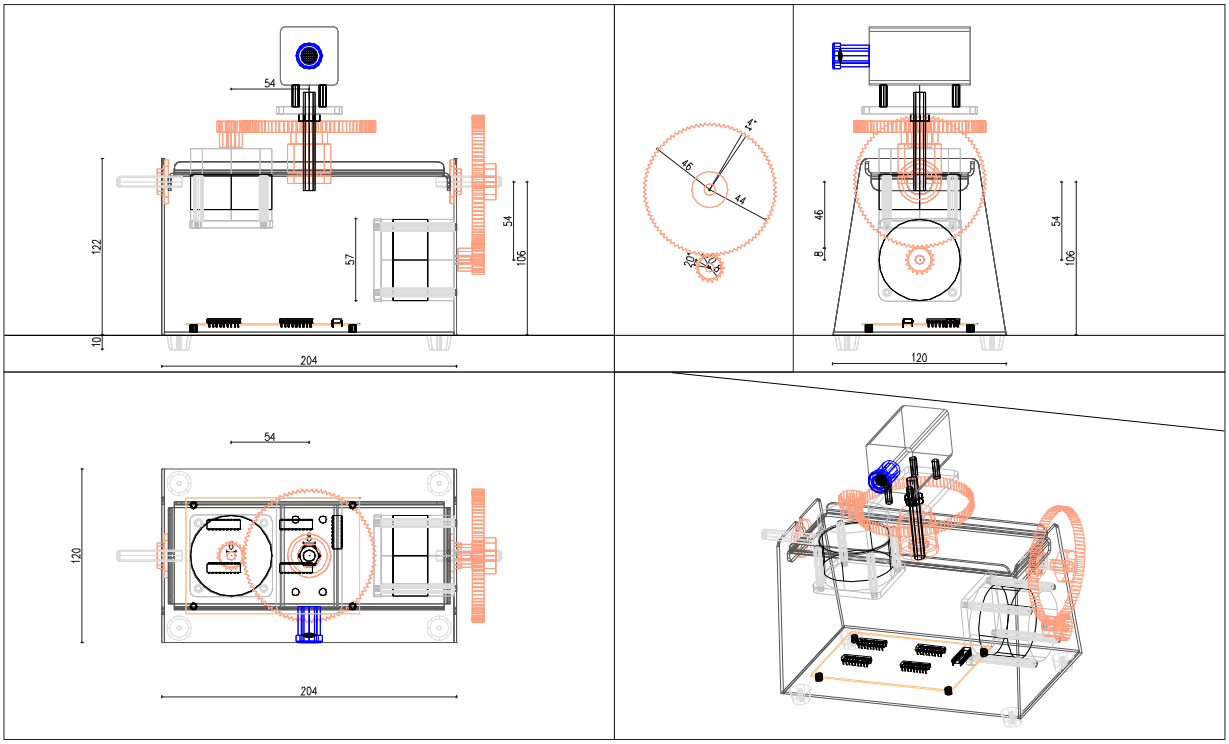
A készülék gumitalpakon nyugszik, mely elnyeli a léptetésből származó rezgésket, a teljes fémszerkezet saválló acélból készült. Az alaplemez 2 mm vastag "U" alakban meghajlítva, ez hordozza a vezérlő elektronikát tartalmazó panelt, az egyik motort és az ehhez a motorhoz tartozó végálláskapcsolót, ami egy infravörös fényű optikai kapu. Az alaplemez hordozza a szintén 2 mm-es saválló lemezből készült horizontális irányba mozgó (le-fel) felső lemezt. Ezen a lemezen található a második motor és az ehhez tartozó végálláskapcsoló. Az alaplemezen lévő motor a már említett 18-90 fogú fogaskerék-páron keresztül mozgatja a felső lemezt. A felső lemezen lévő motor szintén meghajt egy fogaskereket, mely a horizontális mozgást végzi és amelyre a kamera van szerelve.

Minden szükséges pontban csapágyat kapott a szerkezet, ezen csúszócsapágyak mindegyike danamidból készült, mely igen kicsiny tapadási-súrlódási együtthatóval rendelkezik és nagyon jó a kopásálló képessége. Az alaplemez és a felső lemez között a csapágy excenteres, mely segítségével a két fogaskerék holtjátéka minimálisra állítható. A fogaskerekek anyaga teramid, azért ezt választottuk, mert kicsiny a kopása és nagyon nagy a merevsége.

A teljes hardvert megterveztük AutoCAD-ben, a különböző nézetei a (3.10). ábrán láthatók.

3.5.2. Elektronikai felépítés

A rendszer vezérlő elektronikájának tervezését két dolog motiválta. Az első probléma, hogy meg kellett oldani a kommunikációt a számítógép és a robot között. Itt több szempontot is figyelembe kellett venni, első feladat a kommunikációs csatorna megválasztása volt, a párhuzamos portra esett a választásunk, mert nagyon egyszerű programozni, több szálat is vezérelhetünk egyszerre, bemenetet is könnyedén olvas és szabvány 5 Voltos logikai jelekkel dolgozik. Ugyanakkor tehermentesíteni szerettük volna a számítógépet a nagyon sok (másodpercenként esetenként több száz) portkezelési művelet alól, ezért beépítettünk egy saját mikroprocesszort a robotba, mely a kommunikációért, a motorok logikai vezérléséért és a végállások elérésének ellenőrzéséért felel, így a számítógép csak kiküld egy relatív vagy abszolút pozíciót és a mikrogép végrehajtja a szükséges utasításokat. A másik megoldandó probléma a motorok meghajtása volt, a felhasznált léptetőmotoroknak 24 Volt tápfeszültséget kell biztosítani, ekkor 300 mA az áramfelvételük, ám a mikrokontrollerből csak 5 Volt pár mA amit leszedhetünk, ráadásul nem is volt elegendő a kimeneti lábak száma, ezért kellett még egy multiplexert készítenünk, és ennek a kimenő jeleit küldtük az-

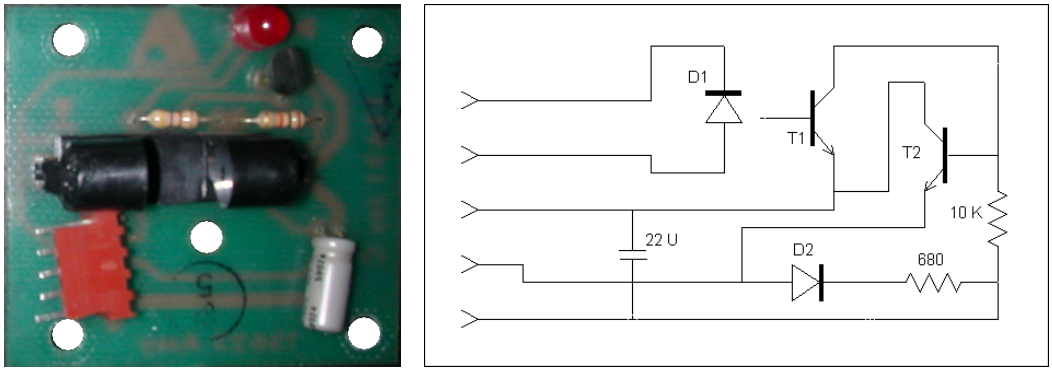


3.10. ábra. A robot műszaki rajza.

tán egy teljesítménytranzisztorokból álló egységnek mely meghajtja a motorokat. Az itt felsorolt problémák miatt döntöttem egy saját elektronikai vezérlőrendszer megvalósítása mellett mely megoldja a felmerülő gondokat. Nézzük meg most részletesen az elektronikai felépítést.

