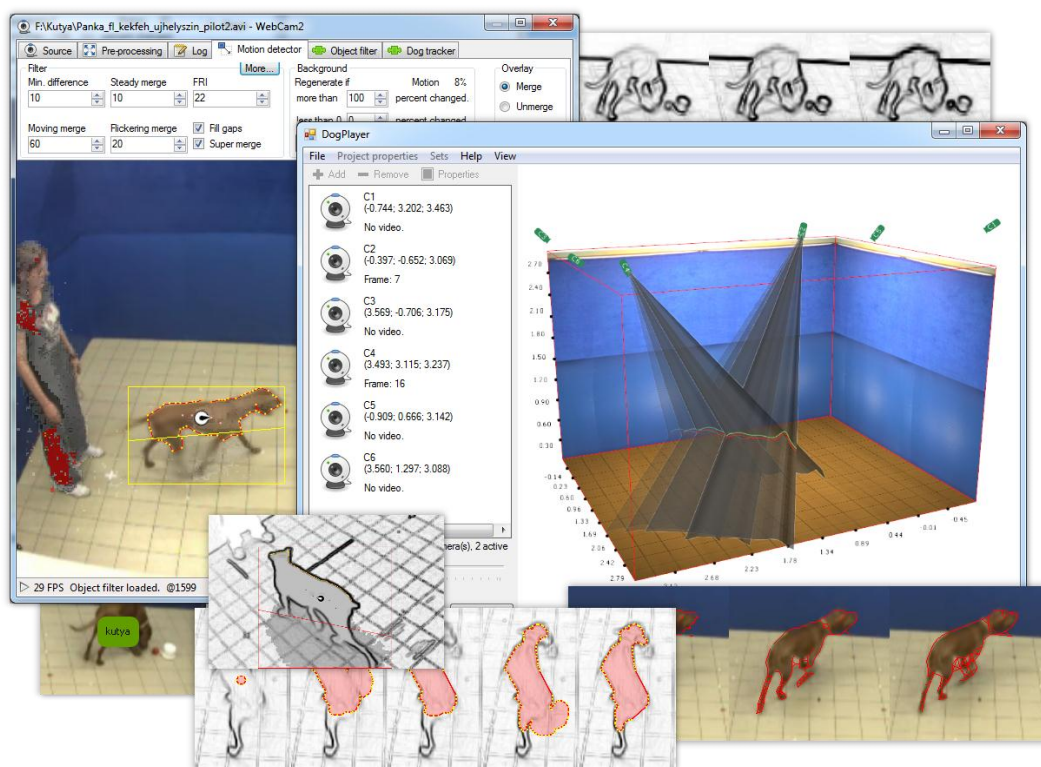


Jelölő nélküli mérőrendszer kutyák mozgásának vizsgálatára



Készítette:

Major Péter

Budapesti Műszaki és Gazdaságtudományi Egyetem, Gépészmérnöki Kar,
Mechatronikai mérnöki alapszak, IV. évfolyam

Témavezető:

Dr. Tamás Péter, egyetemi docens

Budapesti Műszaki és Gazdaságtudományi Egyetem, Gépészmérnöki Kar,
Mechatronika, Optika és Gépészeti Informatika Tanszék

1 Tartalomjegyzék

1	Tartalomjegyzék	2
2	Bevezetés.....	3
2.1	A technika állása	4
2.2	Célkitűzés.....	5
2.3	A megvalósítás eszközei	5
3	Kiinduló rendszer és fejlesztései	7
3.1	Kisebbségi fejlesztések	7
3.1.1	Új videoforrások	7
3.1.2	Feldolgozórendszer fejlesztései.....	7
3.1.3	Modulrendszer	9
3.1.4	Egyéb fejlesztések	10
3.2	Mozgásfelismerés.....	10
3.3	Kommunikáció külső rendszerekkel	14
4	Kutyamozgás elemzés	16
4.1	Élőlények mozgásdetektálása.....	16
4.2	A kutya kiválasztása a képen	16
4.3	A gerinc helyzetének meghatározása egy kamera képén	17
4.3.1	Élkiemelés.....	17
4.3.2	Élfeldolgozás élkövetéssel	19
4.3.3	Élfeldolgozás fékezett vektorgrafikus elárasztással	21
4.4	A gerinc háromdimenziós pozíciójának meghatározása	28
4.4.1	Bevezetés.....	28
4.4.2	Interpoláció	29
4.4.3	Csomópontok transzformációja	32
4.4.4	Héjak metszete	34
4.4.5	Eredmények korrekciója.....	35
4.4.6	Háromdimenziós megjelenítés.....	37
5	Továbbfejlesztési lehetőségek.....	39
6	Eredmények értékelése és konklúzió	41
7	Irodalomjegyzék	43

2 Bevezetés

Mindennapjainkban egyre elterjedtebbek az olyan automatizált rendszerek, melyek képfeldolgozást is felhasználnak működésükhöz. A vizuális szenzorok és az ezekhez kapcsolódó programok használatának nagy előnye, hogy gyorsan képesek olyan összetett információk szerzésére, amelyre más módon nem lenne lehetőség. Ez pedig jól felhasználható például a gyártástechnológiában a legyártott termékek nagy sebességű ellenőrzésére (pl.: üveggyártás, nyomdaipar), különböző forgalomirányító, ellenőrző rendszerekben, biztonságtechnikában, robotrendszerekben, gesztusvezérlésben, mérésekhez stb.

A gépi látás kialakításának kezdetben nagy gátat szabott a számítógépek erőforrásainak szűkössége, manapság azonban az átlagos PC-k is képesek egyszerűbb képfeldolgozási algoritmusok megfelelő sebességű futtatására. A mai tendenciák, amelyek például a processzor(mag)ok számának növelésében és az egyre nagyobb teljesítményű, már tudományos számítások céljára (GPGPU¹) is használható grafikus gyorsítók elterjedésében nyilvánulnak meg, sokat segítenek az egyre kifinomultabb képfeldolgozó rendszerek kialakításában. Általában az ilyen algoritmusok jól párhuzamosíthatók és SIMD² típusú műveletek is igen gyakoriak. Ennek ellenére a számítógépes látórendszerek fejlesztése még a tudomány úttörő területeihez tartozik, mivel az ilyen programok megvalósítása nagyon összetett problémák megoldását kívánja.

Bár az emberi látás nem tekinthető pontos mérőműszernek, mégis a legfontosabb szerepet játssza az emberi információszerzésben. Amellett azonban, hogy látásunk információt ad számunkra a körülöttünk lévő világról, azt is figyelembe kell venni, hogy magához a látáshoz, a látottak értelmezéséhez számos előzetes információra van szükségünk. A legfejlettebb kísérleti látórendszerek nagy adatbázisokkal, összetett mesterséges intelligenciával működnek és így próbálnak meg olyan – az ember számára egyszerű – feladatokat elvégezni, mint tárgyak képeken való felismerése, egymástól való megkülönböztetése. Más, alkalmazásban gyakrabban megjelenő programok nem próbálják meg az ember képességeit teljesen utánozni, inkább speciális módszerek alkalmazásával kísérelnek meg videofolyamból információt szerezni.

A dolgozat a Budapesti Műszaki és Gazdaságtudományi Egyetem Mechatronika, Optika és Gépészeti Informatika Tanszékének és az Eötvös Loránd Tudományegyetem Etológia Tanszékének együttműködésében készül, a kutya mozgását megfigyelő mérőrendszer jelölő nélküli megvalósítási módját mutatja be. A létrehozott szoftver az eddig alkalmazott – kutyán elhelyezett markereken alapuló – mérési móddal szemben képes a kutya testhelyzetének vizsgálatára úgy, hogy arra nem kell semmilyen jelölőt sem rakni.

A dolgozat végigköveti a képek feldolgozását, miközben bemutatja magát programot. A módszer az általam tavaly bemutatott háromdimenziós pozíció meghatározó rendszerre épül, amely számos egyéb fejlesztés mellett a kutya gerincének felismerését biztosító modulokkal, továbbá egy a részeredményekből a gerinc háromdimenziós pozícióját meghatározó alkalmazással lett kibővítvé. A kutyát a rendszer a képből annak vizuális tulajdonságai, elsősorban színösszetétele alapján választja ki előzetes adatok alapján. Mivel a mozgásészlelő rendszer önmagában nem alkalmas a kutya

¹ General-Purpose computing on Graphics Processing Units: olyan technológia, mely lehetővé teszi videokártyákon általános célú programok futtatását.

² Single Instruction, Multiple Data: olyan rendszerek gyűjtőneve, melyek egy adott utasítássorozatot több adatelemen egyszerre hajtanak végre, azaz adatszintű párhuzamos feldolgozást végeznek.

testpozíciójának precíz megállapítására, ezért a képen található élek által nyújtott információk kerültek felhasználásra a méréshez. A rendszer a képen található éleket Sobel operátor segítségével kiemeli, majd azokat egy speciális vektorgrafikus árasztásos kitöltéssel a kutya körvonalát leíró töröttvonalá alakítja. A töröttvonal ezután felhasználható a kutya fej-hát-farok vonalának (gerincének) adott képen való helymeghatározására, illetve egyéb - például a lábak helyzetét vizsgáló - mérésekhez is. Az így kapott kétdimenziós vonalat a kamera projekciós mátrixával a térbe visszavetítve olyan legyezőszerű héjszerkezetek kaphatók, amelyek több kamerából felvéve metszetként megadják a kutya gerincvonalának háromdimenziós helyzetét leíró görbét, amelyet még korrigálni kell a kutya méretének ismeretében. Az így létrehozott rendszerrel a kutya mozgása, annak megzavarása nélkül vizsgálható.

A dolgozat bemutatja a feladat megoldása során felmerült számos problémát, azok megoldásait, az alkalmazott módszerek matematikai és informatikai hátterét, optimalizációját, az eredményeket, a rendszer felhasználási és továbbfejlesztési lehetőségeit.

2.1 A technika állása

A ma elterjedten alkalmazott mozgáskövető rendszerek egy része arra épül [1], hogy a vizsgált élőlényen – általában emberen – jelölőket, markereket helyezünk el. Ezen jelölők valamilyen vizuális tulajdonságuk (fényesség, szín, villogás stb.) segítségével könnyen kiválaszthatóak a vizsgált a képből, majd háromdimenziós pozíciójuk meghatározható, ha több kamerával figyeljük egyszerre őket. Ezen rendszerek viszonylag egyszerűen megvalósíthatóak, de számos esetben csak nehezen alkalmazhatók, hiszen a tesztalany előkészítését igénylik. Az ilyen technikák kutyákon való alkalmazásának további nehézségét adja, hogy egyes fajták hosszú szőrűek, így a jelölők megfelelő rögzítése problémás lehet, hiszen azoknak a bőrrel kellene együtt mozognia. További probléma, hogy az állatok viselkedésére hatással lehet a rájuk rögzített jelölők sokasága, valamint stresszel is járhat a procedura számukra.

Újabban gyors fejlődésnek indultak azon rendszerek is, amelyek nem alkalmaznak jelölőket, hanem a vizsgált lény optikai jellemzőit vizsgálják meg:

- **Aktív módszerek [2, 3]:** az aktív módszerek általában elektromágneses sugárzást bocsátanak ki és megvizsgálják a visszaverődő jelet, majd ebből meghatározzák a test felületének háromdimenziós helyzetét. Ez történhet például úgy, hogy egy lézersugárral „letapogatjuk” a vizsgált tárgyat, majd ezt több kamerával érzékelve háromszögeléssel meghatározzuk a pozíciókat. A lézeres megvilágítás lehet látható, vagy nem látható, alak szerint pedig általában pont, vonal vagy valamilyen minta. A távolság mérés történhet a fény el-visszautazási idejének megméréseivel is. Az ilyen módszereknek több hátránya is van, amely a módszertől függően általában magas ár, relatíve lassú működés vagy pontatlanság formájában jelentkezik.
- **Passzív módszerek [4, 5, 7]:** a passzív módszerek esetében a méréshez csak a rendelkezésre álló mérhető mennyiségeket használjuk, ami általában a látható fényt jelenti. Az ilyen módszerek a vizsgált teret több irányból megfigyelve, a látott képeken közös pontokat keresve és azok helyét háromszögeléssel (sztereoszkópiusan) vagy más módszerrel meghatározva nyerik ki a mért eredményeket. Az ilyen módszerek előnye lehet az alkalmazott szenzorok alacsony költsége (CMOS kamerák ma olcsón előállíthatóak), a gyors

működés, ugyanakkor az ilyen mérésekhez komplex algoritmusok és nagy számítási teljesítmény szükséges.

Az én megoldásom a passzív csoportba tartozik, hiszen semmilyen segítséget nem igényel. A választást indokolta az ilyen rendszerek egyszerű, kényelmes használata, alkalmazkodó képessége, illetve az, hogy a kialakított laborban egyszerű kamerák álltak rendelkezésre a méréshez. Alternatívaként jelent még meg a napjainkban egyre jobban elterjedő mélységképet adó szenzorok alkalmazása is. Bár ezek a munka alatt nem álltak számomra rendelkezésre, a program elkészítése során arra törekedtem, hogy a létrehozott algoritmussal felhasználható legyen az esetleg később rendelkezésre álló mélységkép is. Az ilyen szenzorok legegyszerűbben talán a Microsoft Kinect eszközével érhetőek el, de lehetőség van az eszközt gyártó Prime Sense-től való rendelésre is. Az így kapott mélységképekből élkimieléssel megkereshető az objektumok határa, amelyet a program színes képből kiinduló élkimielése helyett, vagy amellet lehetne használni. Ugyanakkor figyelembe kell venni, hogy ezen eszközök hatótávolsága alacsony (3-4m) és felbontásuk is erősen korlátozott, továbbá számos felülettípussal nem működnek kielégítően.

2.2 Célkitűzés

A munka célja egy olyan program létrehozása, mely képes egy mozgó kutya testhelyzetének elemzésére, úgy, hogy meghatározza a kutya fej-hát-farok vonalának (közelítőleg a gerincének) helyzetét. A feladatot konkretizálásához néhány feltételt szabtam:

- A kamerák nem, viszont a megfigyelt kutyák mozognak (álló entitásokkal nem foglalkozom).
- A háttér állandó, vagy – a megfigyelendő mozgáshoz képest – lassan változik.
- A követendő objektumok a háttérből kivehetők, nem olvadnak teljesen bele (legalább a kontúrnál). Bár a mérőlaborban a falak kék színre lettek festve (ugyanakkor a padló szürke színű), a rendszer ezt nem használja fel, mivel a még tavaly létrehozott mozgásdetektornak erre a segítségre nincs szüksége.
- A követendő entitások kontúrnak legalább egy szakasza az idő nagy részében mozog.
- Több objektum is követhető.
- A követendő entitásokat nem kell megjelölni, speciális háttér sem kell.
- A mozgó objektumok az idő nagy részében nem takarják egymást.
- A világítás megfelelő, nem vakít, de nem is sötét, időben gyorsan nem változik, az árnyékok kevésbé láthatóak, kontúrjuk nem éles.
- Tükröződések nincsenek.
- A feldolgozás előben elvégezhető, gyors, az erőforrásokat (több processzormag, GPU) kihasználó algoritmusokkal történik.
- Átlátható, jól kezelhető grafikus felület, beállítások működés közbeni változtathatósága, eredmények valós idejű megjelenítése, továbbítása, rögzítése.
- Bővíthetőség, kompatibilitás más rendszerekkel.

2.3 A megvalósítás eszközei

Mivel a program alapja már korábban készen volt így a fejlesztőeszközök már adottak voltak, bár több kisebb változás történt:

- A program alapja Microsoft Visual Studio 2008-al, .Net 3.5 SP1 keretrendszerben készült, azonban ez portolásra került .Net 4-re, amely többek közt új funkciókat, nagyobb teljesítményt, új többszálú programozással kapcsolatos elemeket tartalmaz. A program egy része C++, illetve HLSL³[9] alapú. A .Net hasznos programkönyvtáraival, bőséges utasításkészletével relatíve gyorsan hozhatóak létre igényes, funkciókban gazdag, jól kidolgozott grafikai felületű alkalmazások. A teljesítményigényes részekben pedig számos mód van nagy sebesség elérésére: a pointerok használata [8], natív⁴ C++ kód beillesztése, párhuzamos feldolgozás, illetve a videokártya képességeinek felhasználása.
- A háromdimenziós megjelenítést régebben a Microsoft XNA 3.1 rendszere biztosította, erről áttértem a Microsoft XNA 4-re. Ez az áttérés a grafikával kapcsolatos részek jelentős átalakítását igényelte, mivel a kód kompatibilitás nem állt fent. A felújításnak köszönhetően számos új funkció áll rendelkezésre, kiemelendő az újabb DirectX verziók használata (eddig csak a DirectX 9.0c volt elérhető), továbbá a Shader Model 3 támogatása, amely számos új függvényt vezet be, és hosszabb⁵ programok írását engedélyezi [10].



A programot az ELTE Etológia Tanszék laborjának számítógépei futtatják a gyakorlatban, azonban a fejlesztést egy több éves notebookon végeztem, amely hozzásegített a program teljesítménynek magasan tartásához. Bár a módszer nem igényel speciális felszerelést, teljesítményének értékeléséhez fontos lehet a felhasznált eszközök paramétereinek ismerete:

- Kamerák:
 - BASLER scA750-60gc Gigabit Ethernet kamerák (6db), felbontás: 658x492 @ 60Hz
- Számítógép:
 - Asus X71Vn notebook, Intel P8600 CPU (2x2,4GHz, 3MB cache, FSB: 1067MHz), 4GB DDR2 RAM (400MHz), nVidia 9650GT GPU (32db SM v4.0 1375MHz-es árnyaló mag, 1GB DDR2 VRAM 400MHz)

A rendszerkövetelmények:

- Bemenet: DirectShow vagy Basler Pylon kompatibilis kamerák, vagy videofájlok megfelelő DirectShow dekóderrel, továbbá képfájlok: PNG, JPG, BMP vagy GIF formátumban.
- Modern, minimum kétmagos CPU: egy kamera képének feldolgozására (640x480 @ 30-50FPS) kb. egy 2-3Ghz-es processzor(mag), két kamerához két mag kell (bár hálózaton is összekapcsolhatóak a számítógépek), legalább 100MB szabad memória.
- DirectX 10 kompatibilis videokártya, Shader Model 3+ feldolgozó egységekkel.
- .Net Framework 4 Client Profile, XNA 4 Framework és megfelelően új DirectX verzió (ezek részei a Windows 7 operációs rendszernek, ha az frissítve van)

³ High Level Shader Language: a grafikus gyorsítók árnyalóinak programozására szolgáló nyelv

⁴ Olyan program, amely nem egy futtatási keretrendszeren (pl.: .Net, Java) fut, hanem közvetlenül gépi kódba fordítjuk, és úgy hajtjuk végre. Ilyenkor a számítógép erőforrásai közvetlenebbül elérhetők, ami nagyobb gyorsaságot eredményez.

⁵ A Shader Model 3 a kettes verzióhoz képest az alkalmazható utasításszám minimumot 96-ról 65536-ra növeli.

3 Kiinduló rendszer és fejlesztései

A dolgozat nagyban épít az előző TDK-ra beadott Háromdimenziós mozgáskövető rendszer jelölőeszközök használata nélkül [11] című dolgozatomra, így ebben a részben az alapot képező rendszer rövid ismertetésével, és az újdonságok bemutatásával foglalkozom.

3.1 Kisebb fejlesztések

Mint minden mérési feladatban, itt is az első lépés a mérőműszer kalibrálása. Bár egy ilyen kamerás mérés fő előnye és célja, hogy azt gyorsan, egyszerűen lehessen elvégezni – és ez részben a pontosság rovására történik –, mégis igen fontos, hogy a felhasznált kameráról a lehető legpontosabb adatokat szerezzük be. A kalibrációt ebben az esetben nem én végeztem, mivel azt a mérőlaborban már megoldták. A kalibráció eredményeképpen adódott intrinsic és extrinsic mátrixok, valamint a képtorzítási adatok a már kialakított felületen a rendszerbe bevihetőek voltak.