A (3.11). ábrán láthatjuk a végálláskapcsolóként használt optikai kapu és hordozójának fényképét és kapcsolási rajzát. A működése hasonló az görgős egér működéséhez, ha a fény útjába kerül valami akkor egy logikai magas jelet küld az áramkör. A D1 infravörös dióda az R1-en keresztül 5 Volt tápfeszültséget kap, fényét a T1 fototranzisztor érzékeli, ha pedig nincs meg az IR fény a T1 kapcsol és meghajtja a D2 LED-et és a kimenetet.

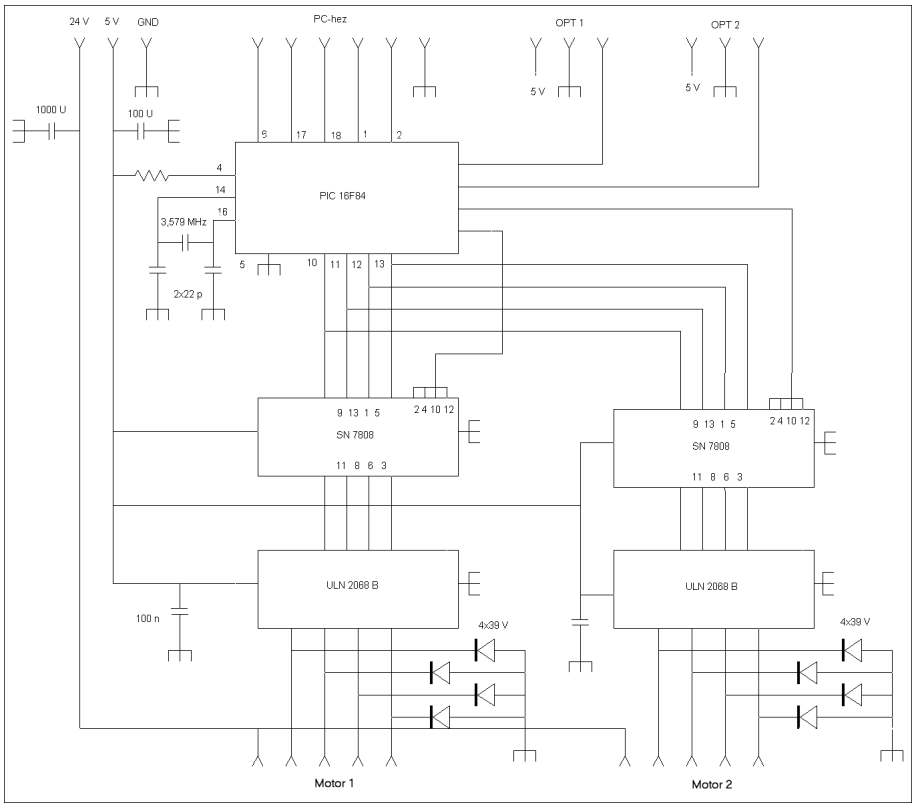
A robot fő elektronikai egységének kapcsolási rajza a (3.12). ábrán látható. A számítógép párhuzamos portjáról érkező logikai jeleket egy PIC 16F84 típusú mikrokontroller fogadja, ez a mikrokontroller egy EEPROM-mal rendelkező maximum 4 MHz órajelfrekvenciával meghajtható μgép, melynek 32 szavas utasításkészlete és 1 Kbájt programmemóriája van. A számítógép párhuzamos portjáról érkező jeleket az IC1 RA0,...,RA3 és RB0 lábai (17, 18, 1, 2 és 6 lábak) kezelik, az első négy adatokat az ötödik pedig órajeleket kezel. Az RB4,...,RB7 és RB2, RB3 lábak a motorok vezérlését látják el, az RB2 illetvel RB3 segítségével kiválasztjuk az első vagy



(a) (b)

3.11. ábra. Az optikai kapu fényképe (a) és kapcsolási rajza (b).

a második motort majd az RB4,...,RB7 kimenetekkel meghajtjuk a tekercseket. Az RA4 és RB1 a két végálláskacsoló adatait fogadja. A μgép működéséhez szükséges néhány passzív alaktrész, egy 3,579 MHz-es kvarcoszcillátort használunk az órajel előállítására ezenkívül szükség volt még néhány kondenzátor beépítésére, hogy a tápfeszültségben keletkező ingadozásokat kiszűrjük.



3.12. ábra. A vezérlőegység kapcsolási rajza.

A mikorkontrollerből két SN 7808 típusú logikai IC-re küldjük az adatokat, ezek az IC-k darabonként 4-4 AND kaput tartalmaznak, így a kiválasztó és az adat vonal ÉS kapcsolatából multiplexelve alakítottunk ki a motorok meghajtását. Az 7808-as

IC-ből a jeleket két ULN 2068B típusú darabonként négy teljesítménytranzisztort tartalmazó IC fogadja, melynek kimenetei már a motor tekercseire vannak kapcsolva. A motorok 4 tekerccsel rendelkeznek, mely terkecseknek a (3.19) mátrixban látható sorrendben biztosítottunk az áramot.

$$\begin{array}{rcccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1.tekercs & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 2.tekercs & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 3.tekercs & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 4.tekercs & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{array} \tag{3.19}$$

A rendszer áramellátásáról egy ipari kivitelű kapcsolóüzemű tápegység gondoskodik, ennek a tápegységnek négy egymástól független kimenete van, kétszer 5 V és kétszer 12 V. 5 V-on a tápegység 10 A-rel terhelhető, és 12 V-on 1,5 A-rel. A logikai áramköröknek 5 V kellett míg a motoroknak 24 V – a két 12 V-os egység sorba kapcsolva – tápfeszültségre volt szükségük.

3.5.3. A robot vezérlő programja

A vezérlés párhuzamos porton keresztül történik, a programot Windowos felületre fejlesztettem, ám egy problémába ütköztem, a Win 2K és a fejlettebb verziók nem támogatják a direkt portvezérlést, mig a Win95 és Win98 illetve a DOS-os felületeken a párhuzamos portot egyszerűen elérhetjük. A probléma megoldására két megoldást is találtam, ez egyik a DiskDude "dlportio.dll" könyvtára, melyhez egy package-et is készítettek ami egyszrűnen telepíthető Borland C++ Builder felületre, ám Visual Studio alatt sikeresebben probálkoztam a Logix4U által fejlesztett "inpout32.dll" könyvtárral.

A másik megoldandó probléma a kamera képéhez való hozzáférés volt, itt is több lehető ség adott, a legelegánsabb ám leggépigényesebb a Microsoft által fejlesztett DirectX, mely a DirectX Software Developement Kit letöltése után egyszerűen használható és rengeteg példaprogramot tartalmaz. A másik általam is alakalmazott módszer a standard headerek közütti "vfw.h"[3], segítségével szintén egyszerűen elérhető a médiaforrás.

A rendszer Unixos platform alatti fejlesztése még nem indult el, de a közeljövőben ezt tekintem az egyik megoldandó problémának több okból is, mivel az algoritmusok meglehetősen gépigényesek nagyobb felbontás esetén négyzetesen nő a számításigény, mely esetleg többprocesszoros gépek használatát igényelheti, ennek megvalósítására UNIX alapú redszerek alkalmasabbak.