3.1.1 Új videoforrások

A rendszer a videót számos új forrásból is el tudja érni, ehhez a videoelérés egységesítve lett, így minden képforrás azonos módon tud a főprogramba kapcsolódni és a modularizált rendszer könnyen bővíthető új egységekkel. Míg a régi verzióban csak DirectShow videoeszközök támogatása volt megoldott, azóta számos új modul készült:

- **Videofájlok feldolgozásának lehetősége:** A rendszer most már fájlokból is képes videóhoz jutni, ami tesztelés szempontjából kiemelkedő fontosságú, továbbá felhasználható nem valós idejű feldolgozás esetén is. A fájlok tetszőleges formátumúak lehetnek, ha rendelkezésre áll a számítógépen a megfelelő DirectShow dekóder. A DirectShow támogatáshoz most már saját megoldást készítettem (szemben a webkamerákhoz használt opensource DirectShow wrapperrel) C++\CLI nyelven. A rendszer automatikusan kiválasztja a fájlhoz használandó videokodeket, és felépíti a megfelelő dekódolási szűrőláncot. Biztosított a videó folyamatos lejátszása, pozicionálás és a lejátszási sebesség állítása is. A rendszer a kapcsolódó hangszávot nem dolgozza fel, csökkentve ezzel a CPU terhelést. A videofájlok feldolgozása most elsősorban a valós idejű működés szimulációjára szolgál, a videót egy fix képkockasebességgel küldi a programba, a batch feldolgozáshoz jobban alkalmazható nem állandó sebességű feldolgozás elkészítése még folyamatban van.
- **Basler Pylon eszközök használata:** a rendszer képes az iparban használt BASLER kamerák elérésére és vezérlésére. A rendszert a Pylon .Net API-ra [12] építve érem el. A Pylon rendszer nagy előnye, hogy segítségével például Ethernet hálózatra kapcsolt kamerák is elérhetőek távolról. A program eléri a konfigurációs beállításokat is, így lehetőség van például az expozíciós idő állítására stb.
- **Képfájlok beolvasása:** ez elsősorban tesztelésre szolgál.

3.1.2 Feldolgozórendszer fejlesztései

Mint azt már az előző dolgozatban ismertettem a képfeldolgozó-rendszer több szálon fut, és egy diszpécser algoritmus osztja szét a képkockákat a futó szálak között. A rendszer lehetővé teszi, hogy egyszerre több képkocka feldolgozását is megvalósítsuk, mivel a feldolgozás lépésekre osztható, és egy adott lépés elvégzéséhez elég az, ha azzal már végeztünk az előző képkockán, nem kell annak teljes elemzését befejeznünk. Természetesen vannak olyan lépések, amelyek nem függnek a többi képkockától, ezek az intra-frame lépések tetszőlegesen elvégezhetők párhuzamosan.

A feldolgozórendszerben is több jelentős fejlesztés történt. A grafikus kártya által működtetett előfeldolgozó immár nem a diszpécser szálon működik, mivel ez szűk keresztmetszetet jelentett azokban az esetekben, ahol a nagy képkockasebesség miatt nem volt lehetőség arra, hogy az adatok videomemóriába töltését és visszaolvasását megvárjuk, mivel ez csak egy szálon történt. A jelenlegi verzióban már minden feldolgozó szál külön előfeldolgozót üzemeltet, és a videokártyát több szál is használja egyszerre. Teljesítményprofilozás során kiderült, hogy jelentős időt vesz igénybe, amíg a grafikuskártya egy adott képkockát tartalmazó céltextúrát felold írás után, így annak visszaolvasását nem érdemes azonnal elkezdni, hanem a következő rajzolás előtt is meg lehet ezt tenni. Ez azért is célszerű, mert a visszaolvasó parancs pollozva várja az erőforrás felszabadulását, így hasztalanul használja a processzort. Az egy képkocka késéssel történő kiolvasás a hasznos CPU időt jelentősen növeli, ugyanakkor késleltetést is visz a rendszerbe, de ez elfogadható.

A képkocka előfeldolgozása a grafikus kártyán futtatott HLSL kódon írt programokkal történik. Itt kerül elvégzésre a képen jelentkező különböző (hordós, radiális, tangenciális) torzulások kezelése, a kép átméretezése és most már a színterek közti konverzió is. A rendszer a képeket RGB, YUV és HSL rendszerből tudja kiolvasni és ezek bármelyikébe át tudja alakítani. Változást jelent, hogy míg a régi verzióban csak 24 bites RGB formátumban történhetett a képek feldolgozása, most már adott a lehetőség az előbb említett mindhárom formátum használatára. A képek már nem 24, hanem 32 bites színmélységgel tárolódnak, amely egy negyedik színcsatorna bevezetésének köszönhető. Ennek egyik fő oka az, hogy a videokártya is ebben a színmélységben tudja visszaadni a képeket (ARGB32), másrészt a plusz csatorna felhasználható további információk, például élintenzitás tárolására. A Canny féle élkimielés bemutatására majd a dolgozat későbbi fejezetében kerül sor. A rendszer képes továbbá a BASLER kamerák által használt YUYV 4:2:2 szintér kezelésére is, ahol a színértékek csak minden második képpontra vannak külön megadva, és csak a fényesség definiált minden egyes pixelre. A szintérkonverziós és képtorzítás-korrigáló HLSL kódok külön függvényekbe kerültek, így azok egymás után könnyen hívhatók, ugyanakkor minden egyes átalakítási mód saját árnyalófüggvényt kapott, amely meghívja az átalakítások egy csoportját. Ez utóbbi azért szükséges, mivel a függvények minden mintavételezési pont esetén újra meghívásra kerülnek, így a bennük használt plusz elágazások érzékelhető növekedést okozhatnak a feldolgozási időben, további ok az árnyalók korlátozott hossza. A függvényhívásoknak nincs jelentősége, mert HLSL esetén minden függvény inline-olásra kerül. Mivel már lehetőség van például olyan előfeldolgozási láncra, amely a képet egyszerre YUYV szintérből átalakítja RGB24-be, közben korrigálja a torzításokat, kiszámolja a Canny féle élintenzitást maximumelnyomással, majd az eredményt már YUVE32 (az E az élinformációt jelenti) formátumban adja vissza, ezért már nem volt elég Shader Model 2 96 darabos utasítás korlátja, hanem szükség volt a Shader Model 3 512-es maximumára. Így az árnyalókat átírtam az új verzióknak megfelelően. A rendszer két különböző felbontásban állítja elő a képet, melyek közül a kisebbiken történik a feldolgozó műveletek orozlánrésze, ugyanakkor a nagyfelbontású változat szolgál például élfeldolgozásra, vagy a feldolgozás közbeni videó megjelenítésre. A kisebb felbontású verzió átlagoltsága segít a zaj csökkentésében is. Tipikusan a kis felbontású verzió 320x240 px, míg a nagy ennek egész számú többszöröse (a könnyű összehasonlíthatóság érdekében).

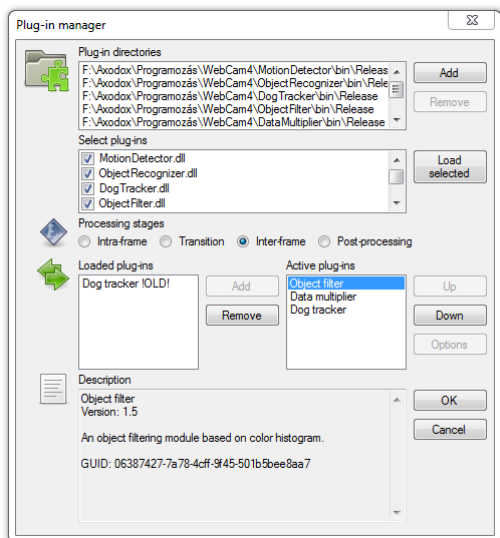
Az előbbiekhöz kapcsolódó fontos változtatás, hogy a képfeldolgozást immár nem csak RGB, hanem HSL és YUV szintérben is lehet végezni. Az előfeldolgozóban állítható, hogy az előbbi három szintér közül melyikben kívánjuk a kimenetet megkapni. Ez egyben azt is jelenti, hogy a programban a

színinformációk ezen színtérben kerülnek mindenhol tárolásra, így fel kellett készíteni a régi modulokat ezek támogatására. Mivel az előfeldolgozó immár az új színterekben adja az eredményeket, így a videó megjelenítésekor azokat vissza kell alakítani. Bár ez elsőre nem tűnik hatékony megoldásnak, hiszen a színterek között el-vissza alakítunk, a gyakorlatban használt YCbCr színtér esetén ez egy mátrixszal való szorzásban is leírható, és mivel a grafikus kártya így is - úgy is használ árnyaló programokat, ez nem jelent jelentősebb GPU terhelést. Az átalakítás ugyanakkor nagyon hasznos, hiszen megkönnyíti például az árnyékok hatásának csökkentését azáltal, hogy szétválasztásra kerülnek egymástól a színösszetételre és a világosságra jellemző információk. Kezdetben HSL színtérrel kísérleteztem, de ennek számos hátránya van, ugyanis a színter komponensek nem függetlenek egymástól és egy színnek több erősen eltérő reprezentációja is lehetséges. Például ha a szaturáció nulla, akkor a színezet tetszőleges értékénél állandó világosság mellett ugyanazt a színt kapjuk. Ugyancsak nehezíti az összehasonlítást, hogy a színezet körkörös felírású, így annak minimum és maximum értéke közel ugyanazt a színt jelöli. További hátránya a HSL színtérnek az átalakítás viszonylagos összetettsége. Ezt kiküszöbölendő döntöttem az YCbCr színtér használata mellett, melyet már a BASLER kamerák programhoz való illesztésénél megismertem. Mint köztudott ez a színtér egy világosságjelet és két színkülönbségi jelet (vörös-világosság, kék-világosság) tartalmaz, és ez nagyon megkönnyíti a képek színeinek elemzését. A színek és a világosság szétválása segíti a színek összehasonlítását, de hasznos az árnyékok hatásának csökkentése szempontjából is. A rendszer az összes modult tájékoztatja az éppen használt színterről, így azok megfelelően alkalmazkodhatnak. A CPU tömeges színtérkonverziót nem végez, ugyanakkor létrehozásra kerültek a színtérkonverziós metódusok C# nyelven is, így a modulok azok segítségével könnyen elvégezhetik a konverziókat.

A feldolgozó futószalagok is fejlesztésre kerültek: most már biztosított az inter-frame feldolgozás teljes lépésekre darabolása, így a párhuzamos feldolgozás mellett jobban kihasználható a processzor. Nyilvántartásra kerül, hogy az éppen feldolgozott képkockák a feldolgozás hányadik lépésénél tartanak, így az ütemezés során ennek segítségével maximálisan kihasználhatók a feldolgozószálak. Az éppen várakozó szálak számára a rendszer megszakítások segítségével tudja jelezni, készen állnak-e a szükséges információk a feldolgozás folytatásához. Továbbfejlesztésre került a hibakezelő rendszer is, amely védi a programot az összeomlástól az egyes szálakon bekövetkező hibák esetén. A rendszer a hibák sűrű ismétlődése esetén leállítja a feldolgozást, ugyanakkor más esetben az tovább futhat. A hibákról is azok keletkezési helyéről és idejéről a programnapló ad tájékoztatást.

3.1.3 Modulrendszer

A rendszer egyik legfontosabb fejlesztése az, hogy a feldolgozó modulok most már nincsenek bebetonozva a főprogramba, hanem azok egy feldolgozó szalag szerkesztővel rugalmasan betölthetőek. Ezt az tette lehetővé, hogy kezdetben egy közös őosztálytól származtattam le az összes feldolgozómodult. Ha a közös őst egy különálló DLL-ben tároljuk és azt behivatkozzuk a modulokat adó DLL-ekbe illetve a főprogramba is, akkor megoldható, hogy reflektórt alkalmazva a modulokat futás közben betöltsük a főprogram alkalmazás tartományába (application domain), majd a létrehozandó osztály nevét ismerve (én mindet ugyanazzal a névvel láttam el az egyszerűség érdekében) annak egy példányát létrehozunk, és az őosztályba kasztoljuk. Ezek után az osztály a közös interfészen keresztül elérhető. Ez a változtatás egyben a modulok osztályainak illesztését is igényelte, hiszen most már nem a program leállításáig maradtak a memóriában, így a memória manuális felszabadítását is meg kellett oldani. Ugyan a .Net gondoskodik a memória felszabadításról,



1. ábra A feldolgozási-lánc szerkesztő

a közös őssztály biztosítja (annak változtatása viszont a modulok újrafordítását igényli). A program kezelőfelületén megadható a modul DLL-ek helye, ezután listáról választhatóak a használni kívánt modulok. A betöltött modulok rendezésre kerülnek aszerint, hogy melyik feldolgozási lépéshez tartoznak, innen rakhatóak át az aktív modulok listájára, ahol sorrendjük is módosítható. A rendszer minden modult azok egyedi azonosítója alapján ismer fel, így azok program indítás utáni újratöltése egyszerűen megoldható. A rendszer felismeri, ha a megadott DLL hibás, vagy a program régi verziójához tartozik, és erre figyelmezteti a felhasználót.

Az képfeldolgozó osztályok emellett fel lettek szerelve az egymással való kommunikálás képességével, ugyanis létrehozásra került egy statikus adatszótár, amelyben az osztályok példányai tetszőleges típusú tartalmat névvel azonosítva elhelyezhetnek és ezt a többi modul kiolvashatja. Ugyanezen helyre a főprogram is elhelyezhet „közérdekű” információkat. A modulok ezen kívül tetszőleges típusú argumentummal ellátott szöveges parancsokat adhatnak a főprogramnak. Így tetszőleges adat átvihető a parancssal.

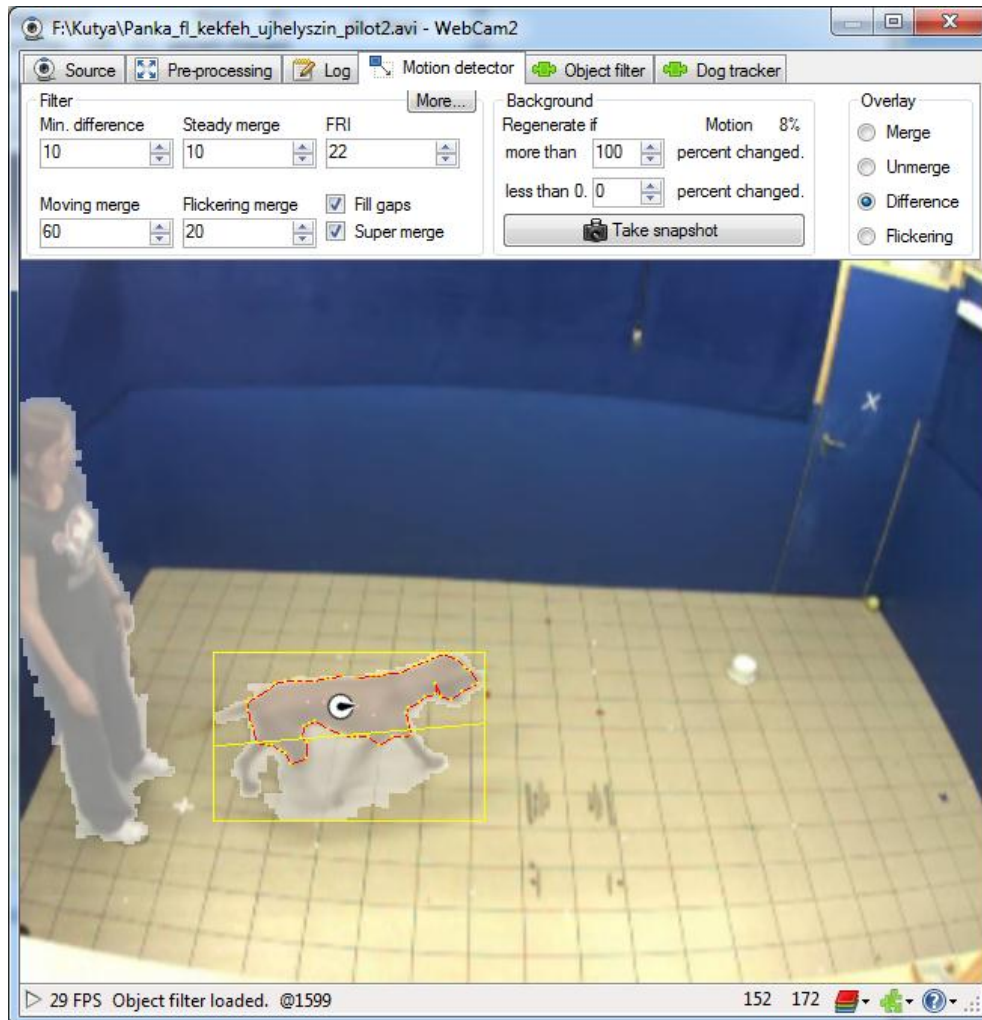
3.1.4 Egyéb fejlesztések

A rendszer mozgásdetektáló algoritmusának alapelve ugyan nem változott, de igen jelentős változtatásokon esett át, és szinte minden részlet további finomhangolásra került. Emellett olyan funkciókkal is kiegészítésre került ez a rész, amely lehetővé teszi élőlények jobb minőségű követését. Az objektumfelismerő és követő modul bizonyos feladatait is átvette, olyannyira, hogy az a jelenlegi esetben nem is vesz részt a feldolgozásban. Az adattovábbító modul is nagy átalakításra került, mivel egyrészt jelentősen változtak a leírandó adatok, másrészt megjelent az igény az adatok fájlba való mentésére is, hogy az egyes kamerák mérési eredményei később is visszanezhetők legyenek. A régi adatfúziós alkalmazás helyett ez a projekt egy teljesen új alkalmazást használ, amely a réginél sokkal több funkcióval rendelkezik, és a kutya mozgás mérése miatt a működése is jelentősen eltér. A többi feldolgozó modul újonnan került a rendszerbe, így azok, valamint a többi jelentősen módosított, vagy teljesen átirított modul bemutatására a következő fejezetekben kerül sor.

3.2 Mozgásfelismerés

A mozgásfelismerő rendszer alapját egy már régebben elkészített modul [13] képezte, amelyet már az előző TDK dolgozatomban is bemutattem, az alkalmazott számos változtatás, finomítás és a

programban játszott fontos szerepe azonban megkívánja annak rövid, összefoglaló ismertetését. A mozgásdetektor működési elvének lényege, hogy az egymás után következő képkockákból egy háttérrel épít fel, amellyel összehasonlítva az adott képkockát megkeresi a változások helyét. A rendszer ezt az adaptív háttérrel kezdetben az első képkockára állítja be, majd az egymást követő képkockák során megfelelően változtatja. Így nincs gond akkor sem, ha kezdetben mozgás volt a videón, illetve kezelhetők a képen beállt változások is.



2. ábra A mozgásérzékelő modul működés közben

A működés több részfolyamatból áll (a képkockaszámok 30 FPS sebességű videóra vonatkoznak):

- **Különbségképzés:** A rendszer a háttérrel összehasonlítva a jelenlegi kép tartalmát, térképet készít arról, hogy hol van eltérés és hol nincs. Az összehasonlítás során figyelembe kell venni az alkalmazott színteret is: például YCbCr színtér esetén célszerű a világosság közötti eltérés maximális értékét nagyobbra választani, mint a színekét, mert így csökkenthető az árnyékok hatása. A rendszer lehetővé teszi az élintenzitás térképek összehasonlítását is. Ez utóbbi ugyan csak az élek mentén működik, azonban ez nem jelent problémát, mivel később a mozgó területekben esetenként létrejövő lyukak feltöltésre kerülnek.
- **Statikus háttérfrissítés:** Ha egy helyen nincs eltérés, akkor a rendszer számolja, hány képkockán keresztül van ez így, és egy adott érték elérésekor beolvasztja a háttérbe az adott

részt (kb. 20 képkockánként). Ezzel elkerülhető a megvilágítás változásából eredő kis eltérések kumulációjából adódó hiba.

- **Kiterjesztett mozgó régiók:** A rendszer készít egy térképet a mozgó régiókból, azok területének 1-2px szélességű kiszélesítésével, ezen zónákban a háttér statikus frissítése letiltásra kerül. Ennek az az oka, hogy a részben megváltozott pixelek háttérbe való beolvasztása rontaná az eredményeket.
- **Dinamikus frissítés:** A rendszer tovább elemzi a mozgó részeket, és azokat egy mozgó háttérrel hasonlítja össze, amely tartalmazza a kép mozgó részeinek képét az előző képkockákról. Így felismerhető, hogy egy tartós változásról van-e szó, vagy folyamatos mozgás történik az adott helyen. Ha az adott pixel eltér a mozgó háttértől, akkor az frissítésre kerül a pixel új állapotával, illetve a változatlanág időtartamát mérő számláló is nullázásra kerül. Ha a pixel sokáig nem változik, akkor a változatlanág-mérő-számláló feltelik (kb. 60 képkocka), és a pixel beolvasztásra kerülhet (esetenként nem, lásd később).
- **Villódzáscsökkentés:** A rendszer figyeli, hogy milyen gyakran vált egy adott képrész mozgó és álló állapot között, ugyanis ha ez túl gyakori, akkor nagy az esélye annak, hogy valójában nem igazi változás történt, hanem az álló háttérben lévő pixelhez nézve az annak megfelelő pixel a képkockákon ép az eltérés határán van. Ezt a kedvezőtlen jelenséget a rendszer az ilyen pixelek háttérbe való frissítésével oldja meg. A változást mérő számláló attól függően, hogy páros vagy páratlan, csak a mozgó illetve csak az álló állapotra való váltáskor nő. Emellett adott képkockaszámonként (kb. 20) az értéke csökkentésre kerül egy mennyiséggel, így ténylegesen a változások gyakoriságát tudja jellemezni.
- **Lyukfeltöltés:** A program felölti a mozgó régiókon belül található lyukakat, ugyanis előfordulhat, hogy egy mozgó képrészen belül véletlenül olyan színű képpont helyezkedik el, mint az álló háttérben, így ott lyuk keletkezne. A gyakorlatban nagyon valószínű, hogy az objektumok kontúrja jobban eltér a háttértől, mint azok belső részei, ugyanakkor nincs szükség „lyukas” entitások követésére. A lyukak feltöltése úgy történik, hogy a kép szélén lévő álló pixelekből a mozgást leíró térképet elárasztjuk, majd azon nem mozgó részeket, amelyekre így nem sikerült eljutni átbillentjük.
- **Biztonsági háttér:** Az algoritmus a dinamikus beolvasztás során biztonsági másolatot készít a beolvasztott részekről, amelyek később a lyukfeltöltés során visszaállíthatók. Ez megakadályozza az egyszínű és/vagy lassan mozgó entitások részeinek háttérbe való beolvasztását.
- **Hibaészlelés:** A rendszer folyamatosan figyeli a mozgó és a villódzó pixelek számát, és határérték elérése esetén az egész képet beolvasztja az álló háttérbe, gyorsítva a hiba (valószínűleg gyors fényviszony változás vagy a kamera elmozdulása által okozott zavar) kijavulását.
- **Entitásorientált beolvasztás (opcionális):** A mozgásdetektáló rendszer eredetileg autók követésére készült, amelyknél adott volt, hogy kontúrok szinte mindig mozognak. Ez azonban kutya és emberek esetén már nem teljesül mindig, így meg kellett oldani, hogy ne lehessen a háttérbe olvasztani a csak részben mozgó entitásokat. Olyan kiegészítéssel láttam el a rendszert, amely már a mozgásfelismerés szakaszában felismerte az összefüggő mozgó területeket a képen, és elemezte, hogy az adott területen hány pixel változik folyamatosan. Ha elég sok pixel változott egy egyébként álló területhez kapcsolódva, akkor a rendszer nem olvasztotta be az adott területet. Azt, hogy mekkora résznek kell egy területen mozognia, azt

a programnak százalékban lehet megadni. Emellett lehetőség van a pixelek beolvasztási számlálójának minimum átlagos értékét is megadni. Ez a kiegészítés biztosítja a kép szélén tartósan szereplő tárgyak helyes érzékelését is. Azonban problémákat is okozott: a százalékos megadás miatt a kis mozgó részek nem, vagy csak lassan tudtak a háttérbe olvadni, hiszen nekik néhány pixel is jelentős részt képviselt, így alkalmazni kellett egy minimum méret kritériumot. Ez nem jelentett problémát, mert az itt követendő kutya és emberek viszonylag nagyok. Ezt a szűrést később a túl nagy objektumokra is kibővítettem, ugyanis ezek általában valamilyen gyors képváltozásból erednek, és ez az új módszer lassította e helyzetekhez való alkalmazkodást. Az összefüggő részek felderítését az objektumfelismerő algoritmus megfelelő részéből vettem át. Az eldobott objektumokat a rendszer törli az azonosítók térképéről. Ez a módszer lehetővé teszi a képről tartósan lelógó objektumok megfelelő kezelését is, ugyanis azok eddig a kép határának elérésekor könnyen beolvadhattak a háttérbe, mivel itt általában nem volt nagy intenzitásváltozás az idő folyamán egy adott pozícióban.



3. ábra Élőlények mozgásdetektálása: a pirossal jelölt területeket a régi rendszer beolvasztotta volna

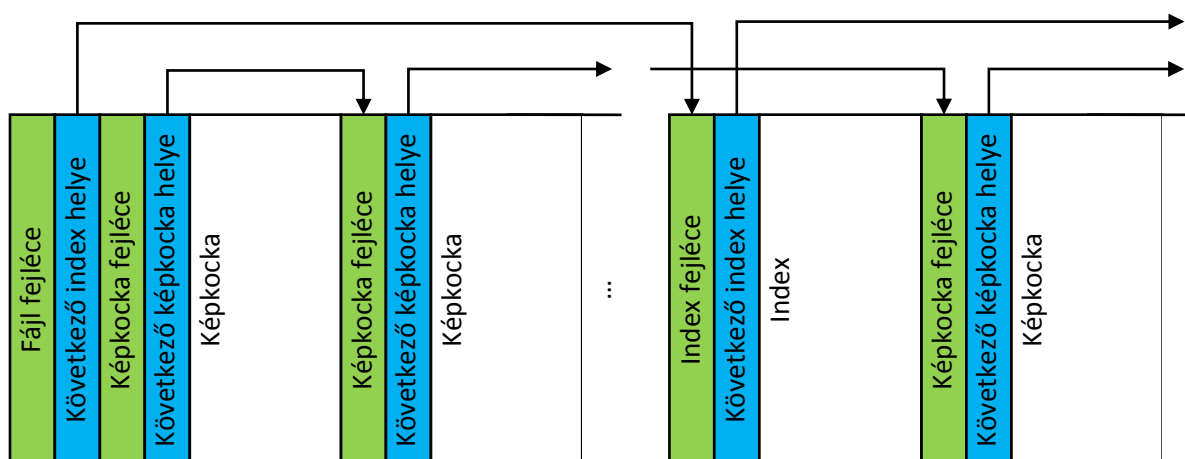
- **Entitás azonosítás (opcionális):** Az utóbbi ponttal összhangban az entitásorientált beolvasztás bekapcsolásakor a modul átvette az objektumkövető szerepének egy részét. Így itt készül el a követett entitások azonosítóit a képhez rendelő térkép, és itt kerül összeállításra az objektumokat tartalmazó lista is. Ugyanakkor az objektumok képkockák közötti követésére már nem kerül sor, ezt továbbra is a régi objektumkövető modul végzi el.

Ezen folyamatgyűttes segítségével jól kiválaszthatóak a képből a mozgó régiók kis CPU terhelés mellett. Ugyanakkor az eredmények az árnyékok, a képzaj és az esetleges hasonló színek miatt nem alkalmazhatóak közvetlenül egy tárgy körvonalának meghatározására.

3.3 Kommunikáció külső rendszerekkel

Ahhoz, hogy az elkészült rendszer használható legyen, külső programokkal további feldolgozás (például az egyes kamerák eredményeinek összevetése) céljából szükség van az eredmények továbbítását biztosító modulra is. Mivel sor kerülhet valós idejű és köteget feldolgozásra is, ezért mindkettőhöz kell támogatást nyújtani. Ezt a feladatot a DataTranmitter nevű modul látja el, amely képes az adatok hálózati továbbítására és most már fájlba mentésére is. Ahhoz, hogy az adatok továbbíthatóak legyenek először fel kellett újítani az őket bináris formába alakító, illetve onnan visszaolvasó eljárásokat. Ezek a procedúrák adatfolyamra (Stream) dolgoznak, ami előnyös mert így használhatók a .Net által felkínált ki/bemeneti lehetőségek számos formáján, csak az adatfolyamot megvalósító objektumot kell átadni nekik. Az átalakítás azonban csak a továbbítás egyik fele, hiszen biztosítani kell az adatok struktúrált adatfolyamba rendezését: ez az alfejezet ennek módját mutatja be fájlban való tárolás esetére. A fájlba való adatmentés esete abban tér el a hálózati kommunikációtól, hogy itt biztosítani kell a képkockák közti könnyű navigációt, akár a fájl sérülése, hiányossága (esetleges programhiba) esetén is. A fájlok bináris formátumúak, hiszen képeket is tárolnak, ráadásul a szöveggé alakítás lassítaná az adatok kiírásának sebességét is.

Minden fájl egy fejléccel kezdődik, amely ASCII szöveg formátumban tárolja a fájl típusának azonosítóját (így a kiterjesztés elvesztése nem probléma), majd az első index terület pozíciója következik. Mivel a képkockák több olyan információt is tartalmaznak, amelynek mérete függ a képkocka tartalmától, így azok eltérő szélességűek lesznek az adatfolyamban. Ha el szeretnénk kerülni, hogy egy egész fájlt kelljen beolvasni ahhoz, hogy az összes képkocka helyét tudjuk, tartalomjegyzéket kell elhelyezni a fájlban, mely leírja, hányas képkocka hol helyezkedik el. A tartalomjegyzék az én fájlformátumomban több darabban helyezkedik el, úgy hogy adott számú képkocka kiírása után a rendszer beszúr egy tartalomjegyzék darabot a fájlba. Ezek a jegyzékek tartalmazzák a képkocka sorszámát, rögzítési idejét, helyét és az általa hordozott információfajtákat (van-e benne információ a kameráról, objektumokról, képről stb., flagek formájában), így lehetővé válik, hogy ezen adatok segítségével könnyen navigáljunk nagy fájlokban is. Minden jegyzék elején van egy mutató a következő jegyzék helyére. Ha a fájl megsérülne, akkor is lehetséges az adatok ép részének visszaolvasása, ugyanis minden képkocka, és jegyzék fejléccel van megjelölve, továbbá minden képkocka elején van egy a következő képkocka elejére vonatkozó mutató. Adott mennyiségű képkocka (jelenleg 300) után kiírom a jegyzéket, a felvétel végén a maradék az utolsó képkocka után kap helyet.



4. ábra Az eredményfájlok felépítése

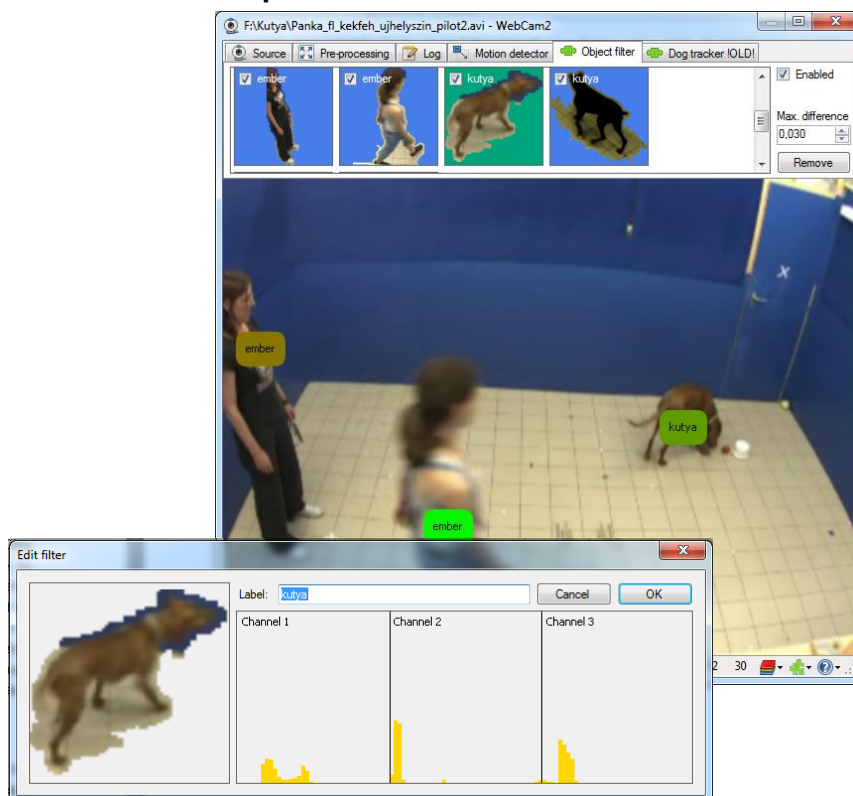
A beolvasó rész a fájlt megnyitva beolvassa annak teljes indexét, ezután már tetszőleges képkockára, akár idő szerint is lehetséges a keresés (a már sorbarendeztett listán intervallumfelezéses keresést implementáltam). Sokszor hasznos lehet, ha rendelkezésre áll egy-egy képkocka képe (tájékozódáshoz, melyik kamera eredményeit látjuk stb.), ugyanakkor nincs szükség már mindegyikre: a rendszer az index alapján könnyen kikeresheti a képet is tartalmazó elemeket. A program jelenleg nem alkalmaz tömörítést, azonban választható csökkentett mennyiségű képkocka kiírása (minden második, harmadik stb. tartalmaz csak képet, de mind tartalmaz mérési eredménylistát).

4 Kutyamozgás elemzés

4.1 Élőlények mozgásdetektálása

Az élőlények érzékelése kiindulópontként azért jelentett problémát, mert az eredetileg autók vagy más tárgyak érzékelésére szánt rendszer nem volt képes olyan helyzetek kezelésére, amelyeknél egy adott entitásnak csak egy része mozog, nagy része viszont nem. Az eredeti algoritmus ilyenkor a nem mozgó részeket elkezdte beolvasztani a háttérbe. Bár a mérőszoba falai kékek voltak, a padló színe az alkalmazott megvilágításnál hasonlított az emberi bőr színére, továbbá egyes esetekben a vizsgált kutya szőrének színére is, így egy egyszerű színszűrésen alapuló algoritmus csak kevésbé volt alkalmazható. Továbbá a probléma megoldási lehetőségeit megvizsgálva kiderült, hogy az a jelenlegi rendszerrel is megoldható úgy, hogy a mozgásdetektáló szinte bárhol – kék háttér nélkül – alkalmazható marad.

4.2 A kutya kiválasztása a képen



5. ábra Színhisztogram alapú szűrés

Ahhoz, hogy meghatározzuk a kutya helyét, először ki kell választanunk a kutyát a képen a mozgó dolgok közül. Sajnos a kutyák nagyon sokfélék lehetnek, ezért általános – minden kutyafajtával működő – algoritmus kidolgozása nagyon nagy munka lenne, hiszen a valószínűleg neurális hálóval történő megoldáshoz igen sok kép kellene, amelyeken a kutya ki van jelölve. Ez utóbbira jelentős bizonyíték, hogy az emberi arcok képen való felismerésére sincs még tökéletes módszer, pedig ott sokkal hasonlóbb dolgokat kell felismerni. Ugyanakkor a kutya bemutatása egyszerűen megoldható, ezért jelenleg egy egyszerű színösszetételén alapuló megoldást használok. A mérés előtt a kutyát a kamerák elé visszük, ahol az operátor a programnak megmutatja melyik követett elem a képen a kutya, a rendszer pedig összeállítja a kutya színhisztogramját. A hisztogram

segítségével a kutya könnyen felismerhető a jó megvilágítású teremben. A többi követett entitás is azonosítható így, ha színösszetételük tetszőleges irányból jól megkülönböztethető. A rendszer működés közben összehasonlítja a tárolt mintákkal a feldolgozott videón látható tárgyakat, és ha a hisztogramok hasonlósága megfelelő mértékű, akkor a megfelelő címkével látja el az objektumot, amelyet a későbbi modulok kiolvashatnak, hogy kiválasszák, milyen típusú elemeken szeretnének dolgozni. A rendszer a három színcsatornára egyenként 32 rekeszű hisztogramot készít, összehasonlításakor kiszámítja a hisztogramok osztályaira jutó átlagos eltérést, és ezt hasonlítja egy maximális értékhez. Ha az eltérések a megfelelési határon belül vannak, akkor a legkisebb eltérést hordozó választást hozza a rendszer. Az osztályokra vonatkozó átlagos eltérés biztosítja, hogy más hisztogramszélesség esetén se kelljen változtatni a határon.

4.3 A gerinc helyzetének meghatározása egy kamera képén

Mivel a kutya gerince a hátának felületéhez közel van, ezért helye megbecsülhető a hát középvonalának helyzetéből. Ha a kutya hátának vonalát tudjuk érzékelni több kamerával, akkor pozíciója meghatározható, ugyanis a hát formáját ismert méretű (lemérhető) hengerdarabokból összeállított objektumnak képzelhetjük. Ez utóbbi nyilván mutatja, hogy itt kis részben becslést is alkalmazunk, ami pontatlanságot okoz, de az etológusoknak az állat mozgásának elemzéséhez nincs szükségük ennél nagyobb mérési precízióra.

A gerinc kiválasztásához biztosított a kutya által elfoglalt terület megjelölése a képen, ez azonban nem használható közvetlenül a körvonal meghatározására, ugyanis számos esetben előfordul, hogy a kijelölt terület nagyobb (néha kisebb) a kutyánál. Ennek egyik oka az árnyékok jelenléte (amelyek hatása ugyan csillapított, de nem tökéletesen), továbbá a színek különbség alapú mozgásdetektálás bizonyos helyzetekben létrejövő pontatlansága a határterületen. Becslést ugyan végezhetnénk, de méréshez ez már nem elég megbízható. Meg lehet próbálni színszűrést alkalmazni a kutyára, de ez meg azért problémás, mert igen nehéz - ha lehetséges egyáltalán - olyan színtartományt (lehet nem összefüggő is) megadni, amiben az egész kutya benne van, tetszőleges pózban, helyen, de a környezet sohasem.

Hamar beláttam, hogy a legegyszerűbben pontosabb eredményt éldetektálás segítségével érhetünk el. Az élek helyzete pontosan és megbízhatóan meghatározható a kutya körvonalánál, ha elegendő a színek különbség. Hátrány ugyanakkor, hogy a módszerrel nehéz kezelni az olyan kutyákat, amelyek színében hirtelen átmenetek vannak, melyek a kutya által lefoglalt területen belül újabb éleket hoznak létre. Ez utóbbi előreláthatólag később kiküszöbölhető olyan módon, hogy a keletkező régiókat jellemző színek szerint szűrjük (itt már kevésbé lesz jelentős a színszűrés hibája, mert egy-egy pixel helyett régiók átlagait nézzük). Jelenleg ez még nem készült el, így most foltos kutyák vizsgálatára nincs mód, de lassabb (elnyújtott) színátmeneteket mutató vagy egyszínű kutyáknál a módszer jól működik.