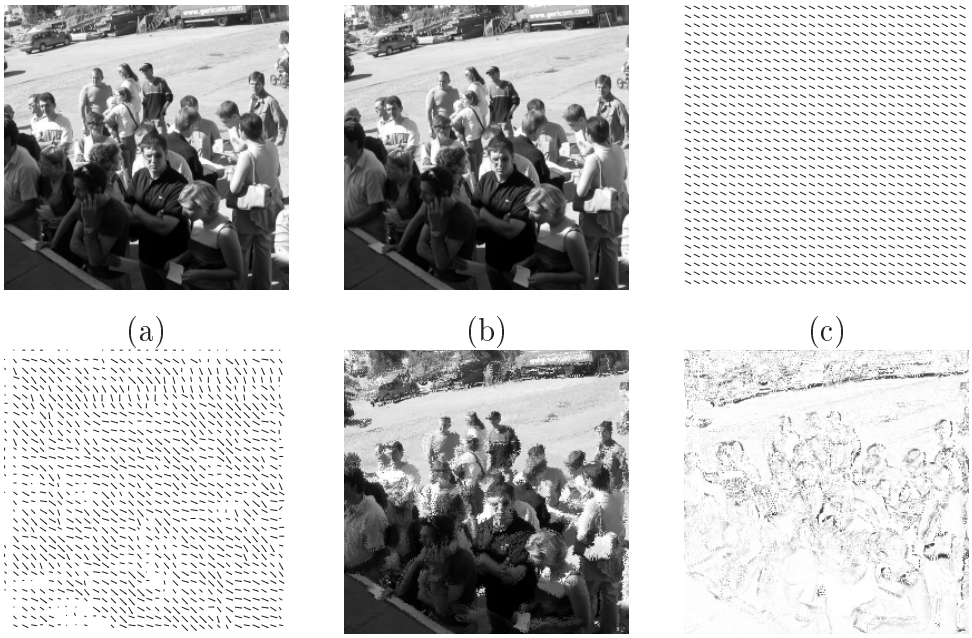
[3] Video for Windows header

Most pedig egy rövid áttekintést adok a robot vezérlő programjáról, melynek teljes forrását és a lefordított fájlokat a CD-mellékleten találhatjuk. A program indításakor inicializálnunk kell a portvezérlést, és el kell készítenünk az ablakon azt a részt amelyre a kamera képét fogjuk küldeni. Ezekután szükségünk lesz két újabb programszál (thread) indítására, melyek közül az első a kameráról érkező jeleket lelopja a második pedig a motorokat mozgatja ha szükséges. A kameráról érkező jeleket egy vektorban kapjuk meg mely nagyon egyszerű struktúrájú, a pixelek színei sorfolytonosan fel vannak benne sorolva, RGB komponensekkel, így minden pixel 3 bájtnyi helyet foglal. A vezérlésre két programot is készítettem, az első egyszerűbb és gyorsabb csak veszi a két utolsó képkocka különbségét, csinál egy szűrést majd megkeresi a mozgó objektumok által megváltoztatott pixelek középpontját, ha ez a középpont egy meghatározott értéknél távolabb esik a kép közepétől akkor elküldjük a motoroknak a pozíciót ahová állniuk kell, megvárjuk, hogy a motorok beálljanak a kívánt helyre majd ha szükséges ismételt mozgatás következik. A másik program a meghatározott optical flow segítségével mozgatja a kamerát, az alapelv hasonló mint az előbb csak itt nem különbséget hanem optical flowt számolunk, majd a megkapott mozgás irányába mozgatjuk a kamerát.

3.6. Kísérleti Eredmények

Ebben a fejezetben bemutatom az eddig ismertetett módszerek alkalmazásával született kísérleti eredményeimet. Először a dolgozat elején ismertetett gyors optical flow meghatározó algoritmus működésére nézünk példákat, megvizsgáljuk geometriai transzformációkra (forgatás, nyújtás eltolás), majd szintetikusan előállított képsorozatra, végül pedig standard tesztsorozatokra. Az energiaminimalizációs eljárás működését szintén standard képsorozatokon teszteljük, majd az expectation maximisation segítségével kinyert jellemzők és a jellemzőkkel leírt objektumok megtalálására mutatok be példákat.

Végül a dolgozat talán leglátványosabb része következik, melyben a robotkamera müködésére láthatunk példákat, először egy egyszerű mozgáskövető rendszer outputját, majd real-time optical flow-t, végül az objektumokat (személyek) követő kamera mozgásából egy képsorozat.



(a) (b) (c)

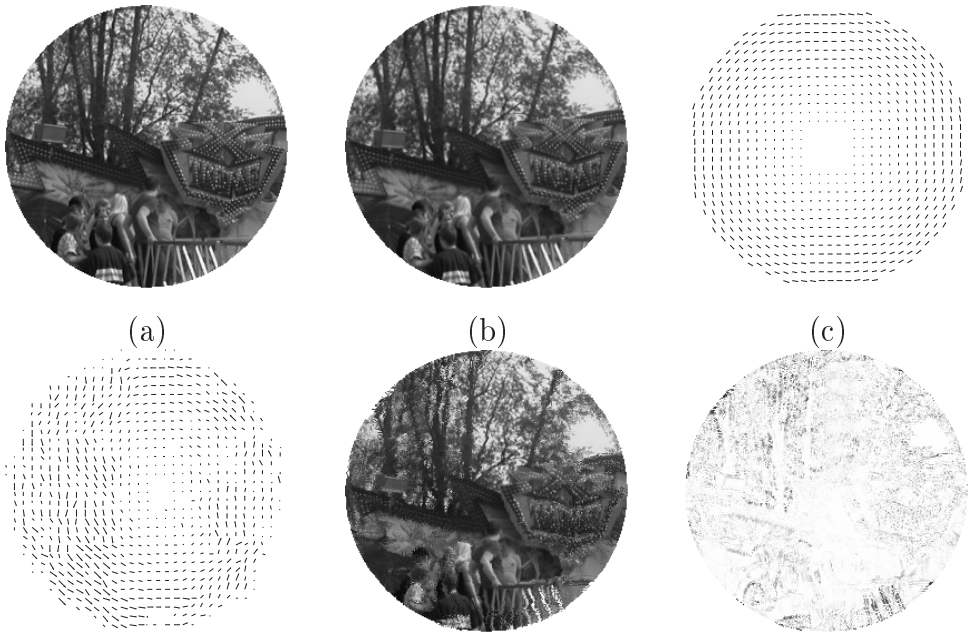
(d) (e) (f)

3.13. ábra. Példa az eltolás műveletre, az (a) és (b) ábrákon az eredeti képek láthatók, a második $(6, -3)$ vektorral lett eltolva, a (c) az eredeti eltolási mezőt a (d) a számított optical flowt az (e) a rekonstruált képet míg az (f) az (b) és (e) különbségét ábrázolja.

3.6.1. Geometriai transzformációk vizsgálata

Eltolás

A (3.13). ábrán az eltolásra láthatunk példát, az első és második kép a két képszekvenciát ábrázolja, melyeken egy vásáron érdeklődő emberek láthatók, a második szekvencia $(6, -3)$ vektorral lett az elsőhöz képest eltolva. A harmadik ábra a korrekt eltolási mezőt ábrázolja, míg a negyedik a 3.2. fejezetben bemutatott algoritmussal számított optical flow-t. Az (e) ábrán láthatjuk a már tárgyalt rekonstruált képet, és az utolsó képen a második képszekvencia és a rekonstruált kép különbségét láthatjuk. Az összes tesztelt eset közül ezen a sorozaton volt a legnagyobb a rekonstrukciós hiba, mely $0,127376$ érték lett. A magasabb hiba a képen található meglehetősen sok homogén területnek köszönhető, figyeljük meg, hogy azokon a területeken ahol az emberek állak és árnyaltabb a kép, a vektorok magasabb arányban korrektek. Összességében mégsem mondható rossznak a számított optical flow, minden vektor a megfelelő irányba mutat nincsenek nagyon rosszul detektált régiók.