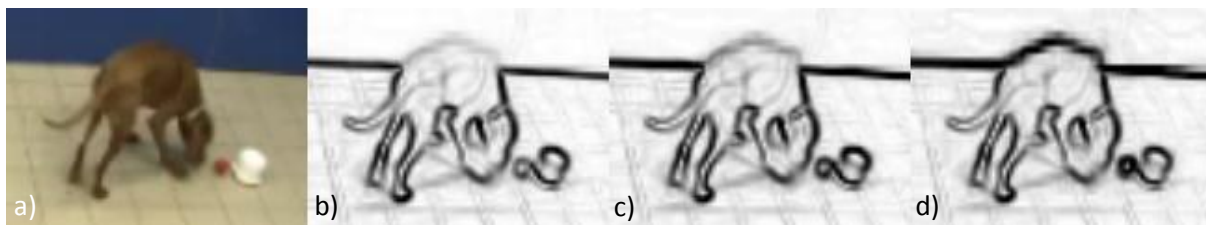
4.3.1 Élkiemelés

A rendszer John F. Canny által leírt [14] élkiemelést valósítja meg. A művelethez a Sobel-operátort alkalmaztam és ezt a műveletet még az előfeldolgozás során, a grafikus kártya segítségével végeztem el. Mivel a videokártyám csak 32 bites színmélységet támogat, így az eddig kihasználatlan alfa csatornát használtam fel az élintenzitás eltárolására. A folyamat során természetesen az alfa blending-et ki kellett kapcsolni. Az élkiemelés HLSL nyelven könnyen implementálható, a hardveres gyorsítás segítségével elkerülhető a CPU terhelés növekedése is. A művelet során a 3x3-as méretű

operátort használtam, amely a célra elegendőnek bizonyult. A szomszédos pixelek kiolvasásához meg kell adni a pixelek méretét a kártya által használt koordináta-rendszerben (ahol a kép bal-felső sarka a (0; 0), a jobb alsó pedig az (1; 1) pont). Az élkimielést beállításoktól függően egy vagy két lépésben végzem el. Az egyszerűbbik esetben csak élkimielést végzek, ugyanakkor bekapcsolható a nem maximális helyek elnyomása is.

A Sobel-operátor a kép intenzitás függvényének gradiensét közelíti úgy, hogy a kép egy adott pixeléből és szomszédjaiból egy mátrixot hoz létre, majd kiszámolja ezen mátrix és a Sobel operátor konvolúcióját. Az eredmény egy érték, ami a Sobel operátor típusától függően - vízszintes vagy függőleges – az adott irányú intenzitásváltozás mértékét adja. Ha ezt a vizsgálatot vízszintes és függőleges irányban is elvégezzük, majd az eredmények négyzetösszegéből gyököt vonunk, megkapjuk az intenzitásváltozás mértékét az összes irányt figyelembe véve. A vízszintes és függőleges értékeket felhasználhatjuk az intenzitásváltozás irányának meghatározására is.

A számítást először fekete-fehér képen végeztem, ez azonban nem túl célszerű, mert a kutya és a háttér között számos esetben csak kis fényességkülönbség van. Így szükség volt a színinformációk hasznosítására is. Az alapot az képezte, hogy a konvolúciót minden csatornára külön el kell végezni (ez egyébként a HLSL-ben szépen megoldható, mert az összes művelet képes vektorokon dolgozni). Az eredmény pedig adódhat a színek szerinti eredmények összevonásából. RGB színtérben az eredmények összegzése (átlagolása) nem célszerű, mert ez azt okozza, hogy a fényesség változás sokkal nagyobb szerepet kap (hiszen akkor az összes csatorna változik), mint a színváltozás. Ha a maximális értéket vesszük eredményül, akkor már jobb eredményeket kapunk. Ezután megnéztem az eredményeket YCbCr színtérben is, mivel itt fényerő és vörös-világosság, kék-világosság színkülönbségi csatornák vannak, ezért a színváltozás nem szorul a háttérbe, ahogy az a következő ábrán látható:

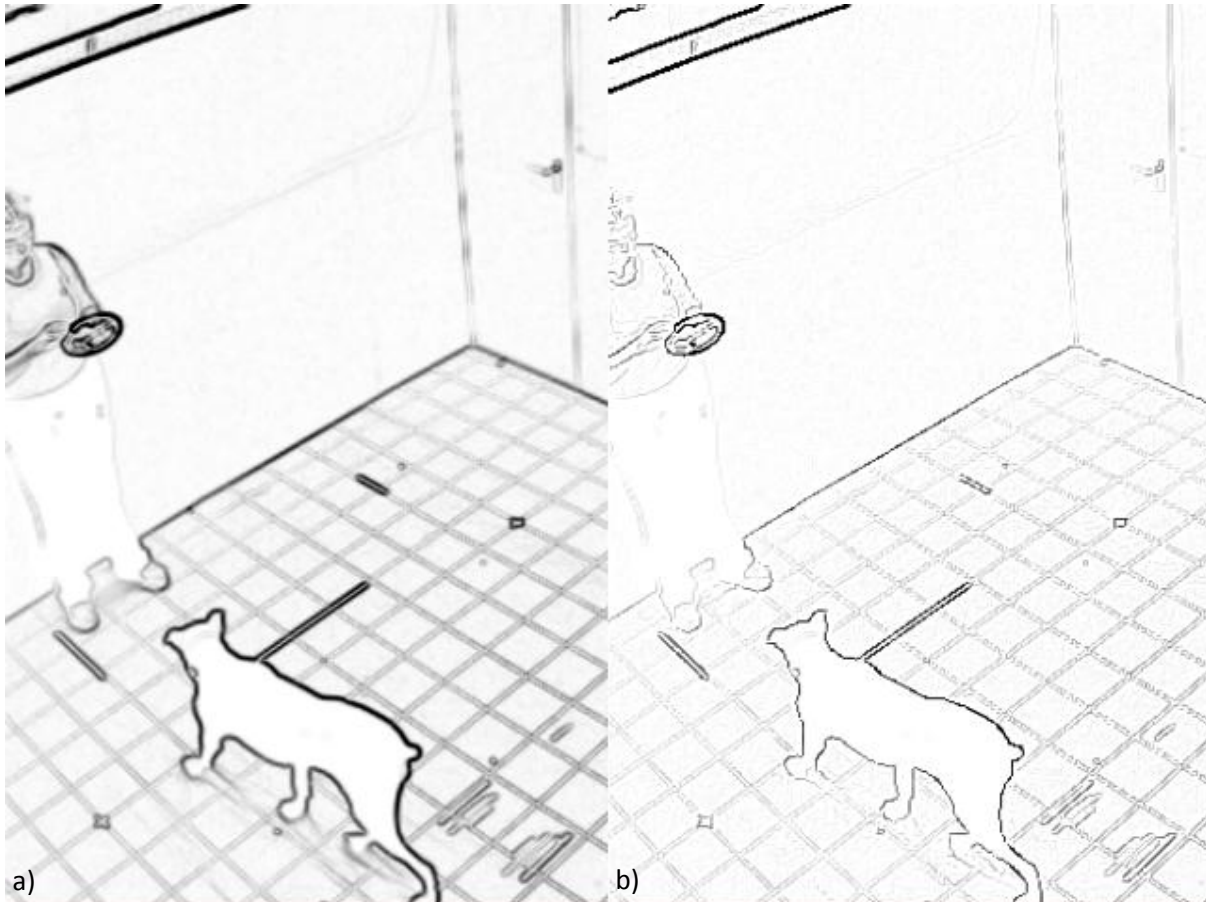


6. ábra A Sobel operátoros élkimielés módszerei: a) Eredeti b) Sobel RGB átlag c) Sobel RGB maximum d) Sobel YCbCr maximum

A rendszer ezen feldolgozási szakaszban RGB képeken dolgozik, ugyanakkor megoldható a színtérkonverzió a Sobel operátor alkalmazása után is a három színtérkomponenshez tartozó részeredményre, ugyanis a transzformáció lineáris. Az élkimielő módszerek többsége alkalmaz zajszűrést a képen, az általam vizsgált videókon azonban a zaj kicsi ahhoz, hogy zavarja az élkeresést. Lehetőség van arra is, hogy az éleket – lineárisan vagy más módon – erősítsük, ilyenkor azonban számítani kell arra, hogy a grafikus kártya a megengedett [0..1] intervallumon esetlegesen túllógó értékeket csonkolja.

A rendszerbe beépítettem a Canny által leírt nem-maximum elnyomást is. A vízszintes és függőleges operátorokkal vett konvolúcióból meghatározható értékek arkusztangense megadja az intenzitásváltozás irányát, ha ebben az irányban a szomszédos pixeleken az élintenzitás kisebb, mint a vizsgált pixelen, akkor az adott pixelt meghagyjuk, különben töröljük. Canny módszere vízszintes,

függőleges és átlós irányokat enged csak meg, én viszont - felhasználva, hogy a grafikus kártya a mintavételezés során interpolál - tetszőleges (nem egész pixel) helyen is lekérhetem az élintenzitást, így a vizsgált irány tetszőleges lehet. Ebben az esetben az élkiemelés két lépésből áll: az első lépésben meghatározom az élek intenzitását, illetve a vízszintes és függőleges irányú komponensét a gradiensnek, és az eredményeket eltárolom egy külön textúrában. A második lépésben az így meghatározott adatokból már megkaphatóak a meghagyott és eldobott pixelek:



7. ábra A nem-maximum elnyomás, a) az eredeti, b) a kezelt kép

A módszer hátránya, hogy olyan videókon, amelyek tömörítésre kerültek, jelentősen romlik az eredmények minősége, ugyanis a legtöbb tömörítő a képet kis téglalap alakú területekre, blokkokra osztja, ezen területeken belül a pixelek színét egy adott értéktől való különbséggel írja le (így a színinformáció kisebb helyen tárolható). Ez azt jelenti, hogy adott blokkon belül a pixelek által felvehető színek száma korlátozott lesz, amely az intenzitásfüggvényben szakadásokat hoz létre, ez pedig az élek széttöredezéséhez, szakadásokhoz vezet az eredményben. Tömörítetlen videók esetében ez a jelenség természetesen nem jelentkezik. A különböző zajelnyomó, a képet puhábbá tevő szűrők egyébként könnyen megírhatók HLSL nyelven.

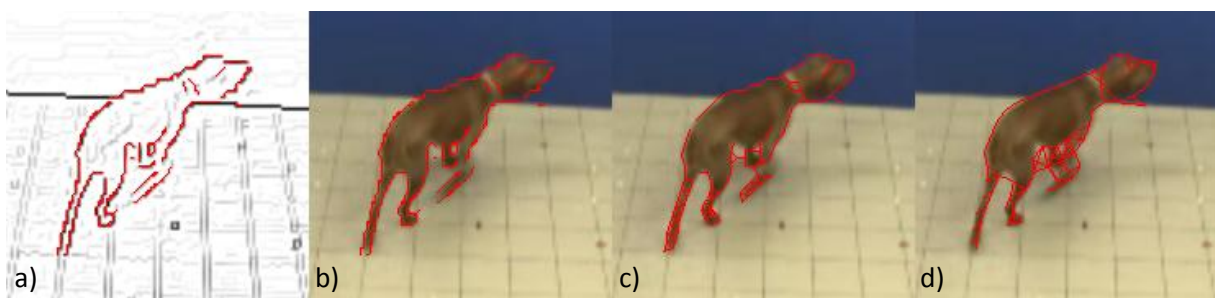
4.3.2 Élfeldolgozás élkövetéssel

Az első módszer, amelyet az élek feldolgozásához implementáltam, úgy kívánja az éleket lekövetni, ahogyan azt egy ember tenné, ha megkérnének, hogy húzza végig az ujját egy görbe mentén. A rendszert úgy készítettem el, hogy az az előző részben bemutatott maximum elnyomás nélkül is működjön, mivel az etológusok kísérleteit későbbi elemzés céljából videón rögzítik, amelyeket tömöríteni célszerű. Ennek eredményeként azonban kezelni kell tudni a vastag éleket is.

Az algoritmus első lépése, hogy a kutyaferismerő modul által megjelölt objektum területét megvizsgálja, itt olyan pontokat keresve, ahol nagy az élintenzitás. Ezekből a pontokból kiindulva ezután végighalad a képen található éleken.

Ahhoz, hogy az élek követése hatékony legyen ügyeltem arra, hogy ahol lehetséges pointerműveleteket használjak, így jelentősen (többszörösére) növelhető a futási sebesség, ugyanakkor a programkód sokkal kevésbé lesz átlátható, illetve a kép memóriaterületéről való kifutás veszélye is fenyeget. Az első lépés egy a navigációt gyorsító tömb generálása volt, amely adott sugarú körben lévő pixelek relatív pozícióját tárolja a jelenlegi – a kör középpontját adó – pixelhez képest. Továbbá szükség volt egy másik tömbre is, mely a körvonalba befoglalt pixelek relatív pozícióit tárolta, ez később a már bejárt területek megjelölését végrehajtó művelethez kellett. Fontos, hogy a két tömb nem tartalmazhat közös elemeket. Szükség van egy olyan tömbre is, amely a feldolgozandó videóval egyező felbontású, és tárolja a már felderített területeket. A térkép egész szám alapú, ahol a tárolt szám az adott pixelhez rendelt vertex indexét tárolja, vagy a terület felderítetlenségét / tiltottságát mutatja meg. Alaphelyzetben tiltott minden olyan pixel, amely a kép széléhez egy keresési sugárnál kisebb távolságra van, vagy nem része a célobjektumhoz tartozó képterületnek. A térképet minden képkocka feldolgozása előtt alaphelyzetbe kell állítani. Az eredményt tulajdonképpen egy gráf jelenti, amelyet a csúcsok listája képez le. A csúcsok osztálya a pozíció és azonosító mellett tárolja a kapcsolódó elemek tömbjét is (az osztályoknál ez persze csak referenciátárolást jelent).

A rendszer a megfelelő határértéknél nagyobb élintenzitású pixelekről elindítva kezdi meg az élek felderítését. Amikor egy adott pixelen állunk ellenőrizni kell az adott távolságra lévő körben elhelyezkedő szomszédok intenzitását. A távolság lehetne egy is, de gyakorlatban jelentősen nagyobb sebesség érhető el akkor, ha egyszerre 2-6 pixelt is lépünk, a felbontástól és a keresett alakzat méreteitől, komplexitásától függően. Természetesen, ha a távolságot nagy értékre állítjuk, akkor átugorhatunk a közelben lévő élekre, amelyekkel nincs valódi összeköttetés. A mostani feladat esetében mivel a hát vonalát követjük, így nagyobb távolság is megadható, amely ugyan a lábak környékén hibákat eredményez, de ez nem zavarja a hát helyének meghatározását.



8. ábra Az élkövető algoritmus kimenete: a) kimenet háttérben az élintenzitással b) élkimenet 1-es keresési sugárral c) élkimenet 3-as keresési sugárral d) élkimenet 5-ös keresési sugárral

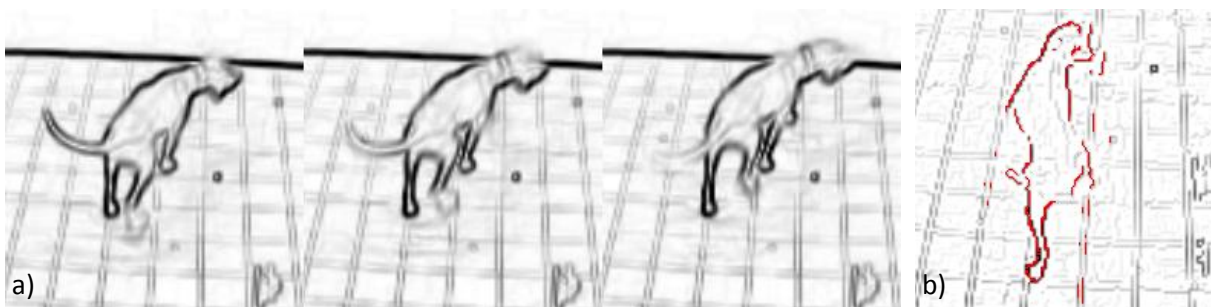
A kör mentén elhelyezkedő pixelek helyét úgy kapjuk meg, hogy a jelenlegi pixel mutatójához hozzáadjuk a kör indextömb elemeit. A rendszer fel van készítve arra is, ha egymás mellett több egyező intenzitású pixel van a körben, ilyenkor a középutat választja a program. Ilyen „vastag” élek úgy jöhetnek létre, ha egymás mellett lévő pixeleken az intenzitásváltozás mértéke egyenlő. Ennek különösen nagy az esélye, ha az éleket erősítjük, és beleütközünk a maximális intenzitásértékbe. Miközben körben haladunk összehasonlítjuk a jelenlegi pixel intenzitását az előtte lévővel: ha

nagyobb, akkor feljegyezzük, hogy itt csúcsérték kezdődik, ha kisebb pedig azt, hogy csúcsérték végződik. Ha maximum első és utolsó indexe rendelkezésre áll, akkor helye meghatározható. A rendszer jelenleg így két maximum értéket keres, melyek közül egyik célszerűen az a pixel ahonnan jöttünk, a másik pedig ahova menni kellene. A rendszer ezután megvizsgálja a célt, ilyenkor három dolog történhet:

- Foglalt: kiolvassuk a térképből az ide tartozó vertex indexét, és ahhoz kapcsoljuk a jelenlegit
- Tiltott: nem csinálunk semmit
- Szabad: létrehozuk az új csomópontot, amelyet kölcsönösen összekapcsolunk a szülővel, majd hozzáadjuk az csúcsok listájához

A lépések során folyamatosan feltöltjük a térképet a kitöltő indextömb használatával. Így nem fogunk körbe-körbe járni.

A csúcsok listája két módon épül: egyrészt a kiindulópontok hozzáadásával, másrészt az éleken való végighaladással. Az így keletkezett gráf leírja az élek geometriai helyzetét és összeköttetéseit, és készen áll a további feldolgozásra. Sajnos a rendszer valós tesztje során ezzel a módszerrel több nehézség jelentkezett, az egyik a részeredmények számos esetben igen nehéz értelmezhetősége, ennek oka, hogy egy hurkokkal és elágazásokkal teli gráfban kellene kiválasztani a kutya hátának vonalát. A kutya ugyanis az esetenként élekkel rendelkező környezettel érintkezve hamis – nem a kutyaéhoz tartozó – éleket is összeszed. További probléma, hogy számos esetben a kutya körül nem keletkezik zárt körvonal, amelynek oka lehet a kutya képről való lelógása vagy a motion blur (utóbbi a videó tömörítés, vagy a túl hosszú expozíciós idő eredménye). A harmadik nehézség, hogy esetenként a kutyán belül is keletkeznek élek, amelyek értelmezése nehézkes lehet. A rendszer elkészítése során a szakirodalmat tanulmányozva eszembe jutott egy olyan módszer, amely ugyan több számítási műveletet igényel, de közvetlenül eredményre vezet, ráadásul lekezelhető vele a motion blur által okozott élmegszakadások, és a képről lelógó kutya esete is.



9. ábra A motion blur hatásai: a) a kutya farka mozgás közben „eltűnik” b) a kutya gyorsan mozgó lába miatt a hátárvonalból hiányzik egy szakasz

4.3.3 Élfeldolgozás fékezett vektorgrafikus elárasztással

Az első módszer után annak gyengeségeiből is tanulva, olyan módszert kívántam kidolgozni, amely képes a kutya körvonalát visszaadni, akkor is, ha az élekben szakadások vannak. Ha a módszer eredménye egy zárt töröttvonal, az nagyon kedvező a további feldolgozás szempontjából, hiszen abból a gerinc és a lábak helyzete is jól látható.