(a) (b) (c)

(d) (e) (f)

3.14. ábra. Forgatás, a második képkocka az elsőhöz képest óra járásával ellentétes irányban 3°-kal lett elforgatva.

Forgatás

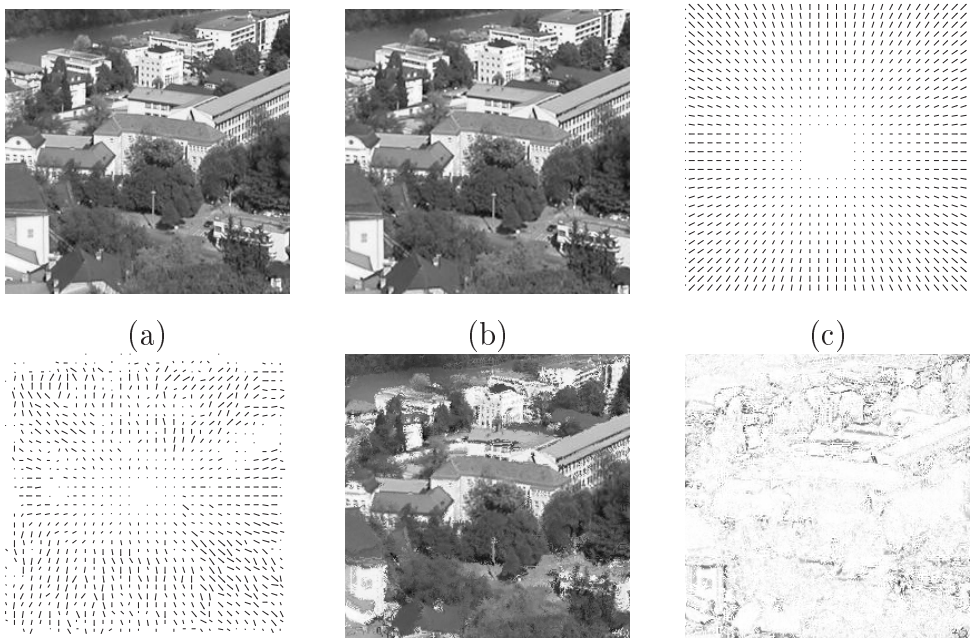
A (3.14). ábrán láthatunk a forgatásra példát, a második ábra órajárással ellentétes irányban 3°-kal el lett forgatva, a képen egy vidámpark körhintáját láthatjuk. A rekonstrukciós hiba értéke $0,10202$ lett, mely a geometriai transzformációk közül a legkisebb, mely betudható a nagyon kevés homogén mezőnek. Az optical flow-t megvizsgálva a vektorok között itt sem találunk rossz irányba mutatót.

Átméretezés

A (3.15). ábrán átméretezésre láthatunk példát, a kép a Salzburgi-vár kilátójában készült, a második ábra $105,5\%$-kal fel lett nagyítva. A rekonstrukciós hiba $0,1107$ lett. A számított optical flow-t megvizsgálva láthatjuk, hogy általában korrektek a vektorok, de a homogén területeken itt is vannak problémák (háztetők, Duna-felszíne).

3.6.2. Mozgás detektálása szintetikusan előállított képsorozatokon

Szintetikusan előállított képre vizsgáljuk az algoritmust, a képen egy bika és egy mérföldeket jelző tábla található konstans háttérrel, a bika néhány pixelt lefelé és balra mozog, a tábla lefelé és jobbra. A rekonstrukciós hiba 0.057175, amely igen



(a) (b) (c)

(d) (e) (f)

3.15. ábra. Példa átméretezésre, a második kép 14 pixelnyivel, azaz 105, 5%-kal lett felnagyítva.

jónak mondható (3.16). Ebben az esetben és a továbbiakban sem tudjuk megmondani a korrekt optical flow-t, csak a becslést.

3.6.3. Természetes mozgások detektálása

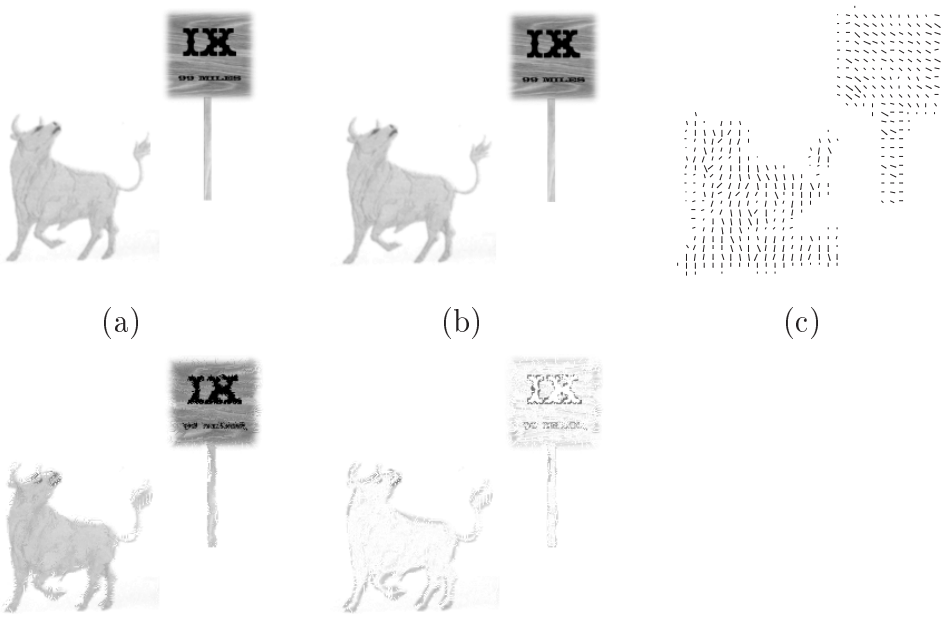
Az algoritmust két standard képsorozaton teszteltük. Az első egy utcai képet ábrázol, fix kamerával lettek a szekvenciák felvéve, melyeken néhány személyautó és egy busz mozog. Láthatjuk, hogy a busz tetején nem sikerült jól detektálni a mozgásvektorokat, ez a homogén felszínnek köszönhető. A felvétel igen zajos, így a $0,04906$ érték a rekonstrukciós hibára nem tekinthető rossznak (3.17).

A másik standard képsorozat melyre az algoritmust teszteltem az asztaliteniszező játékost ábrázolja (3.18). A képsozozat nagyon jó minőségű, szinte zajmentes, így az eredmény igen jó. A rekonstrukciós hiba $0,03423$ lett.

3.6.4. Eredmények az energiaminimalizációs eljárás segítségével

A 3.3. fejezetben bemutatott módszer kísérleti eredményeit láthatjuk a (3.19, 3.20 és 3.21) ábrákon, mindhárom tesztsorozat szimulált hűtéssel lett optimalizálva. A szimulált hűtés futásideje $100 - 1000$ másodperc között mozog.

A (3.19). ábrán a mozgásdetektáló algoritmusok tesztelésére szolgáló tesztsoro-



(a) (b) (c)

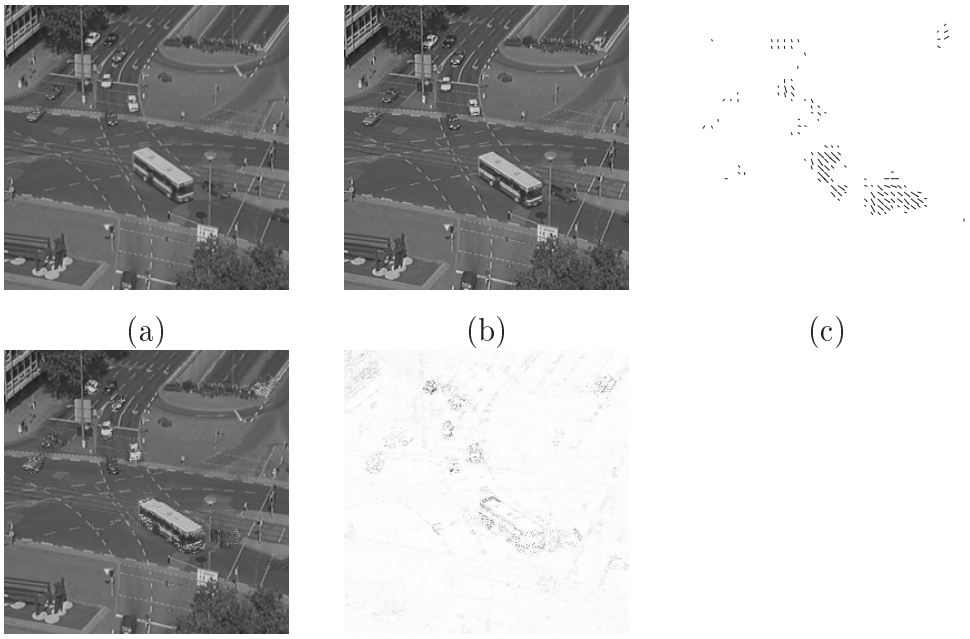
(d) (e)

3.16. ábra. Példa szintetikusan előállított képre.

zatok közül a legkedveltebb a Hamburgi taxi látható melyet a Hamburgi egyetem készített. A képen egy utca látható néhány autóval és egy gyalogossal. A járművek közül három mozog, az egyik egy taxi a másik egy kis Volkswagen mely nagyon sötét színű így alig tér el a háttértől, a harmadik pedig egy kisteherautó, mely pont ágak mögött van, így nehézségekbe ütközik a mozgásának megállapítása. Összességében a képsorozat nem tekinthető könnyű sorozatnak mozgásdetektálás szempontjából. A (b) ábrán a becsült optical flow-t láthatjuk, a (c) ábrán pedig a k-means algoritmus által osztályokba sorolt vektorokat, míg az utolsó képen a szimulált hűtés segítségével megoldott optimalizációs feladat megoldását láthatjuk. Ez esetben a vektorok detektálása már megfelelő, bár a kisteherautó alakját így sem sikerült teljesen megtalálni.

Második tesztsorozat (3.20) a már többször bemutatott asztalitenniszező példa, az eredetileg is kielégítő minőségű optical flow becslésből nagyon jó eredmény született.

Végül be szeretnék mutatni egy olyan tesztesetet amelyre kimondottan nem javasolt a mozgások osztályokba sorolása, ez a geometriai transzformációk közül a forgatás (3.21), a forgó mozgás minden pontban más iránnyal és nagysággal rendelkezik, ezért nem tudjuk korrekten régiókba sorolni a mozgásokat. A (b) árbrán a becslés, a (c)-n az optimalizálás eredménye, míg a (d)-n a korrekt megoldás látható.



(a) (b) (c)

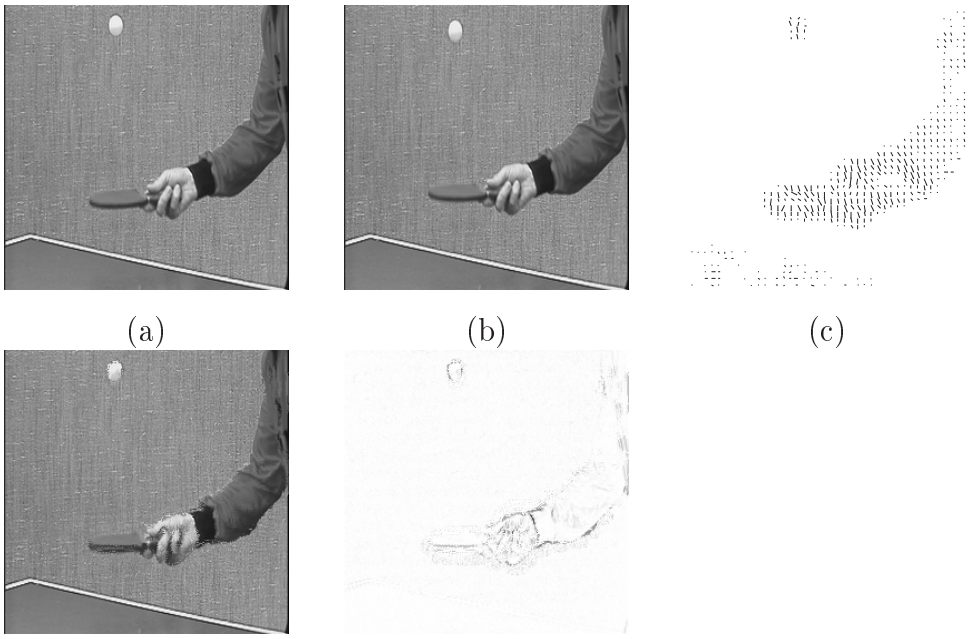
(d) (e)

3.17. ábra. Példa egy közlekedési képsorozatra.

3.6.5. Példák statisztikus módszerrel történő mozgáskövetésre

Most pedig a már korábban bemutatott EM-segítségével működő módszer eredményeit mutatom be. A (3.22). ábrán láthatjuk a képsorozatot, fehér célkeresz mutatja a módszer által talált objektumot, a jelenlegi implementáció még fix méretű *bounding box*-ot használ. A teszt során egy színes dobozt követtünk, megpróbáltuk azt az esetet, hogy változtatjuk a mozgás irányát és nagyságát, és az egyik személy átadja a másik kezébe a tárgyat, de azt nem veszítettük el, végül pedig ellentétes irányú gyors mozgást végeztünk, de ebben az esetben is stabilnak bizonyult a módszer. Ennél a tesztesetnél 4 Gauss-görbét illesztettünk a hisztogrammra, és a keresési ablak a bounding box kétszerese volt. Tapasztalataim szerint színes objektumok esetében $3-5$ Gauss-görbét érdemes használni, több görbe esetén sem sikerült jelentő javulást elérni a követésben, ha pedig emberi arcokat követünk az arc karakterisztikájából kifolyólag elegendő 1 vagy 2 görbét használnunk.