Ezen módszer leginkább az árasztásos kitöltésre hasonlít, bár számos eltéréssel is rendelkezik attól. A legfontosabb különbség, hogy míg árasztásos kitöltésnél raszteresen dolgozunk, ennél a

módszernél vektorgrafikusan közelítjük meg a problémát. Erre azért van szükség, mert a határvonal lyukain való kifolyást így lehet a legkönnyebben elkerülni, felismerni. A módszer abból indul ki, hogy ismerjük a kutya képen elfoglalt területének egy belső pontját. Ebbe a pontba elhelyezünk egy magot, egy szabályos sokszöget, melynek csúcsai sugárirányban mozognak kifelé. Minden csúcs rendelkezik egy energiaszinttel, amelyet az éleken való áthaladás csökkent, míg az él nélküli területeken való haladás feltölt. Ha az energia elfogy, akkor az adott csúcs megáll. A rendszer folyamatosan nézi az egyes csúcsok távolságát, ha egy él túl hosszú, akkor annak felezőpontjára új csúcsot szúr be. Ha a felosztandó él átmetsz egy igazi élet, akkor ezt a részt befagyasztjuk, rögzítjük (ezzel a módszerrel nem lehetne olyan jól csillagszerű alakzatokat feltölteni, de itt nem erre van szükség). Az új csúcs mozgásiránya a két szülő mozgásirányának átlaga, kivéve, ha ez nagyon kis szöget zár be a felosztandó éllel, mert akkor ezen éllel merőleges, kifelé mutató. Ha szakadás van a körvonalon, akkor két dolog történhet: ha a közelben van él, akkor addig terjed ki a terület, ha nincs, akkor a mozgásdetektor által kijelölt nem mozgó képrész elérése a csúcsokat mindkét irányban töröljük az első már rögzített pontig, így levágva a felesleges részt. Előfordulhat még, hogy a kutya lábainál sziget keletkezik, így a rendszert meg kell gátolni az önmagába való visszaterjedésben. Ezért az alakzat önmetszését ellenőrizni kell, ezt én úgy végeztem el, hogy egy alacsony felbontású vászonra felrajzolom az éleket raszteresen és közben nézem, nem rajzolok-e már foglalt területre. A metsző és szomszédos élek csúcsai leállításra kerülnek. A számítás tehát három fő lépésből áll:

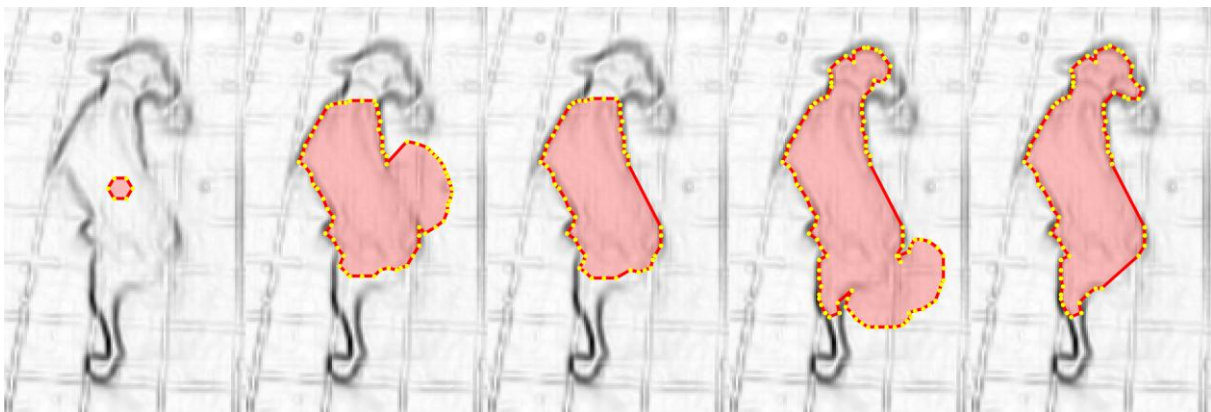
- Csúcsok mozgatása
- Távol lévő csúcsok közé új csúcs beillesztése
- Önmetszés ellenőrzés

Ezeket a lépéseket addig ismételjük, míg be nem áll egy állandósult állapot. Ez a módszer elég robosztus működésű, ugyanakkor számításigénye is jelentősen nagyobb. Hogy növeljem a sebességet, több optimalizációs lehetőséget kerestem:

- Az élek maximális hossza növelhető, így kevesebb csúcs lesz, hátrány azonban, hogy ilyenkor a kutya lábait gyakran levágja a rendszer.
- Az ütközésellenőrzést nem végezzük el minden ütemben, hanem csak kb. minden 15-50. lépésben, mivel a rajzolás elég gépigenyes, ezért ez nagy sebességnövekedéssel jár.
- A már megállt éleken nem végzünk műveleteket.

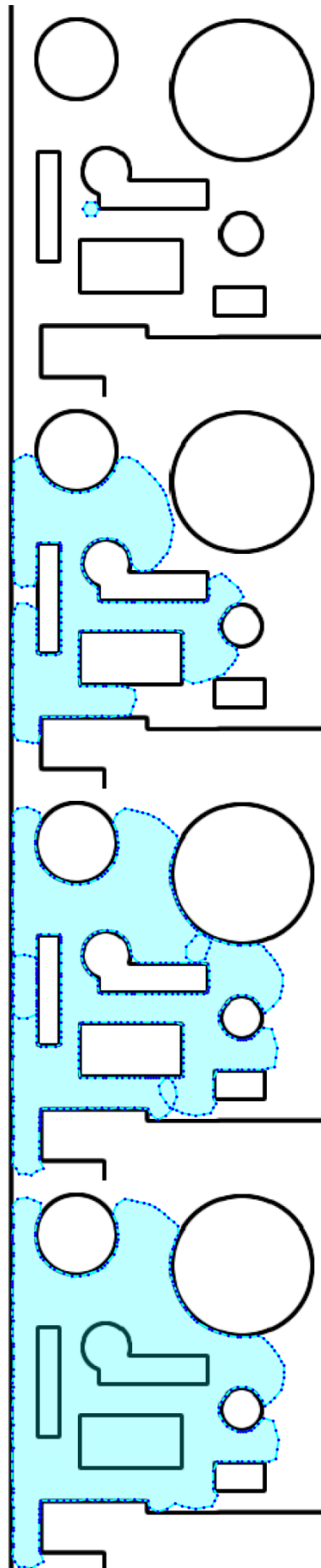
A csúcsok tárolása kettős láncolású listában történik, amelynek végei össze vannak kapcsolva, így könnyen hozzáadható, vagy eltávolítható egy elem. Ehhez létrehoztam egy új, generikus osztályt, amely kettős láncolású, végtelen lista tárolására alkalmas. A C# ugyan nem rendelkezik teljes körű pointertámogatással (ami van az elsősorban tömbbejárásra és más programokkal való együttműködéshez használható), de a referencia típusú változók gyakorlatilag implicit pointerként használhatók. A listát felszereltem az elemek hozzáadásának, beillesztésének, törlésének képességével. Eközben természetesen a kapcsolatok mindig megfelelően kerülnek módosításra. A rendszer rendelkezik egy speciális összekapcsoló művelettel is, amellyel a nem szomszédos elemek összekapcsolhatók úgy, hogy a közöttük lévőket töröljük. A listát elláttam egy indexszel is, amely az egyes csúcsok memóriacímeit adja meg azok azonosítója után (az egymás melletti csúcsok nem feltétlen rendelkeznek egymás melletti azonosítókkal, hiszen új elemeket szúrunk be a meglévők közé). Így a sorszám segítségével az elemek $O(1)$ idő alatt elérhetőek. A lista méretét az index miatt előre meg kell adni, de telítődés esetén az index mérete automatikusan duplázásra kerül.

Mivel az elemekkel tetszőleges irányú mozgást kellene megvalósítani, ezért fontos volt annak megoldása, hogy ne kelljen lebegőpontos koordinátákat használni, hanem minden egészműveletekkel működjön. Ezt a következőképpen valósítottam meg: a rendszer a mozgási sebességet x és y irányban egészre kerekíti, majd kiszámolja, hogy mekkora az egy lépéssel keletkező hiba, ebből kiszámolható hány lépésenként lesz az eltérés pontosan egy. Ezután a lépéseket számlálva korrigáló lépéseket iktat be a rendszer: például ha túl gyorsan megyünk, adott időközönként egyet visszalép. Mivel az a lépésszám, ami a korrekciók között van, nem egész, ezért azt még egészre kell alakítani. Ezt virtuális lépésszámnak neveztem el, mert egy lépés nem egy egységet ér. A korrigálási intervallumot az int típus maximális értékének felére veszem (mert bele kell férjen az int értékkészletébe mikor túllépünk rajta, mielőtt korrigálni kell), és ezek után a lépésszámlálóhoz lépésenként adott értéket arányosan növelem (kerekítve). Így itt már csak egészek kelljenek. Még ki kell számolni a kezdeti eltérést is, illetve az ezzel arányos virtuális lépésszámot, és erről kell indítani a lépésszámláló értékét, figyelembe véve a korrekció és a kezdeti pozíció eltérés irányának viszonyát. Ez azt jelenti, hogy egy új elem létrehozása után már csak egész műveletekre (összeadásokra, kivonásokra, hasonlításokra) van szükség, amely jelentősen növeli a sebességet. A mozgás során a rendszer az adott pozícióban lévő élintenzitást lekérdezi, és ennek megfelelően csökkenti, vagy növeli az adott csomópont energiaszintjét. Ezzel figyelembe vehető az él vastagsága, hiszen több pixelen áthaladva egy kevésbé erős él is felhasználhatja az összes energiát. Ha egy csomópont energiája negatív vagy nulla lesz, akkor annak mozgása véget ér. A rendszer megállítja a pontokat a kép határának elérése esetén is. Amennyiben az él eléri a mozgás detektor által kijelölt terület határát, akkor az adott élszakasz visszavágásra kerül egészen az első már megállt csúcsig, így a további növekedés meggátolható. Ha lyuk van az él között, akkor a körvonal vagy megáll a környezetben lévő élknél, vagy az előbbi módon levágásra kerül (ez a jobbik eset, mert ilyenkor nem sérül erősen a körvonal).



10. ábra Hiányos él kezelés

Az egyes él hosszának vizsgálatokor csak az egész értékeket használja a rendszer, és négyzetösszeget vizsgál. Mivel az él hossza egynél nagyobb is lehet (általában 3-5 pixel szélességet alkalmaztam), ezért a rajz kisebb részét kell bejárni, mint például árasztásos kitöltés esetén, a módszer további előnye, hogy kevesebb listaművelet is kell hozzá. Ha egy él hossza az előre megadott maximumértéket meghaladja, akkor új csomópont kerül besúrára. Az új csomópontok besúrása során a rendszer megvizsgálja, hogy van-e él a képen a kettévágandó szakasz helyén, ha van, akkor az új csomópont energiaszintje nulla lesz, ellenkező esetben a standard energiaszintet kapja. A szülő vertexek energiaszintjét figyelembe véve három eset lehetséges:



- **Mindkettő energiaszintje nulla:** az új vertexé is nulla lesz, ahogy a sebessége is

- **Egyik szülő energiaszintje sem nulla:** az új vertex a szülők energiájának átlagával indul. Mozgásiránya a következőképpen alapul: kiszámítjuk a szülők sebességvektorainak összegét, és a közöttük lévő él irányába mutató vektort, mindkettőt normalizáljuk, majd a skaláris szorzatukat összehasonlítjuk egy előre megadott határértékkel:

- Ha a határértéknél kisebb az eredmény, akkor a két vektor nagy szöget zár be, ilyenkor az új vertex sebessége az első irányvektor lesz.

- Ha a határértéknél nagyobb az eredmény, akkor az oldal irányvektorával merőleges, kifelé mutató, egységnyi sebességet választunk. Mivel az él iránya megadja, melyik oldalon van a külső rész, így könnyen (a koordináták cseréje, egyik negálása segítségével) kiszámítható az új irány. E lépés azért szükséges, mert az élekkel párhuzamosan mozgó csúcsok egyrészt feleslegesek, másrészt potenciális hibalehetőség szolgáltatók.

- **Csak az egyik szülő energiaszintje nulla:** ilyenkor az élre merőleges, kifelé mutató, egységnyi sebességet választok. Ez azért célszerű, mert ez tipikusan akkor fordul elő, ha egy él éppen hozzáillesztődik egy falhoz (ilyenkor gyorsítja a rés eltűnését), vagy lyukon való átfolyás esetén (lehetővé teszi az oldalirányú terjeszkedést). Az energiaszint a nem nulla energiaszintű szülőtől származik.

Fontos eleme a működésnek az ütközésellenőrzés, amellyel felismerhetők azok az esetek, amelyeknél a rendszer visszaömlik ugyanarra a helyre. Erre akkor kerülhet sor, amikor egy szigetet körülölelnek az élek, majd találkoznak a túlsó oldalon. Az ütközéskeresésnél gyakorlatban rajzolás történik. Ugyan rendelkezésre állnak hatékony metsző algoritmusok (pl.: sweep line algorithm [17]), de raszteres megoldással az ezen algoritmusokhoz köthető keresések, rendezések és az intenzív listahasználat elkerülhető. Hogy a rajzolás gyors legyen, megvalósítottam Bresenham vonalrajzoló algoritmusát [6], amely nagyon hatékony, ugyanis kizárólag egész kivonásokat, összeadásokat, bit shiftelést és összehasonlításokat tartalmaz, nincsenek benne osztások, szorzások, és lebegőpontos műveletek. Működése nagyon hasonló a már bemutatott mozgató algoritmuséhoz, így itt most nem ismertetném. A rajzolás egy olyan tömbbe történik, amely színek helyett az adott élhez tartozó kezdővertex sorszámát tartalmazza. Amikor végigmegy a program az éleken, folyamatosan ellenőrzi,

11. ábra A vektorgrafikus árasztás kimenete egy tesztképen

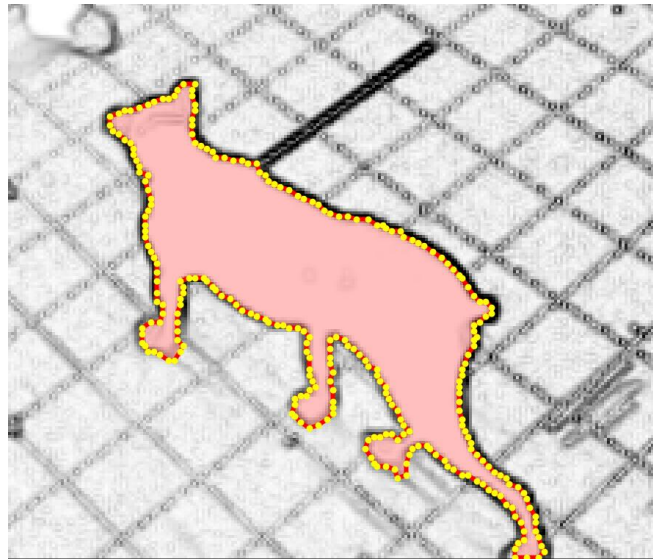
hogy a soron lévő pixel foglalt-e már. Foglaltság esetén a rendszer a sorszám

alapján ellenőrzi, hogy ez az előző szakasz vége-e, ha nem, akkor feljegyzí az ütközés tényét, és szembenálló elem indexét. A szakasz rajzolása után az érintett élekhez tartozó csúcsok megállításra kerülnek, így a további ömlés nem lehetséges. Ez a módszer bár működik, három nagy hátránya van: egyrészt a szigetek félsziget formájában megmaradnak, másrészt sokszor szükséges a szomszédos élek megállítása is mivel máskülönbén egy-egy csomópont „elszabadul”, de a legnagyobb probléma, hogy a rajzolat minden iterációban el kell végezni.

Hogy az eredmények szebbek legyenek, gyorsabb és stabilabb legyen a módszer, kidolgoztam egy továbbfejlesztett megoldást. Az új módszer az ütközések helye alapján kijavítja a határoló szakaszsereget tartalmazó körkörös listát, és kiköti a felesleges részeket a gráfól. Ennek előnye, hogy az eredmények sokkal jobban értékelhetők, de még ennél is fontosabb, hogy ezt a módszert nem kell minden lépésnél elvégezni. Az adott szakasz kirajzolása után a rendszer egy indextömböt épít, amely tartalmazza adott indexről, hányas számú elemre lehet ugrani (kölcsonösen). A metszés során előfordulhat, hogy egy szakasz többet is elmetsz, ekkor mindig a legutolsó metszés ténye marad az ütközési tömbben, a régi összeköttetés törlésre kerül (ennek okáról később). A rendszer egy a kép szélességével egyező hosszúságú logikai tömbbe feljegyzí, hol történtek metszések, erre a következő lépésben lesz szükség. Hozzárendelek sorszámokat is a csúcsokhoz, amelyek már sorrendben követik egymást az élek mentén. Ha metszéspontok ismertek, akkor a következőt csinálom:

- Keresek egy kezdőpontot, amely biztosan megmarad, tehát külső élhez tartozik.
- Elindulok a kezdőpontból és végighaladok az egymást követő éleken közben ellenőrizve, hogy van-e metszés az adott helyen, ha van, akkor megnézem, hogy az elérhető csúcs sorszáma nagyobb-e az aktuálisénál (ez olyan esetekben fontos, ha több él halad egy helyen, egymáson), és ha igen, akkor átugrok arra az élre, és kapcsolatot hozok létre az eredeti pozícióval, ezzel a köztes csúcsok kikötésre kerülnek a láncból. Ez azért lesz mindig jó döntés, mert ha egy él metsz egy másikat, akkor az adott él „alábukik” a másiknak, így már nem lehet többé külső él. Az új élen pedig azért kell a „menetirányban” - azaz a csúcsok felírásának irányában - haladni, mert a terjedés módja miatt nem keletkeznek olyan alakzatok, mint például egy végtelen jel, melynek lenne olyan éle, amin a lista irányával ellentétesen kellene haladni ahhoz, hogy a határvonalat megkapjuk.
- Ha ezen a módon visszaérek az eredeti pozícióra, akkor a folyamat véget ér.

Ezzel a módszerrel megszüntethető az összes visszaáradás miatt létrejövő hiba, és folytatható az iteráció. A művelet elvégzését elég 10-50 lépésenként ismételni, figyelembe véve, hogy a keletkező plusz csomópontok is erőforrást használnak. Miután már nincs változás, vagy elértük az iterációs limitet, az utolsó körben még egyszer lefut ez az algoritmus. A kezdőpont kiválasztását úgy végzem, hogy megkeresem a leghosszabb olyan szakaszt vízszintes irányban, amelyre merőlegesen van él, de nincs metsző él. Ennek felezőpontját veszem, és addig megyek lefele a kutya burkoló téglalapjában az élek rajzán, amíg bele nem ütközök egy élbe, ami a kezdőél lesz. Ez azért fontos, mert ha nem külső élből indulok, akkor rossz eredményt kapok. Olyan helyet kell keresni, ahol nincs metszés, mert különben ha egymáson halad több él, akkor a legutoljára kirajzoltat tudjuk elkapni, ez viszont általában kivágandó élszakaszhoz tartozik.