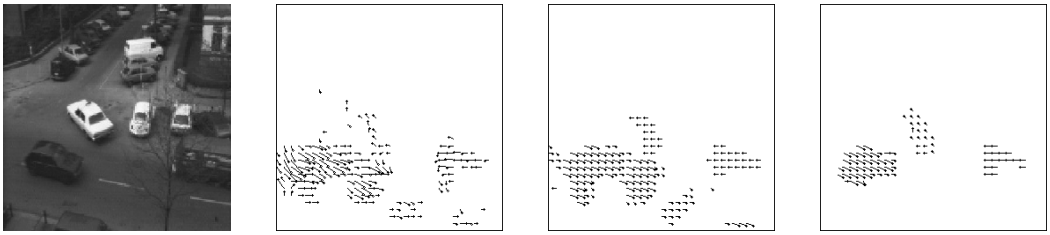
A második teszt során (3.23) mozgó kamerát használtam, a kamerának az arcomat kellett követnie, és ezt sikeresen meg is tette, ebben az esetben 2 Gauss-eloszlást használtam. Itt nagyon jól érzékelhető az, hogy a módszer nem érzékeny a fény erősségére, más-más képkockákon egészen eltérők a fényviszonyok. A sorozat minden 6. képkockáját rögzítettem. A követés a jelenlegi implementáció melett 15-20 frame/sec sebességre képes a bounding box méretétől függően.



(a) (b) (c)

(d) (e)

3.18. ábra. Az asztaliteniszező játékos és a számított optical flow.



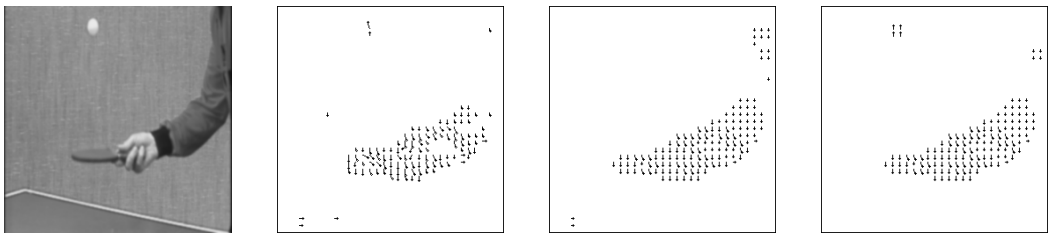
(a) (b) (c) (d)

3.19. ábra. Hamburgi taxi sorozat.

3.6.6. Mozgás követése robotkamera segítségével

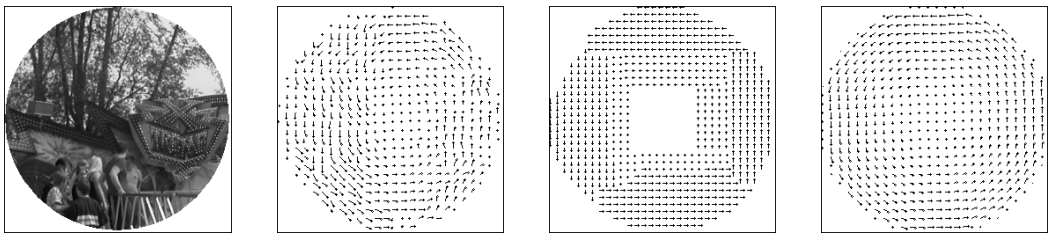
Ebben a fejezetben három különböző tesztesetet vizsgálok, az első egy real-time mozgáskövető rendszer, mely a mozgó objektumra egy célkeresztet illeszt, a második ugyanez, az algoritmus de mozgó kamerával, a harmadik real-time optical flow detektálást mutat be. A tesztek egy Pentium®4 1,7 GHz-es 256 MB memóriával rendelkező számítógép segítségével készültek.

A (3.24). ábrán egy real-time mozgásdetektáló algoritmus kimenetét láthatjuk, az implemetált algoritmus átlagosan 20-25 frame/sec sebességgel képes meghatározni a képen látható mozgásokat, az ábrán minden 3. képkockát rögzítettem, igy egy kb. 2 másodperces részletet láthatunk. A célkereszt mutatja a mozgások középpontját. A használt kamera felbontása 320×240 pixel, melyet maximum 25 frame/sec sebességgel képes rögzíteni, sajnos a képe meglehetősen zajos, és nagyon fényérzékeny, a vezérlője pedig beépített fénykiegyenlítést is tartalmaz, amely bezavart az optical



(a) (b) (c) (d)

3.20. ábra. Asztaliteniszező játékos.



(a) (b) (c) (d)

3.21. ábra. Forgatás.

flow detektálás során, de ezt sikerült kiküszöbölni.

A (3.25). ábrán a robot segítségével követett emberek láthatók, a képsorozat minden 10. kockája lett rögzítve, így a sorozat kb 10 másodperc hosszúságú, láthatjuk, hogy az első képkockától kezdve a robot állandóan változtatja a kamera irányát a képsorozat végéig a mozgó személyek mozgásának irányába.

A (3.26). ábrán a szerzőt láthatjuk minden 3. képkocka lett rögzítve, a robot akkor sem vesztett el amikor takarásban voltam

A (3.27) és (3.28) ábrákon láthatjuk a real-time optical flow algoritmusunk kimenetét, minden 5. képkocka került rögzítésre, a jelenlegi implementáció mellett $8-10$ frame

sec sebességre képes, bár véleményem szerint ez még tovább gyorsítható. Az első képen a mozgásvektorok a kéz mozgásának irányát jól követik, de a homogén területeken sajnos itt sincsenek meg a korrekt vekotrok.

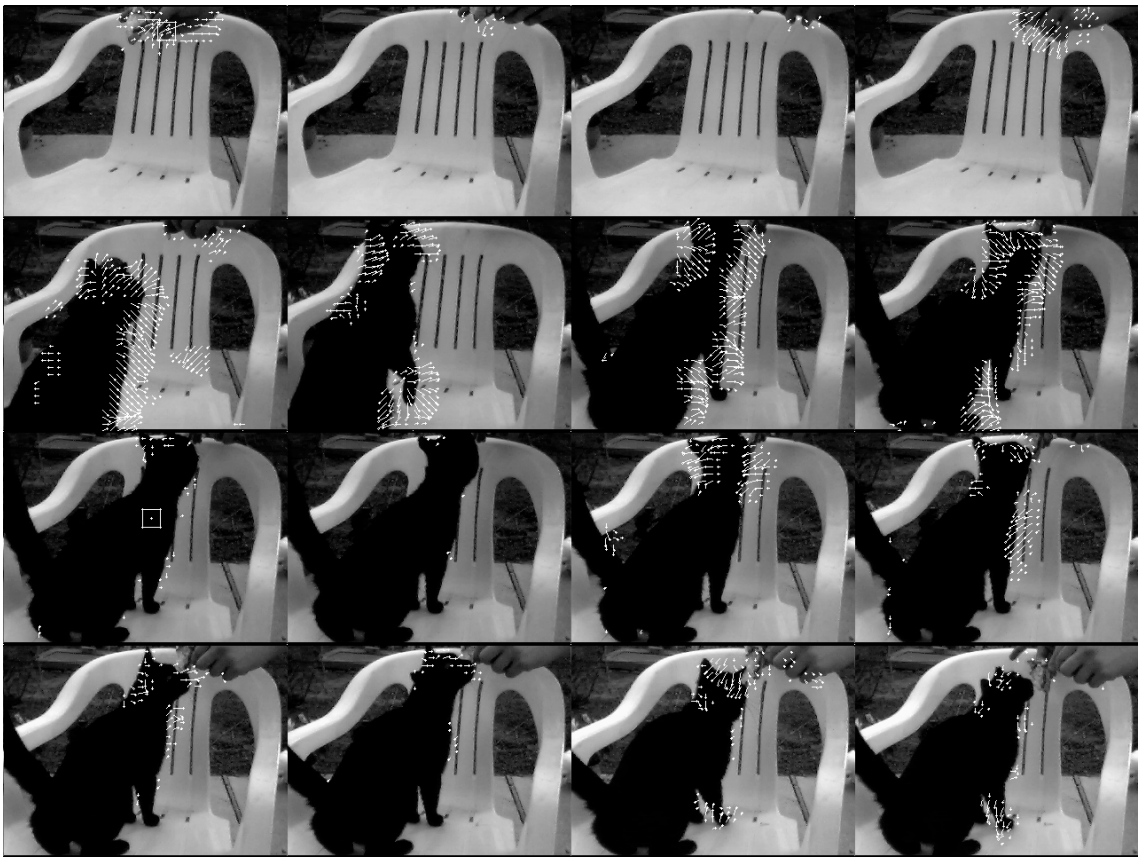
3.22. ábra. Expectation maximization segítségével készült tesztsorozat.