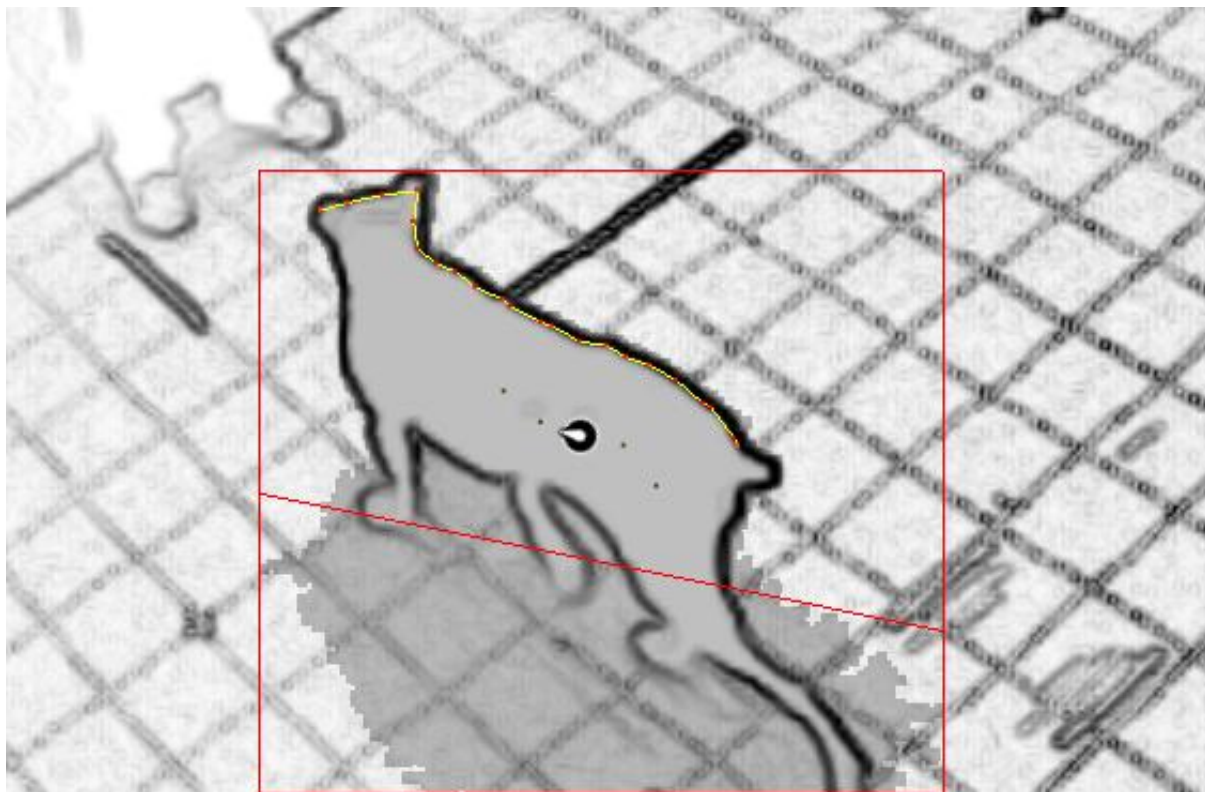


12. ábra A kutya körvonalának kiválasztása

Az eddig bemutatott műveletek függetlenek a kutyától, annak pozíciójától, azonban a farok-hát-fej vonal meghatározása, illetve a kezdőpont kijelölése már nem. Jelenleg a kutya gerincének helyzetéhez azokat a képeket tudom felhasználni, ahol a gerinc és a kép vízszintes tengelye max. 60 fokos szöget zár be. Ez – tekintve, hogy a laborban 8 kamera van – bőven elegendő a pozíció meghatározásához. Ilyenkor a kutya felső határolóvonalának egy szakaszát keressük, ez lesz a gerinc látszólagos helye.

A módszer működésének előfeltétele, hogy ismerjük a keresett síkidom egy belső pontját (természetesen lehetséges lenne több mag elhelyezésével a kép felderítése, de ez nem lenne erőforrástakarékos megoldás). A kezdőpontot legkönnyebb akkor kijelölni, ha a kutya éppen mozog, ilyenkor ugyanis feltételezhetjük, hogy a kutya gerince a mozgás irányába néz. Ebben az esetben a mozgás irányára merőleges szakaszokkal elmetszhetjük a kutyát, hogy megkeressük az intenzitásmaximumokat. A maximumok közül kiválaszthatjuk fentről lefelé haladva az első elég nagy távolságra lévő párt, vagy például a legtávolabb esőket. Ez a módszer sok esetben jól működik, azonban a kutya lábai, az élek megszakadása, illetve a külső élek (amelyek nem a kutyához tartoznak, de a mozgásdetektor árnyék miatt tévesen a mozgó területhez sorolja őket) megzavarhatják, ezért a gyakorlatban nem egy, hanem több (jelenleg öt) ilyen vonalon keresem a kiindulópont megfelelő helyét. Ezek átlagából kapom a kiindulópontot. A rendszer a kutya által elfoglalt terület tömegközéppontját folyamatosan követi a mozgásirány meghatározásához. Mivel ennek sebessége a zaj miatt nehezen kezelhető, ezért a rendszer egyrészt csak egy minimális távolságú elmozdulás felett számol pozíciókülönbséget, továbbá az így kapott sebességeket átlagolja. Így ugyan egy kis holtidővel fog rendelkezni a sebesség irányával kapcsolatban a rendszer, de ez elfogadható, mert nem akadályozza a kiindulópont keresést. Ha megvan a kutya mozgási iránya, akkor a rendszer ilyen irányú egyenest fektet a tömegközéppontra és elmetszi ezt a kutya határoló téglalapjával. A kapott szakaszon elhelyez adott számú pontot, úgy hogy a köztük lévő távolságok nagysága a szakasz hosszához képest megadható. Lehetőség van arra is, hogy a pontokat a mozgás irányában a kutya sebességét figyelembe véve eltoljuk, ez elsősorban hosszú farokkal rendelkező kutyafajták esetén lehet szükséges. Az így kapott pontokban merőlegest állítok a sebességre, és a kapott egyenesek burkolótéglalapba eső részén végzem a már ismertetett keresést. Ha a kutya megáll, vagy például leül/fekszik, akkor a tömegközéppont kis elmozdulása biztosítja, hogy a már kiválasztott keresési

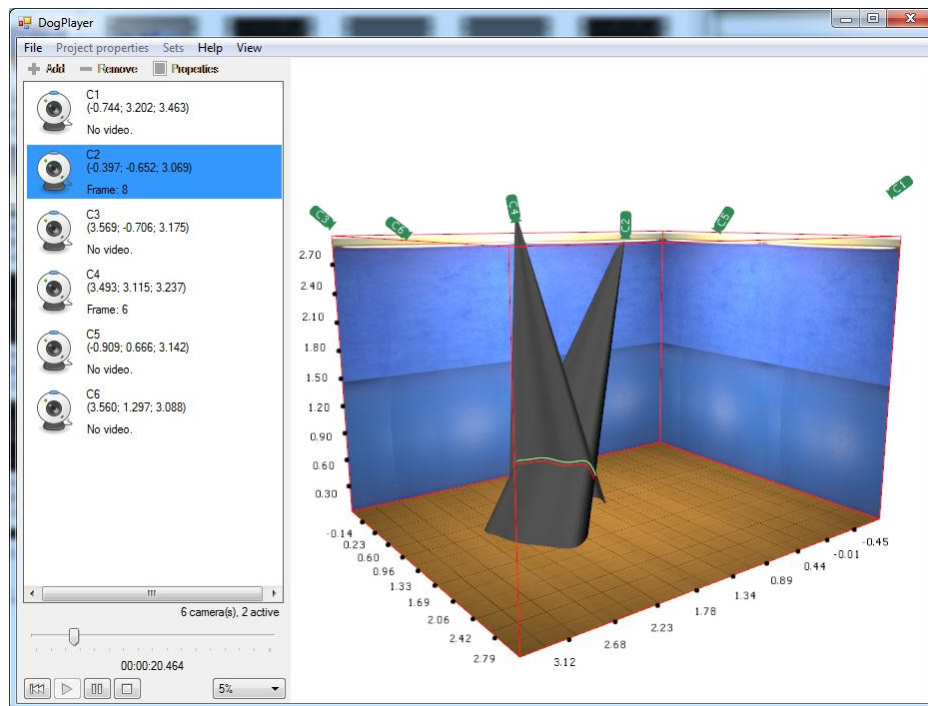
irány megmaradjon, a rendszer méri az egy helyben tartózkodás idejét, és ha ez elég hosszú akkor a következő elmozdulást nem veszi figyelembe, így ezen helyzetek kezelhetők. Még nincs megvalósítva, de adja magát a lehetőség, hogy ha egyszer már találtunk egy jó kiindulópontot, és megkaptuk a gerinc vonalát, akkor ezt felhasználva és összevetve a mozgás irányával, nagyságával (tömegközéppont), kikövetkeztethetjük a tömegközéppont új helyét, majd ezt ismételhetjük, így sok időt takaríthatunk meg. A rendszer akkor is képes a kiindulópont helyét meghatározni, ha a kutya lelóg a képről, ilyenkor a képszélét virtuális élnek képzelve hajtom végre a keresést.



13. ábra A kutya gerincének vonala kiválasztva a képből, jelölések: piros keret - a mozgó képrész befoglaló téglalapja, szürkével festett rész – a mozgásdetektor eredménye, nyíl - a kiválasztott kiindulópont, nyíl melletti szürke pontok - kiindulópont jelöltek, sárga vonal - hát vonala, piros pontok - a töröttvonal csomópontjai, piros vonal közepén – mozgás iránya

A módszernek ezután meg kell határozni a hát vonalát, illetve később a lábak stb. kezelése is szóba jöhet. Ez egyszerűen úgy történik, hogy az élek rajzát felhasználva, megkeressük a felső éleket, azok közül is a bal és jobb szélsőt. Ezután az élek között végigmenve folyamatosan nézzük, hogy változik-e vízszintes irányban az élek iránya, az ilyen mozgásirányváltások közötti töröttvonalakból kiválasztjuk a leghosszabbat. Ez utóbbi módszerre azért van szükség, hogy a kutya lábait ne vegyük bele a gerincvonalba. Mivel néha előfordul, hogy több csúcs jut egy vízszintes pozícióra (a kis maximális élhossz, függőleges él, vagy a rasterizáció miatt), ezért a rendszer az irányváltások keresésekor hiszterézissel működik: adott irányba fordulás után több ellentétes irányú él kell az irányváltás elfogadásához. Ha a gerinc pontjai megvannak, a rendszer létrehozza a megfelelő objektumokat a képkocka objektumlistájában, ezeket megfelelő címkével is ellátja, hogy a feldolgozó program könnyedén kiválaszthassa.

4.4 A gerinc háromdimenziós pozíciójának meghatározása



14. ábra A gerinc háromdimenziós pozíciójának meghatározása

4.4.1 Bevezetés

A gerinc kétdimenziós megkeresése után a kiszámított töröttvonalat a rendszer hozzáadja az adott képkockán lévő mozgó objektumok listájához címkével megjelölve. Ezután az adattovábbító modul az adatokat a lemezre menti, ahonnan egy különálló programmal visszaolvashatóak. Ez a program képes a különböző kamerákkal felvett adatok összevetésére, és belőlük a háromdimenziós pozíciók megállapítására.

Ahhoz, hogy ez lehetséges legyen először egy környezetet kellett kialakítanom a feldolgozáshoz. Az összedolgozó programban létrehozhatóak projektek, amelyekben a munkánkat végezzük. A projekt tartalmazza a mérőlabor tulajdonságait (méretek, koordináta-rendszer origója), a kamerák tulajdonságait (transzformációs mátrixok) és a kamerához társított mérési eredményeket tartalmazó fájlok elérési útjait. A rendszer biztosítja, hogy egy kamerához több fájlt rendeljünk, ez esetben a különböző videók felhasználhatók egymás utáni időintervallumokhoz, illetve a videók között csoportok is megadhatók, amelyek eltérő mérések egy projektben lévő tárolását teszik lehetővé. Ez utóbbi azért hasznos, mert ugyanazon beállítású kamerákkal gyakran végzünk több mérést egymás után, és így nem kell többször megadni a kamerák beállítását.

A kamerák pozícióját leíró transzformációs mátrixokat a mérőlaborból megkaptam, így a kamerák kalibrálásával nem kellett külön foglalkoznom. A kamerák kalibrálási eredményei tartalmaztak két transzformációs mátrixot, amelyekből egyik a kamerához rögzített és a világ háromdimenziós koordináta-rendszere közt transzformál (extrinsic paraméterek mátrixa), míg a másik a kamera vetítését írja le (intrinsic paraméterek mátrixa). A mérés során e mátrixok inverzére van szükség, hogy megkapjuk a kép pontjaihoz tartozó kamerából induló irányvektorokat. A kalibrációs eredmények megadják a kép torzításait is, így azok megadhatóak az előfeldolgozóban a torzulások kiküszöböléséhez.

A kamerákhoz tartozó a gericvonal csomópontjaihoz rendelt irányvektorok közé háromszögeket kiegészítve legyezőszerű héjszerkezetek kaphatóak. Ha két ilyen legyezőt egymással elmetszünk, akkor egy háromdimenziós töröttvonalat kapunk eredményül, amely közelíti a gerinc formáját. A töröttvonal azonban a kamerák és kutya helyzetétől függően többé-kevésbé a kutya felett fog lebegni. Ennek korrigálására csak a kutya szélességének ismeretében van lehetőség, ilyenkor a kutyát hengerekből összerakott testtel közelítjük.

A következő alfejezetek bemutatják, hogy miként határozható meg a gerinc háromdimenziós pozíciója a kamerák kétdimenziós mérési eredményeinek összevetésével.

4.4.2 Interpoláció

Mivel a dolgozat megírása idején nem volt lehetőség a kamerák szinkronizált működtetésére, ezért ki kellett dolgozni egy módszert a pontok időbeli interpolációjára. Bár ez a módszer a triggerelhetőség lehetőségét figyelembe véve nem ajánlott, elkészítése még is hasznos lehet, hiszen így nemtriggerelhető kamerák is használhatóak. A művelet során interpolálni kell héjmetszet kiszámításához használt töröttvonalakat, amelyek a kutya hátát leképezik. Ezzel azonban van egy nagy probléma, nevezetesen, hogy ezek a vonalak az egymás utáni képkockákon eltérő számú csomóponttal rendelkeznek. Ahhoz, hogy közöttük időben megoldható legyen az interpoláció, azonos csomópontszámra van szükség. Ennek eléréséhez azt a lehetőséget választottam, hogy egy görbét fektetek a pontokra, és a görbén már lehetőség van adott mennyiségű pont kijelölésére.

Mivel a görbe tetszőleges alakú lehet, ezért két függvényillesztést végzek el, ugyanis a töröttvonalakat leíró pontpárok helyett, az x és y koordinátákat a görbehosszhoz kapcsolom. Így létrehozok egy a hosszparaméterhez x és egy y koordinátát kapcsoló függvényt. Közelítő függvénynek egyszerű polinom függvényt választottam, amelynek együtthatóit a legkisebb négyzetek módszerével határozom meg. Mivel egy n fokú polinom pontosan $n+1$ adatponton tud egyszerre illeszkedni, így maximum ekkora polinomot szabad használni, hiszen a többlet szabadságfokok anomáliákhoz vezethetnek. Ha sűrűre választjuk a csomópontokat akkor jelentősen kisebb fokú polinom is elég lehet a jó leképezéshez, ami egyrészt csökkenti az erőforrásigényt, másrészt a mérési hibák hatásait is, hiszen a kutya hátának vonala viszonylag sima. Mivel a fokszámnak illeszkedni kell a bemeneti pontpárok számához, így olyan algoritmust kellett készíteni, amely tetszőleges fokú polinom illesztésére képes.

A legkisebb négyzetek módszerének alkalmazása során egy lineáris egyenletrendszert kell megoldani, ezért először egy lineáris egyenletrendszer megoldó algoritmust kellett létrehoznom. Több nyílt forráskódú programkönyvtár létezik az ilyen problémákra, azonban mégis saját implementálás miatt döntöttem, ugyanis ezek az algoritmusok viszonylag gyorsan implementálhatók, nem éri meg hatalmas modulokat hozzáadni a projekthez, és könnyebb a karbantarthatóság is, illetve magasabb teljesítmény is elérhető.

A lineáris egyenletrendszer megoldó módszerek nagyrészt a Gauss elimináción alapulnak, azonban annak teljesítményén több módon is javítanak. Ezek közül elterjedtebben alkalmazott az úgynevezett LU dekompozíció és visszahelyettesítés [15]. A módszer lényege, hogy az egyenletrendszer együttható mátrixát két mátrix szorzatává alakítja, a tényezők felső illetve alsó trianguláris mátrixok lesznek. Az U mátrix főátlója alatt csak nullák találhatók, míg az L mátrix főátló feletti elemei lesznek mind nullák. Ezzel a módszerrel valójában két lineáris egyenletrendszer

megoldását kell elvégeznünk, ezek megoldása azonban nagyon egyszerű és gyors, mivel a trianguláris mátrixokkal felírt egyenletek egyike önállóan megoldható, és az így kapott eredményt a következőbe helyettesítve egymás után végighaladhatunk az összesen:

$$\underline{\underline{A}} \cdot \underline{x} = \underline{b}$$

$$\underline{\underline{A}} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{bmatrix} = \underline{\underline{L}} \cdot \underline{\underline{U}}$$

$$\underline{\underline{L}} \cdot \underline{\underline{U}} = \begin{bmatrix} l_{11} & 0 & 0 & \dots & 0 \\ l_{21} & l_{22} & 0 & \dots & 0 \\ l_{31} & l_{32} & l_{33} & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ l_{n1} & l_{n2} & l_{n3} & \dots & l_{nn} \end{bmatrix} \cdot \begin{bmatrix} u_{11} & u_{12} & u_{13} & \dots & u_{1n} \\ 0 & u_{22} & u_{23} & \dots & u_{2n} \\ 0 & 0 & u_{33} & \dots & u_{3n} \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & u_{nn} \end{bmatrix}$$

Az egyenletrendszer megoldásának két lépésre bontása:

$$\underline{\underline{A}} \cdot \underline{x} = \underline{\underline{L}} \cdot \underline{\underline{U}} \cdot \underline{x} = \underline{\underline{L}} \cdot (\underline{\underline{U}} \cdot \underline{x}) = \underline{b}$$

$$\underline{\underline{L}} \cdot \underline{y} = \underline{b}$$

$$\underline{\underline{U}} \cdot \underline{x} = \underline{y}$$

Az LU felbontás felírható egyetlen mátrixban is, ha úgy végezzük el, hogy az egyik mátrix főátlójában csak egyesek szerepelnek. Az LU felbontás stabilitása szempontjából szükség van a sorok normalizálására, melynek során a sort annak legnagyobb elemével leosztjuk. A felbontás során Crout algoritmusát alkalmaztam, ahol az alsó trianguláris mátrix főátlójában csak egyesek szerepelnek. A felbontás során oszloponként végighaladunk a mátrixon. A végighaladás során csak az aktuális oszloppal megegyező, vagy nagyobb sorszámú sorokat kell nézni. Először ki kell választani a legnagyobb értéket az adott oszlopban, majd ha ez nem abban a sorban van, amely a vizsgált oszloppal azonos sorszámú, akkor ki kell őket cserélni. Ezután már csak arra van szükség, hogy az oszlophoz tartozó sorral elimináljuk az alatta lévő sorokban az oszlophoz tartozó elemeket. A folyamat során a keletkező nullák helyére beírjuk azt a számot, amellyel az elimináló sort megszoroztuk az adott sor feldolgozása során, így pont egy ilyen alakú mátrixot kapunk:

$$\underline{\underline{LU}} = \begin{bmatrix} u_{11} & u_{12} & u_{13} & \dots & u_{1n} \\ l_{21} & u_{22} & u_{23} & \dots & u_{2n} \\ l_{31} & l_{32} & u_{33} & \dots & u_{3n} \\ \dots & \dots & \dots & \dots & \dots \\ l_{n1} & l_{n2} & l_{n3} & \dots & u_{nn} \end{bmatrix}$$

A folyamat során fel kell jegyezni az alkalmazott normalizáló szorzókat, illetve a sorok eredeti helyét is, hogy az egyenletrendszer jobb oldalán álló b vektort is át tudjuk rendezni a változtatásoknak megfelelően. Innen már egyszerűen megoldható az egyenletrendszer:

$$y_i = \frac{b_i - \sum_{j=1}^{i-1} l_{ij} \cdot y_j}{l_{ii}}, i = 1..n$$

$$x_i = \frac{y_i - \sum_{j=i+1}^n u_{ij} \cdot x_j}{u_{ii}}, i = n..1$$

A lineáris egyenletrendszerek megoldási algoritmusának elkészítése után erre építettem az általános legkisebb négyzetek kiszámító algoritmusát. Ezzel tetszőleges számú adatpontra illeszthető általunk megadott fokszámú polinom (figyelembe véve, hogy a pontok számánál eggyel kisebb fokú polinomot kell választani a pontok közötti anomáliák elkerülése érdekében).

Tekintsük a következő függvényt:

$$f(x, a_0, a_1, \dots, a_n) = \sum_{i=0}^n a_i x^i$$

Továbbá adottak a következő pontpárok:

$$(x_j, y_j), j = 1..m$$

Ha a polinommal közelítjük a pontpárokat, akkor az eltérések négyzetösszege:

$$E(a_0, a_1, \dots, a_n) = \sum_{j=1}^m (y_j - f(x_j, a_0, a_1, \dots, a_n))^2$$

Az eltérések négyzetösszege minimális lesz ha:

$$\frac{\partial E}{\partial a_i} = 0, i = 0..n$$

Az egyenletrendszert mátrix formában felírva:

$$\underline{A} \cdot \underline{x} = \underline{b}$$

$$a_{ij} = \sum_{j=0}^n \sum_{l=0}^m x_l^{i+j}$$

$$b_i = \sum_{j=0}^m y_j \cdot x_j^i$$

Az implementált algoritmus a többször szereplő hatványokat és összegeket csak egyszer számolja ki. A vonalhossz mentén paraméterezve két dimenzióban két közelítés kerül kiszámításra. A paraméterezés során kiadódik a vonal hossza, így annak mentén könnyen kiválasztható megfelelő mennyiségű pont az időbeli interpolációhoz. A rendszer a közelítő polinom fokszámát a bemenő pontok számától függően választja ki (arányos és maximum fokszám is megadható egyszerre), így sosem eshet meg, hogy a közelítési feladat alulhatározott lesz, és a polinomban nemkívánatos hullámok jönnek létre. Én nagyrészt a pontok számának harmada fokszámú polinomot (de maximum 7. fokút) használtam, de ez beállítható.

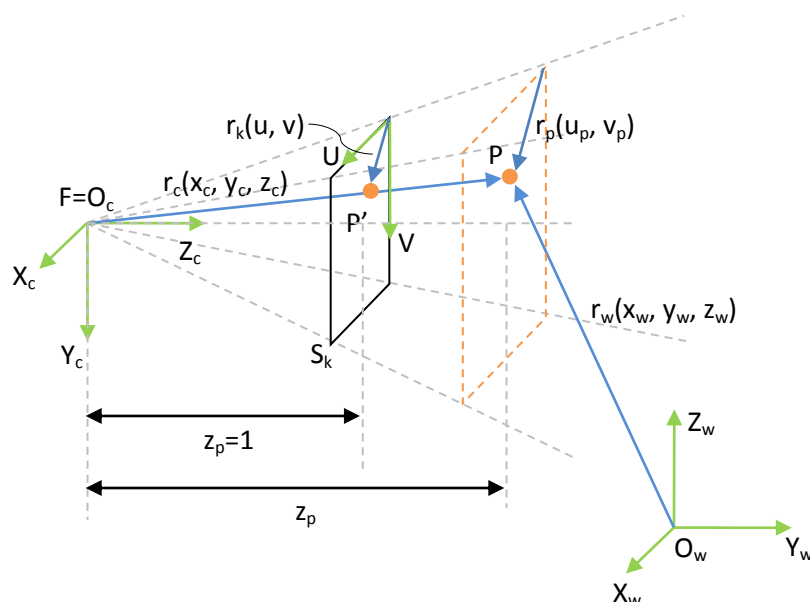
Ha a közelítő polinomok rendelkezésre állnak, akkor lehetőség van arra, hogy a vonalhossz mentén adott mennyiségű pontot kiválasszunk, majd az egymás utáni képkockák között interpoláljuk helyzetüket. Az így keletkezett új töröttvonal már tetszőleges időpontban elérhető. Mivel a kamerák elég nagy sebességűek, így én csak lineáris interpolációt használtam időben, azonban kisebb képkocka sebesség esetén érdemes lehet legalább egy másodfokú közelítést választani, mert ellenkező esetben a pontok sebességében ugrások tapasztalhatóak, ami rontja az eredmények minőségét.

4.4.3 Csomópontok transzformációja

A feldolgozás során négy koordinátarendszert használunk:

- A kép koordinátarendszere a kép pixeleinek elérésénél használt koordinátarendszer, melynek origója a kép bal-felső sarkában van, a koordináták pedig balról-jobbra (x-tengely) illetve fentről lefelé (y-tengely) nőnek.
- A vetítési koordinátarendszer: ez egy speciális a perspektivikus vetítéshez használt koordinátarendszer. Ez a koordinátarendszer a kép koordinátarendszerére illeszthető rá (x és y tengely), azt egy további mélység dimenzióval kiegészítve (z tengely). A perspektivikus vetítésnél az itt felírt háromdimenziós vektorok koordinátáit leosztva azok mélységi koordinátájával, megkapható képen elfoglalt helyük (a kép a $z=1$ síkban van).
- A kamerához rögzített háromdimenziós koordinátarendszer: ez a koordinátarendszer már valós egységekben méri a pozíciókat, origója a kamera fókuszpontjában van, z tengelye a képsíktól mért távolságot mutatja, másik két tengelye pedig párhuzamos a képsík megfelelő tengelyeivel.
- A világkoordinátarendszer a mért pozíciók laborhoz rögzített leírására szolgál.

Ez az úgynevezett túlyuk kamera modell:

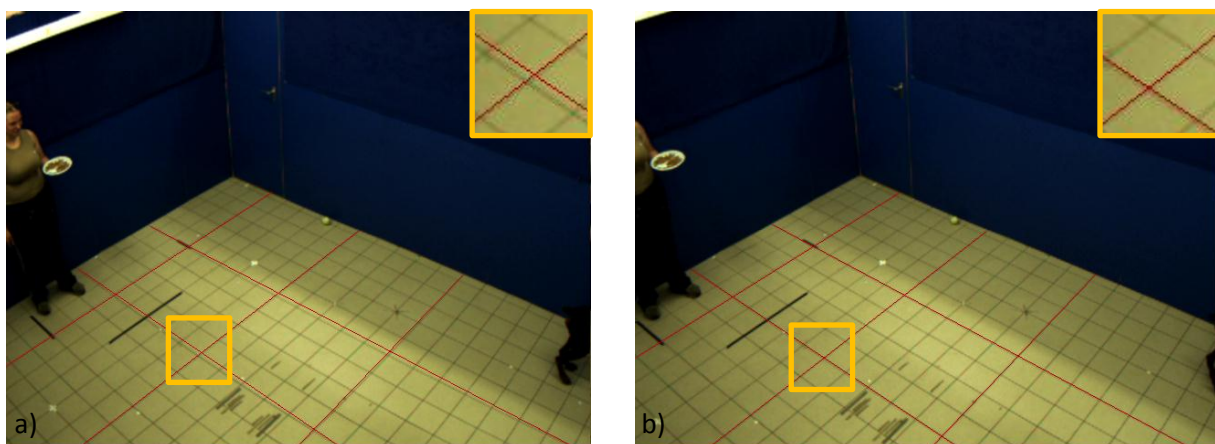


15. ábra A használt koordinátarendszerek

Ahol:

- P : a leképezendő pont
- P' : a leképezett pont
- O_w : a világ koordinátarendszer origója
- X_w, Y_w, Z_w : a világ koordinátarendszer tengelyeinek egységvektorai
- x_w, y_w, z_w : P koordinátái a világ koordinátarendszerben, valamely „valós” mértékegységben (pl.: mm)
- O_c : a kamera koordinátarendszer origója, fókuszpont
- X_c, Y_c, Z_c : a kamera koordinátarendszer tengelyeinek egységvektorai
- x_c, y_c, z_c : P koordinátái a kamera koordinátarendszerben, valamely „valós” mértékegységben (pl.: mm)
- u_p, v_p, z_p : a perspektivikus vetítési térben használt koordináták, a z_p a kamerától mért távolság, u_p, v_p a képsík P pontra illeszkedő képén vett távolság a kép bal-felső sarkától, ezen koordináták már pixelben vannak
- U, V : a képsík koordinátarendszerének egységvektorai
- u, v : a P' pont koordinátái pixelben

Az itt bemutatott koordinátarendszerek közül az utolsóból a harmadikba az extrinsic paraméterek mátrixa segítségével lehet eljutni, míg a harmadikból a másodikba az intrinsic mátrix visz, innen az elsőbe pedig a mélységkoordinátával való osztással jutunk. A mérés során ezen műveleteknek a fordítottját kell elvégezni. Ehhez a kép koordinátarendszerében lévő pontok koordinátáit háromdimenziósra bővítjük egy egységnyi értékkel, majd innen az intrinsic és extrinsic mátrixok szorzatának inverzével jutunk a képponthoz rendelhető irányvektorhoz. Így a töröttvonalból kapott irányvektorokat a kamera helyének világkoordinátarendszerben felírt pozíciójával együtt tárolom egy erre a célra létrehozott osztályban. A transzformáció előtt fontos figyelembe venni a kamera torzításait is: ez történhet az előfeldolgozóval, amely a teljes képet torzulásmentessé teszi, illetve lehetőség van arra is, hogy a gerinc képen vett koordinátáit korrigáljuk csak, hiszen a mozgásdetektálást illetve a kutya körvonalának keresését nem zavarják ezek a hibák. Én az előbbit választottam, mivel a képviszatorzítást a grafikus kártya végzi, tehát a CPU-t nem terheli.



16. ábra A képtorzítások korrigálása: a) torz kép b) helyreállított kép

A radiális és tangenciális képhibák javítását már bemutattam az előző TDK dolgozatban [11], így itt bővebben nem ismertetném. Lényege, hogy a szoftver előre kiszámít egy vektor textúrával leképezett vektor-vektor függvényt, amely a torzítatlan koordinátákhoz torzítottakat rendel, majd a feldolgozás során, a grafikus kártyán korrigálja a hibákat. Ez úgy történik, hogy amikor egy képpontot renderelni kíván az adott processzormag, először kikeresi hova került a torzítás során az a pixel, aminek értékére szükség van, majd azt írja a kimenet adott helyére. Így a kép torzításai megszüntethetők, ráadásul a kártya által biztosított interpolációs algoritmusok biztosítják az eredmény magas minőségét.

4.4.4 Héjak metszete

Egy kamera pozíciójából és a belőle kiinduló - az előző lépésben kapott – irányvektorokból egy legyezőszerű héjszerkezet hozható létre. Ha tekintünk két szomszédos gerinccsomóponthoz tartozó irányvektort a kamera pozíciójából indítva, akkor ezek végpontjai a közös kiindulópontjukkal egy háromszög három csúcspontját határozzák meg. Ha az irányvektorokat végtelen nagyra növeljük, akkor a háromszögek félsíkokká válnak, és ezen félsíkok sorozata egy végtelen távolba nyúló héjat alkot.

Ha el akarunk metszeni két ilyen héjat, akkor először egy-egy szegmens metszését kell megoldani. Ehhez először ki kell számítani az adott félsíkok síkjainak metszetét. Egy sík leírható egy pontjával és normál vektorával:

$$\mathbf{n}_1 \cdot \mathbf{r} = h_1$$

$$\mathbf{n}_2 \cdot \mathbf{r} = h_2$$

Ahol:

- $\mathbf{n}_1, \mathbf{n}_2$ – a síkok normálvektorai
- h_1, h_2 – a síkok normálvektorának és a síkok egy pontjának skaláris szorzata
- $\mathbf{r}$ – egy a síkon lévő pont helyvektora

Ha egy pont rajta van mindkét síkon, akkor annak helyvektora kielégíti a következő egyenletet is:

$$\mathbf{r} = (c_1 \mathbf{n}_1 + c_2 \mathbf{n}_2) + \lambda (\mathbf{n}_1 \times \mathbf{n}_2)$$

Mivel a két sík normálvektorai és vektoriális szorzatuk egy bázist alkotnak, így ennek a pontnak is felírhatónak kell lennie lineáris kombinációjukként (ha a két sík normálvektorai nem párhuzamosak). Ezen egyenletet a síkok egyenletébe helyettesítve:

$$c_1 = \frac{h_1 - h_2(\mathbf{n}_1 \cdot \mathbf{n}_2)}{1 - (\mathbf{n}_1 \cdot \mathbf{n}_2)^2}$$

$$c_2 = \frac{h_2 - h_1(\mathbf{n}_1 \cdot \mathbf{n}_2)}{1 - (\mathbf{n}_1 \cdot \mathbf{n}_2)^2}$$

Ezzel a metszővonal egy pontja már ismert, iránya pedig a normálvektorok keresztszorzatából adódik. Így megkapom a két sík metszővonalát. Ezután meghatározom a metszővonal és a félsíkok határoló félegyeneseinek metszéspontjait, és kiszámítom ezek távolságát a metszővonal felírásakor használt ponttól. Ezek olyan távolság intervallumokat adnak, melyek közös része mutatja meg a két

félsík metszészíkjának végeit. Ha nincs közös rész a két félsíknak nincs metszete. A vonalak metszéspontja a térben a numerikus pontatlanság miatt általában nem valósul meg, ezért ehelyett az egyeneseket összekötő legrövidebb szakasz felezőpontjába teszem a metszéspontot. Ehhez először kiszámítom az egyenesek távolságvektorát: a két egyenes kezdőpontjának távolságát skalárisan kétszer megszorozom az egyenesek irányvektorainak normalizálásával kapott vektorral. Majd a síkok metszéséhez hasonló módon járok el, és felírom a két egyenes távolságát jelentő vektort az egyenesek irányvektoraiból és azok vektoriális szorzatából álló bázisban. Innen kifejezhető, mekkora távolságot kell megtenni az egyik egyenes mentén ahhoz, hogy elérjük a távolságvektor kiindulópontját, ha ehhez még hozzáadjuk a távolságvektor felét, akkor megkapjuk a kívánt eredményt. A metszéspontok meghatározása után kiszámolom azok távolságát a síkok metszetét adó egyenes kiindulópontjától, majd ellenőrzöm van-e közös szakasza a közös egyenes félsíkokkal vett metszeteinek.

A rendszer úgy számítja ki a héjak metszetét, hogy elindul az egyik héj mentén, és megkeresi a lehetséges metszeteket a másik héjjal. Az algoritmus úgy választja ki a másik oldalról a megfelelő félsíkot, hogy megkeresi a kiinduló oldal félsíkjának első határfélegyeneséhez legközelebb eső félegyeneset, hiszen ha van metszés, akkor annak ezen félegyenes környezetében kell lennie. Ezután addig vizsgálja a túlsó oldalról a szomszédos félsíkokat (egy félsíkot több is metszhet), amíg nem ér ki a kiinduló oldal által közrefogott térrészről. Így minden metszéspont megkapható, amelyeket a rendszer a kiinduló oldal félsíkjai szerinti tömbbe rendez, ezen tömb minden eleme egy lista, amely az adott félsíkhöz tartozó töröttvonalrészt tartalmazza. Mivel ezen töröttvonalrész irányítottasága változó lehet, ezért külön algoritmust igényelt ezek sorba rendezése. Ez az összerakó algoritmus megnézi, hogy a metszetdarabok hogyan kapcsolódhatnak egymáshoz a végeik azonos pozíciója alapján, és ezt figyelembe véve építi fel az eredményt.

4.4.5 Eredmények korrekciója

A héjak metszetéből kapott töröttvonal még nem mutatja meg jól a kutya helyét, mivel a héjak a kutya hátának eltérő vonalát érintik. Ezért az eredményeket korrigálni kell a kutya szélességének ismeretében. Tételezzük fel, hogy a hát felülete nem más, mint egy görbe tengelyű henger, amelynek ismerjük a sugarát. Ez a henger felbontható differenciálisan kicsi hengerekre, amelyek már egyenesek. Ha egy ilyen elemi henger palástjához két térbeli pontból egyeneseket vetítünk, akkor két félsíkot kapunk. Ezek metszete az a vonal, amit az előző számításokból kaptunk. Ha a henger tengelyére merőlegesen összekötjük a vonal pontjait a henger tengelyével, akkor a szakaszok palásttal vett metszete adja a hát tényleges vonalát. Ezek alapján lehetőség van arra, hogy ezt a számítást visszafelé is megoldjuk, azaz a látszólagos vonalból meghatározzuk a háton lévő vonal igazi helyzetét.

A megoldáshoz az algoritmus végigmegy a héjmetszésből kapott töröttvonalon, és szakaszonként kiszámítja a szakasz irányvektorát, majd ezeket átlagolja, ez adja a hengerdarabok irányát. Azért nem lehet a szakaszonként kezelni a henger irányát, mert ilyenkor, ha a kamerák szinkronja nem tökéletes, akkor létrejöhetnek a kamerák tengelyeinek irányába forduló hengerdarabok, amelyek torzulásokat okoznak. Mivel a számítás eleve becslésen alapul, így nem viszünk be ezzel jelentősebb hibát a rendszerbe.

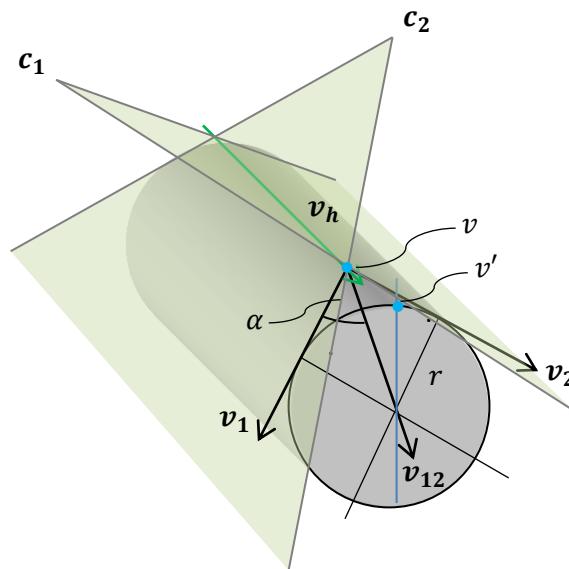
Ezután a program kiszámolja a hengert érintő félsíkok hengertengelyre merőleges irányvektorait:

$$\mathbf{v}_1 = \frac{(\mathbf{v} - \mathbf{c}_1) \times \mathbf{v}_h \times \mathbf{v}_h}{|(\mathbf{v} - \mathbf{c}_1) \times \mathbf{v}_h \times \mathbf{v}_h|}$$

$$\mathbf{v}_2 = \frac{(\mathbf{v} - \mathbf{c}_2) \times \mathbf{v}_h \times \mathbf{v}_h}{|(\mathbf{v} - \mathbf{c}_2) \times \mathbf{v}_h \times \mathbf{v}_h|}$$

Ahol:

- $\mathbf{v}_1, \mathbf{v}_2$ – a kiszámolandó vektorok, amelyek merőlegesek a henger tengelyére, és beleesnek a kamerák helyvektoraira és a javítandó töröttvonal szegmensre fektethető síkra
- $\mathbf{c}_1, \mathbf{c}_2$ – a kamerák helyvektorai
- $\mathbf{v}_h$ – a henger tengelyének irányvektora



17. ábra A hátvonal hibájának korrekciója

Legyen $\mathbf{v}_{12}$ vektor $\mathbf{v}_1$ és $\mathbf{v}_2$ vektorok összegének normáltja, ezen helyezkedik el a henger tengelyének egy pontja:

$$\mathbf{v}_{12} = \frac{\mathbf{v}_1 + \mathbf{v}_2}{|\mathbf{v}_1 + \mathbf{v}_2|}$$