3.23. ábra. Mozgó kamerával végzett objektumkövetés EM segítségével.

3.24. ábra. Mozgás követése robot nélkül, a célkereszt a a mozgó objektum középpontját mutatja.

3.25. ábra. Mozgás követése a robot segítégével.

3.26. ábra. Sétáló alak követése robotkamera segítségével.



3.27. ábra. Példa optical flow real-time detektálására.

3.28. ábra. Optical flow valós idejű meghatározása.

4. fejezet

Programok a CD-mellékleten

A dolgozat CD-mellékletén megtalálható az összes felhasznált programcsomag, az általam írt programok, a project készítése közben készült fényképek és videók, a tervrajzok és a teljes dokumentáció. Most részletesebben ismertetem a CD-melléklet tartalmát.

4.1. Aktív kontúr

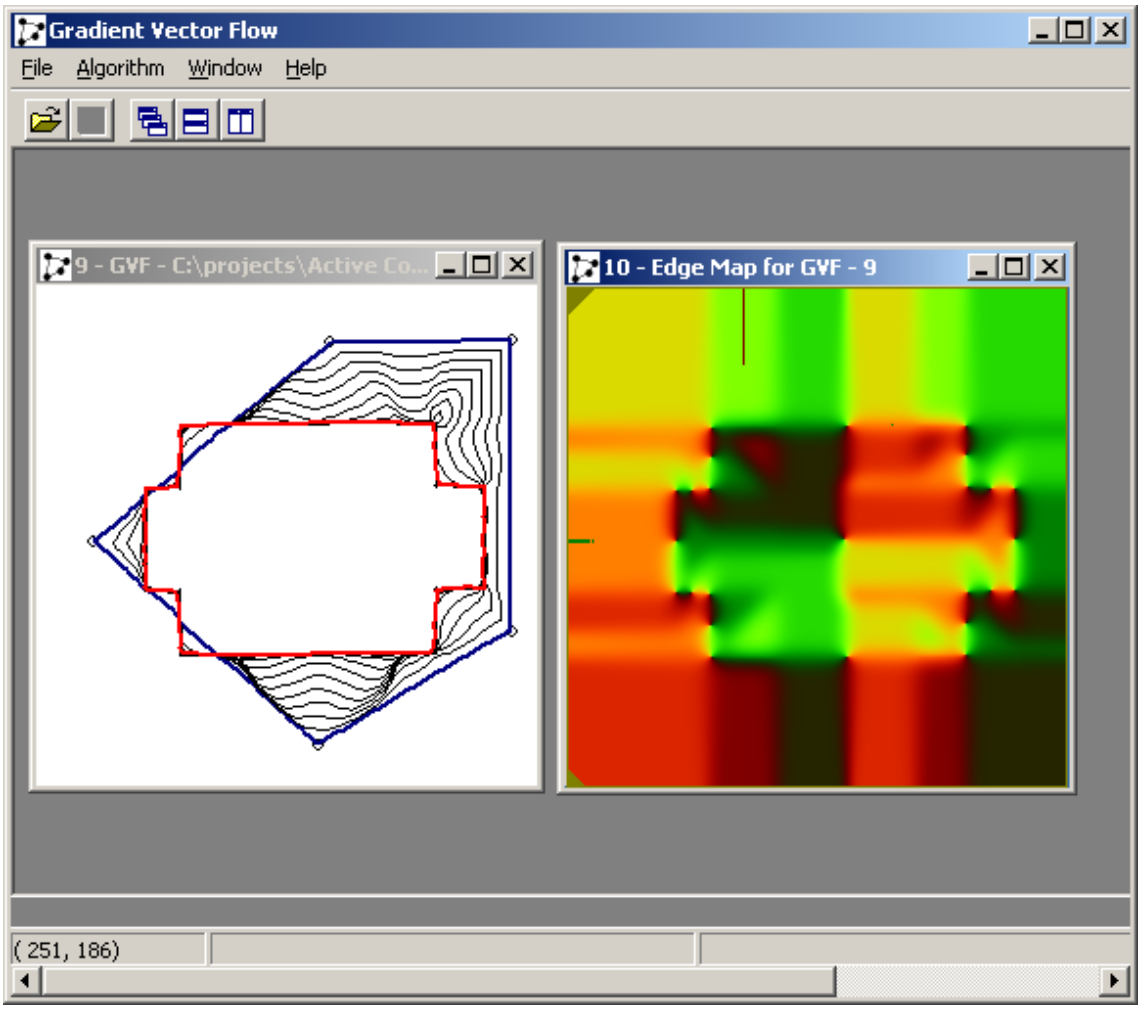
A mellékelt CD-ROM */ac/egvf/* könyvtárában találjuk meg az *Extended Gradient Vector Flow* nevű programot. A program különböző aktív kontúrokkal történő szegmentálásban nyújthat segítséget. Kezelése nagyon egyszerű, a program egy képét láthatjuk a (4.1) ábrán. Az algoritmus paramétereinek beállítását egy ablakban végezhetjük, melyről képeket a (4.2) ábrán találunk. Az első panelen a módszert választhatjuk meg (a), a másodikra akkor van szükségünk ha több kép alapján szeretnénk a szegmentálást végzni, a harmadik a kontúr inicializálásának beállításait tartalmazza (b), míg az utolsó a kimeneti specifikációkat.

A */ac/images/* könyvtárban találunk tesztképeket és videókat a program működéséről.

4.2. Mozgás detektálás

4.2.1. A */motion/doks/* könyvtár

Ebben a könyvtárban található a dolgozat egy oldalas kivonata, mely TEX-ben készült, a */dolgozat/* könyvtár tartalmazza jelen dokumentumot, a benne található képeket *.eps* formátumban, és az optical flow teszteket külön könyvtárban *.pgm* formátumban.



4.1. ábra. Az Extended Gradient Vector Flow program.

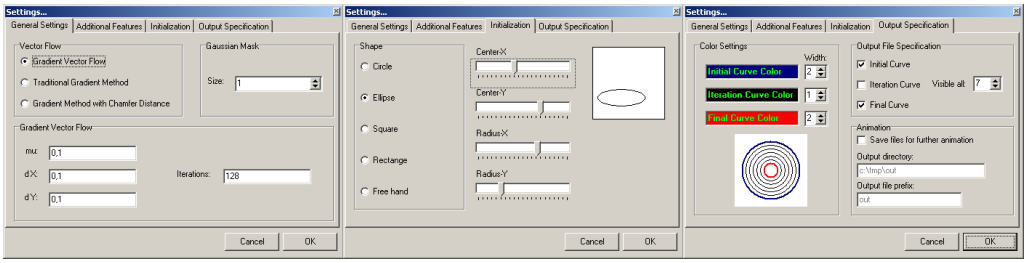
4.2.2. A */motion/images/* könyvtár

A könyvtárban találjuk az összes fotót mely a munka közben készült, a */autocad/* könyvtár tartálmazza a robot AutoCAD-ben készült terveit, a */captured/* alkönyvtárban találhatjuk, a 4×4-es mátrixba rendezett tesztsorozatokat. A */robotphotos/* sok képet és videót tartalmaz a robotról.

4.2.3. A */motion/install/* könyvtár

Minden olyan program installját tartalmazza, melyek az algoritmusok reprodukálásához szükségesek lehetnek és letöltésük nehézséget okozhat. Ezek:

- *C++ Templated Image Processing Library* : az INRIA-n fejlesztik, egy template-ekkel működő képfeldolgozási könyvtárgyűjtemény.
- *Color Conversion Algorithms*: Egy *.html* dokumentum színtérkonverziókról.



(a) (b) (c)

4.2. ábra. Az program beállításai.

- *TDLPortIO version 1.2*: Portvezérlő csomag Borland C++ Builderhez.
- *Microsoft DirectX 9.0 Software Developement Kit*: DirextX-es felület szerkesztéséhez, megtaláljuk a tejes dokumentációt is.
- *Intel Open Source Computer Vision Library*: A Intel processzorokra optimalizált képfeldolgozó könyvtár, nyílt kóddal.
- *WebCap*: A *vfw.h* felhasználásával működő kamera capture-elő programocska.

4.2.4. A */motion/prog/* könyvtár

Ebben a könyvtárban megtaláljuk a dolgozatban ismertetett módszerkhez készült programokat, ezek:

- *Color Code Tracking*: Egy érdekes megjelenítése a mozgásoknak, egy webkamera szükséges hozzá, a mozgások különböző irányát más-más színezéssel mutatja meg.
- *Expectation Maximization for Color Images*: Egyszerű program egy képet megnyit elkészíti a 2D hisztogrammját majd egy EM algoritmust végez és vizualizálja azt. Futtatásához szükségesek a Borland VCL komponensei!
- *Optical Flow Detecting*: Egy webkamera segítségével real-time optical flow detektálást végez, ha rendelkezésünkre áll egy robot is, akkor azt is tudja vezérelni.
- *Tracking001*: Egyszerű mozgásdetektáló és robotvezérlő programocska, egy webkamerával robot nélkül is használható.
- *EM Tracking*: Expectation Maximization segítségével a kijelölt objektum követésére használható program.

Irodalomjegyzék

[1] P. Anandan: A Comutational Framework and an Algorithm for the Measurement of Visual Motion, *Int. J. Computer Vision*, Vol. 2, No. 3, pp. 283-310

[2] P. Bouthemy and E. Francois: Motion Segmentation and Qualitative Dynamic Scene Analysis from an Image Sequences, *International Journal of Computer Vision*, 10:2, 157-182, 1993.

[3] H. Borgeforst: Hierarchial Chamfer Matching: A Parametric Edge Matching Algorithm, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 10. No. 6., November 1988.

[4] P. J. Burt and E. H. Adelson: The Laplacian Pyramid as a Compact Image Code, *IEEE Transactions on Communications*, Vol. COM-31, no. 4, april 1983 pp. 532-540

[5] L. D. Cohen and I. Cohen: Finite-Element Methods for Active Contour Modells and Balloons for 2-D and 3-D Images, IEEE Transactions on Pattern Anal. Machine Intell. vol. 15pp. 1131-1147. Nov. 1993.

[6] R. Courant and D. Hilbert: Methods of Mathematical Physics, vol. 1. *New York: Interscience*, 1953.

[7] D. Cremers: Statistical Shape Knowledge in Variational Image Segmentation, *PhD Thesis* 2002, Mannheim.

[8] D. Cremers, C. Schnörr: Statistical shape knowledge in variational motion segmentation, *Image and Vision Computing 21*, (2003) pp. 77-86.

[9] R. Cucchiara, M. Piccardi, A. Prati and N. Scarabottolo: Real-time Detection of Moving Vehicles, *IEEE Transactions on Image Processing*, 1999 pp. 618-623, 1999.

[10] A. Dempster, N. Laird and D. Rubin: Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B* , 38, 1977.

[11] D. J. Fleet és A. D Jepson: Computation of Component Image Velocity from Local Phase Information. *IJCV*, 5(1): 77-104, 1990.

[12] F. Gibou, R. Fedkiw: A Fast Hybrid k-Means Level Set Algorithm for Segmentation

[13] F. Heitz and P. Bouthemy: Multimodal Estimation of Discontinuous Optical Flow Using Markov Random Fileds, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 15. No. 12., December 1993.

[14] B. K. P. Horn and B. G. Schunk: Determining Optical Flow. *Artificial Intelligence,* 17:185-204, 1981.

[15] Horváth P. és Kató Z.: Optical Flow Meghatározása Energiaminmalizációs Módszerrel, *KEPAF IV. Miskolc*, pp 125-130, 2004 január.

[16] Horváth P. és Kató Z.: Szín, Textúra és Mozgás alapú Szegmentálás Gradient Vector Flow segítségével, *KEPAF IV. Miskolc*, pp 131-136, 2004 január.

[17] M. Kass, A. Witkin and D. Terzopoulos: Snakes: Active Contour Modells, *Int. J. Comput. Vis.*, vol1., pp. 321-331, 1987

[18] Kocsor A.: Gépi tanulási módszerek statisztikus megközelítésben. *Előadásjegyzet*, SZTE TTK 2003.

[19] B. Leroy, I. Herlin and L. D. Cohen: Multi-Resolution Algorithms for Active Contour Modells, *12th Int. Conf. Analysis and Optimization of Systems*, pp. 58-65. 1996

[20] T. Lin and J. L. Barron: Image Reconstruction Error for Optical Flow

[21] B. D. Lucas and T. Kanade: An Iterative Image Registration Technique with an Application to Stereo Vision, *Proc. 7th Int. Joint Conf. Articcicial Intelligence.*, Vancouver, Canada, pp. 674-679, 1981.

[22] É. Mémim and P. Pérez: Joint Estimation-Segmentation of Optical Flow, *ECCV'98*, Freiburg, Germany, 1998.

[23] D. Mumford and J. Shah: Optimal Approximations by Piecewise Smooth Functions and Associated Variational Problems, *Communications on Pure and Applied Mathematics*, Vol. XLII pp. 577-685, 1989.

[24] N. Paragios, R. Deriche: Geodesic Active Contours and Level Sets for the Detection and Tracking of Moving Objects, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 2. No. 3.,pp. 266-280, Marc 2000.

[25] N. Paragios: Geodesic Active Regions and Level Set methods, *PhD thesis* 2000.

[26] K. Prazdny: Motion and Structure form Optical Flow. In *IJCAI*, 702-704, 1979.

[27] Y. Raja, J. McKenna and S. Gong: Segmentation and Tracking Using Color Mixture Models, *ACCV'98*, pp.607-614. 1998.

[28] J. A. Sethian: Level set methods. *Cambridge University Press*, New York, 1996.

[29] A. Singh: An Estimation-Theoretic Framework for Image Flow Computation. In *IEEE ICCV*, 168-177, Osaka, Japan, 1990.

[30] C. Xu and J. L. Prince: Snakes, Shapes, Gradient Vector Flow, *IEEE Transactions on Image Processing*, Vol. 7, no. 3, march 1998 pp. 359-369