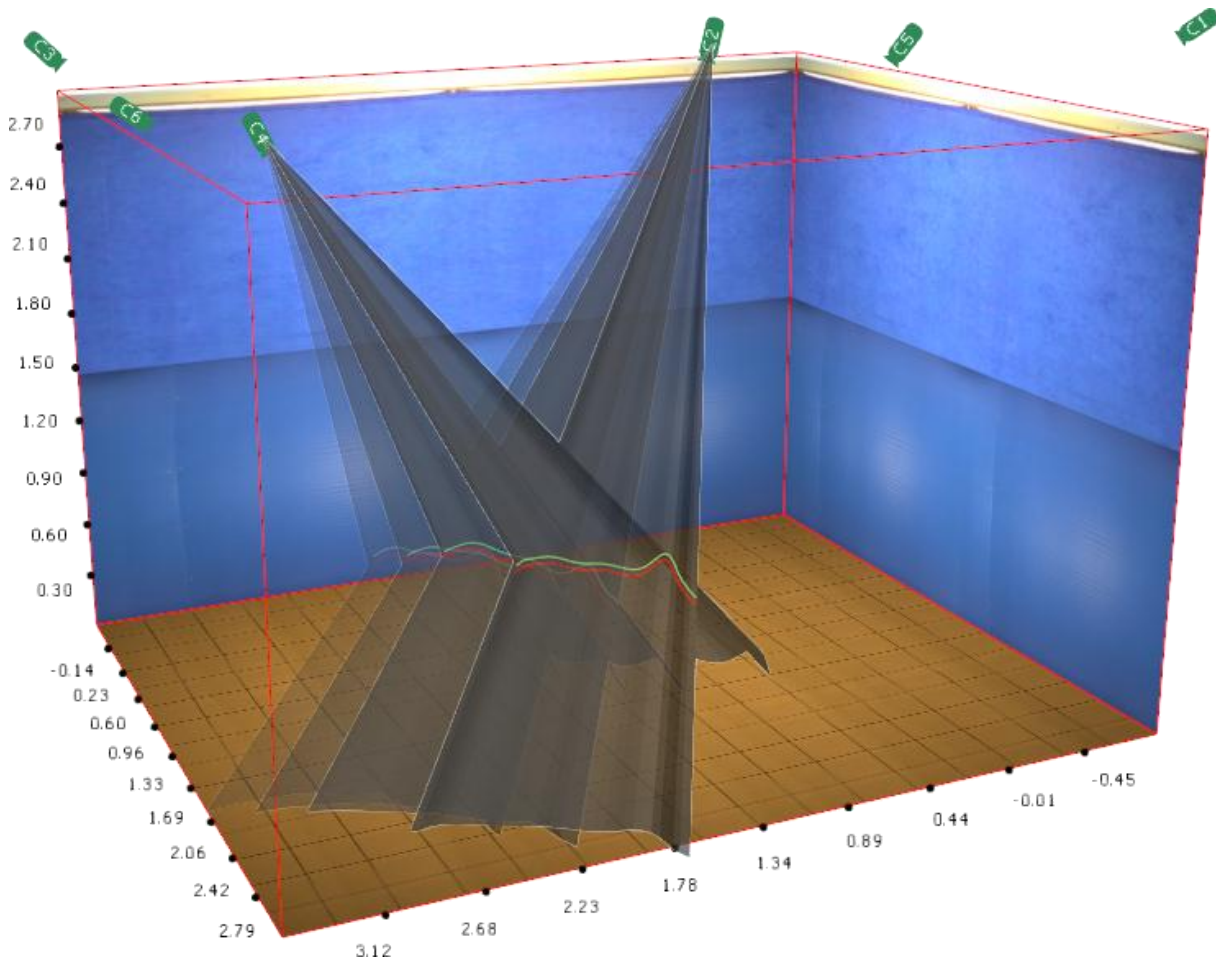
Innen könnyen kiszámolható az adott csomópont új helye:

$$\sin(\alpha) = \frac{|\mathbf{v}_1 \times \mathbf{v}_{12}|}{|\mathbf{v}_1| \cdot |\mathbf{v}_{12}|}$$

$$\mathbf{v}' = \mathbf{v} + \mathbf{v}_{12} \cdot \left[\frac{r}{\sin(\alpha)} \right] + \begin{bmatrix} 0 \\ 0 \\ r \end{bmatrix}$$

A bemutatott módszerrel az eredmények minősége és ezzel használhatósága erősen javul, különösen olyan esetekben, amikor a héjak kis szöget zárnak be egymással.

Ha két kamera képe rendelkezésre áll, akkor meghatározható a gerinc helyzete, ha több mint két kamera áll rendelkezésre, akkor választani kell közülük. Ha eltekintünk attól az esettől, mikor több kamera a kutya egy részét látja, és ezeket kellene kombinálni, akkor viszonylag egyszerű a feladat. Jó eredményeket azok a kamerapárok tudnak adni, amelyek szembe néznek egymással, így azt a párt kell kiválasztani, amely ennek az elvárásnak legjobban megfelel. Ehhez elég skalárisan összeszorozni a kamerák normált irányvektorait és minél közelebb áll az eredmény -1-hez annál jobb a választás. A kamerák irányvektorainak magasság irányú komponensét alapesetben nem kell figyelembe venni. Természetesen, ha több kamera is jó szögben van, akkor az eredmények átlagolására is lehetőség lenne, bár ez jelenleg nincs implementálva. Mivel egyelőre elsősorban az oldalnézeti megfigyeléssel foglalkoztam, így kevésbé gyakori, hogy több kamera is nagyon jó helyzetben van. Amint elkészülnek a maradék nézeteket kezelő részek tovább fogom fejleszteni a kamerakezelést is.



18. ábra A kutya hátvonalának eredeti (zöld) és korrigált (piros) helyzete

4.4.6 Háromdimenziós megjelenítés

Az adatösszevonó modul fontos részét képezi az eredmények háromdimenziós megjelenítése, melyet az XNA keretrendszer felhasználásával oldottam meg. Ez a rendszer viszonylag alacsony szintű hozzáférést biztosít a DirectX hardverekhez, és biztosítja azok objektumorientált sémájú elérését. Az XNA-ról számos programozási ismeret megszerezhető a [18]-ből.

A háromdimenziós grafikon elkészítése során a beépített árnyaló algoritmusokat használtam, mivel funkcionalitásuk megfelelt elvárásaimnak. A grafikon koordinátarendszere különbözik a

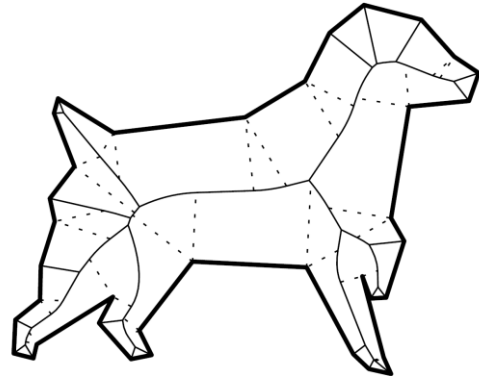
valószínű, hogy könnyen kezelhetőek legyenek különböző méretű laborok is. Programlogikai szempontból a grafikon kocka alakú és csak az utolsó lépésben torzítom el a valóságos arányoknak megfelelően. A grafikon falaira fényképek alapján készített végtelenített textúrákat raktam. A kamerák ikonjai és skálák billboarding [18] technikával készültek: azaz úgy kerültek elhelyezésre, hogy pontszerű háromdimenziós pozíciójukból kiszámítottam a perspektivikusan vetítettet, majd ide elhelyeztem egy a kétdimenziós, állandó nagyságú képet, így háromdimenziós hatást értem el. A skálák a kamera helyzetétől függően más-más élek mentén jelennek meg, így bármely szögből jól láthatók. A kamera mindig a grafikon középpontjába néz, ugyanakkor helyzete – amely két szöggel és a középponttól való távolsággal van leírva – változtatható. A rajzolás során vonalláncokat és háromszögeket rajzoltam fel, melyek egyszínűek vagy textúrázottak. A jobb kiértékelhetőség végett két irányfényforrást használtam, melyek függőlegesen lefele és felfele irányulnak. A fényforrások Lambert visszaverődés törvénye alapján adják a legyezők felületéről visszaverődő fény intenzitását. A fényforrások a legyezőkön kívül más objektumot nem érintenek. A Lambert visszaverődéshez természetesen a legyezők szegmenseinek normálvektorait is meg kellett határozni, mivel egy csúcshoz csak egy normális adat rendelhető, ezért ugyanazt a csúcsot, ha több szegmens része, többször is el kell küldeni a grafikai memóriába. A rajzolás során felhasználtam a back-face culling technikát, amely a háromszögek körüljárási irányát használja fel annak megállapítására, hogy azok a kamera felé néznek-e. Ez ebben az esetben azért hasznos, mert így elkerülhető a grafikont máskülönbben eltakaró, a nézőhöz közelebb eső falak kirajzolása. Mivel a grafikonon több rajz is azonos mélységben van a kamerához képest, ezért például a kocka drótvázának rajzolásánál úgy járok el, hogy azt kétszer rajzolom ki: először mélységpuffer nélkül, majd azzal együtt. Így a falak felett látszik a drótváz, ugyanakkor a legyezők a hátsó vonalakat ki tudják takarni, de az elől lévőket nem (a 18-as ábra esetén ez nem látszik, mivel ott több különböző grafikon-t helyeztem egymásra, a mozgás érzékeltetésére). A rajzolás során bekapcsoltam az élsimítást 8 mintával, így a vonalak csipkéződése nem jelentkezik. A szövegek kiíratásához bitkép betűtípust használtam, amelyhez egy speciális elrendezésű képet kell készíteni, amely tartalmazza a használni kívánt karaktereket. Az XNA előfeldolgozója az összes tartalmat: a textúrákat, effekteket, betűtípusokat stb. feldolgozza, lefordítja. Így például a bitképbetűtípus használata során már egyszerűen stringeket írathatunk ki. A falak textúrái egy bitképben vannak elhelyezve, ahonnan azok kirajzoláskor a csomópontokhoz kapcsolt textúrákoordinátákkal érhetőek el.

A grafikon elkészítése során ügyeltem arra, hogy az jól újrahasznosítható legyen más projektekben is. A későbbiekben a textúrákat kicserélve láthatóvá válik majd a szoba tájolása, az ajtók stb. helye alapján, így még könnyebbé válik a látott adatok értelmezése.

5 Továbbfejlesztési lehetőségek

A program képes a kitűzött célok megvalósítására, ennek ellenére számos olyan terület van, amelyen még sokat lehet fejleszteni. Ezek közül talán az alábbiak a legfontosabbak:

- **A kutya képből való kiválasztásának megoldása fejlettebb módszerrel:** a jelenlegi tesztek alapján a színhisztogram alapú módszer teljesen jól használható, hosszútávon mégis érdemes lehet egy olyan rendszert kidolgozni, amely több adatot felhasználva, például különböző tárgyak képeit tartalmazó adatbázis hasznosításával ismeri fel az objektumokat. Bár így nőne a gépigény, de lehetővé válna több tárgy felismerése, az emberekről, állatokról általános modell felállítása is, amellyel az operátor közreműködése nélkül is végbemehetne például ismeretlen külsejű kutyák vagy emberek felismerése is.
- **Fejlettebb helyzetfelismerés:** A program jelenleg a kutyát járás, futás közben tudja legjobban követni, ha a mozgás iránya a kamera képének vízszintes tengelyével kb. 0-60 fokos szöget zár be. A program az esetek nagy részében, a sokáig ülő, forgolódo állatot is jól követi, itt azonban néha már előfordulhatnak hibák, elsősorban az árasztás kezdőpontjának elhelyezése miatt, hiszen ez jelenleg járáshoz, futáshoz van optimalizálva, a fekvéshez, egyhelyben való mozgáshoz (pl.: kutya hátán forgolódik stb.) érdemes más módszert is kifejleszteni.
- **Csontvázgenerálás:** A kutya határvonalának ismeretében létrehozható egy csontvázszerkezet is, amely ugyan nem lesz hű a kutya valóságos csontvázának felépítéséhez, de több kamerából összevetve alkalmas lehet például a kutya lábainak pozíció elemzésére is, és a kutya testhelyzetének széleskörű vizsgálatát is lehetővé tenné. Ilyen csontvázak generálására használható például az ún. Voronoi-diagram, mely a kutya már meghatározott körvonalából felépíthető [16].
- **Viselkedéselemzés:** A projekt elsődleges célja a kutya viselkedésének elemzése, ehhez olyan módszereket kell kidolgozni, melyek megfelelő mérések alapján képesek felismerni, hogy a kutya áll, ül, fekszik, fut stb. Ez elvégezhető lenne például mérési eredményekkel betanított neurális hálók segítségével.
- **Interfészek készítése különböző programokhoz:** Ugyan a program könnyen bővíthető további modulokkal, ugyanakkor célszerű ezen adatok más rendszerek által való elérését (pl.: LabVIEW, MATLAB) is biztosítani, ez már most is megtehető, hiszen a fájlbeolvasást és hálózati kommunikációt biztosító modulok elkülönítve is rendelkezésre állnak, ugyanakkor az adatok eléréséhez még a rendszerek összekapcsolását is meg kell oldani.
- **Videó batch feldolgozása:** A jelenlegi videoelérő rendszer a valós idejű feldolgozást szimulálva állandó sebességgel bocsájtja rendelkezésre a képkockákat, azonban ez nem használja a ki a gép teljes sebességét batch feldolgozás esetén, ezért ezt még meg kell valósítani.

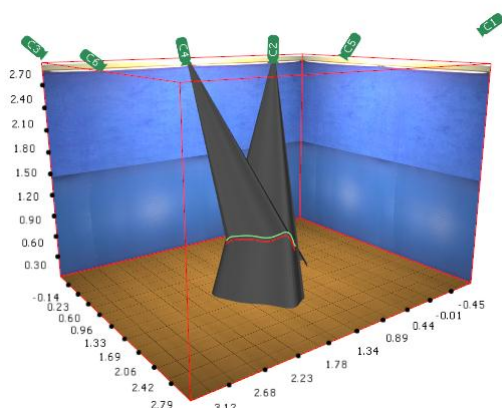


19. ábra Kutya csontvázának előállítása Voronoi-diagrammal

- **Többnyelvű kezelőfelület:** a rendszer jelenleg teljesen angol nyelvű, ennek ellenére érdemes lenne azt még német, magyar stb. kezelőfelülettel is ellátni. A .Net rendszer fejlett lokalizációs képességekkel rendelkezik így ez könnyen megoldható.

Ezen felsorolt funkciók jelentős része a programozás, tesztelés nagy időigénye miatt nem került még be a programba, de a munkát tovább folytatom ezek irányában. A továbbfejlesztéshez tartozik a program további tesztelése, javítása, optimalizációja, még robosztusabbá tétele, a kezelőfelület bővítése és megfelelő dokumentáció készítése.

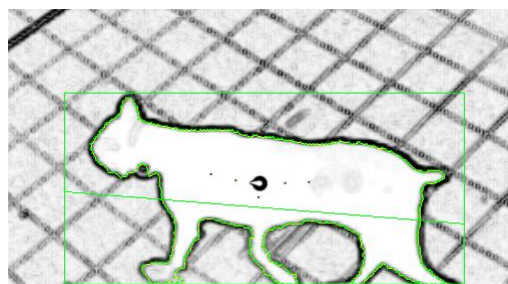
6 Eredmények értékelése és konklúzió



20. ábra Gerincpozíció meghatározása

A rendszer jelenleg jól használható a kutya mozgás közbeni helyzetelemzésére, megmérhető a hát vonalának háromdimenziós helye. A rendszer pontosságának tesztelésére még nem volt lehetőség, de ez elsősorban az alkalmazott kamerától, azok és a kutya elhelyezkedésétől függ. Fontos tényező az is, hogy a kutya szőre milyen hosszú, hiszen a hosszú szőr egyértelműen rontja a rendszer precizitását. Beállítástól függően 0,5 - 3cm pontossággal kell számolni (2-3 méterre a kamerától), ez azonban teljesen megfelelő, ha figyelembe vesszük, hogy itt elsősorban a kutya különböző mozgásformáit: fut, jár, fekszik, ül, ugrik stb. kell felismerni, amely ilyen vagy ennél kisebb pontosság esetén is megoldható. Ahogy az a grafikonon is látható, az eredmények zajossága alacsony, a görbe alakja megfelelő. A rendszer adatfúziós alkalmazása projektkezelési szolgáltatásokkal könnyíti a felhasználók munkáját. A kapott eredmények exportálása után beolvashatók más feldolgozó rendszerbe, ahol majd elvégezhető a mozgásformák felismerése. Mivel egyelőre csak kevés videó anyag állt rendelkezésemre, így nem tudtam még minden részt teljesen letesztelni, azonban az eredmények már most is jól értékelhetők.

Az érzékelés elsősorban oldalirány esetében alkalmazható jelenleg, ugyanakkor belátható, hogy az elkészült kutyakörvonal alapján szemből vagy hátulról is kinyerhetők a szükséges adatok, ennek implementálására azonban eddig az idő- és erőforráshiány miatt nem kerülhetett sor. A kétdimenziós határvonal-keresés során számos módszert megvizsgáltam, és bemutattam ezek előnyeit és hátrányait. A létrehozott vektorgrafikus árasztás rugalmasan alkalmazható, szemben a klasszikus árasztási módszerekkel, alakja bármikor poligonként elérhető és elemezhető. Ez utóbbinak köszönhetően megoldható az éleken lévő szakadások kezelése és az élek vastagságának felhasználása is. Amennyiben a résen való kiömlés mégis megtörténik, akkor is könnyen megoldható ennek felismerése, és a felesleges rész eltávolítása. Az eredményül kapott gráf könnyen elemezhető, hiszen nem kell rajta semmilyen keresési műveletet végezni, mint például az elsőként bemutatott élkövető módszer esetében. A módszer jól alkalmazható zajos, rossz minőségű és/vagy tömörített képek esetén is. Az itt kifejlesztett algoritmus felhasználható egy teljes funkcionalitású szegmentáló rendszer kidolgozásához is. A rendszert a későbbiekben kombinálni szeretném mélységérzékelő szenzorokkal, mivel a mélységképeken sokkal jobban elvégezhető a kutya határvonalának megkeresése. Ezt csontvázgenerálással kombinálva teljes értékű motion capture rendszer nyerhető, amely alkalmazható más élőlényekhez, köztük emberekhez is.



21. ábra Kutya körvonal meghatározása

A létrehozott vektorgrafikus árasztás rugalmasan alkalmazható, szemben a klasszikus árasztási módszerekkel, alakja bármikor poligonként elérhető és elemezhető. Ez utóbbinak köszönhetően megoldható az éleken lévő szakadások kezelése és az élek vastagságának felhasználása is. Amennyiben a résen való kiömlés mégis megtörténik, akkor is könnyen megoldható ennek felismerése, és a felesleges rész eltávolítása. Az eredményül kapott gráf könnyen elemezhető, hiszen nem kell rajta semmilyen keresési műveletet végezni, mint például az elsőként bemutatott élkövető módszer esetében. A módszer jól alkalmazható zajos, rossz minőségű és/vagy tömörített képek esetén is. Az itt kifejlesztett algoritmus felhasználható egy teljes funkcionalitású szegmentáló rendszer kidolgozásához is. A rendszert a későbbiekben kombinálni szeretném mélységérzékelő szenzorokkal, mivel a mélységképeken sokkal jobban elvégezhető a kutya határvonalának megkeresése. Ezt csontvázgenerálással kombinálva teljes értékű motion capture rendszer nyerhető, amely alkalmazható más élőlényekhez, köztük emberekhez is.

A rendszer felhasználja mindazt, amit előzőleg kifejlesztettem, és azt bővítve, javítva ad annak új felhasználási lehetőséget. Az itt bemutatott technológiák természetesen nem csak kutya

mozgásának elemzésére használhatóak: az élek feldolgozásának kérdése számos képfeldolgozási feladatban előkerül. A körvonal megkeresése például használható egy villanykörte gyártósor működésének ellenőrzésére, a háromdimenziós rendszerrel pedig számos felületmérési feladat is megoldható. Számos más feladat megoldásához ugyan szükség van módosításokra, de az itt kialakított algoritmusok jól felhasználhatóak alapként. Fontos figyelembe venni azokat a fejlesztéseket is, amelyeknek nincs közvetlen kapcsolata ugyan a kutya mozgáselemzésével, de mégis nagyon fontosak. A képfeldolgozó keretrendszeren végzett módosítások jelentősen megnövelték annak teljesítményét és funkciókészletét. A rendszer teljesen moduláris lett, és nagyon könnyen bővíthető, fejlett feldolgozásvezérléssel, előfeldolgozóval, naplózási szolgáltatásokkal, hálózati kapcsolattartás lehetőségével, távvezérelhetőséggel, adat-vizualizációs képességekkel, hardveresen gyorsított, többszálú feldolgozással várja a további fejlesztéseket.

A munka eredményeként tehát sikerült egy olyan program létrehozása, amely jól megvalósítja a kezdeti célkitűzéseket, terveket. A program képes számos probléma megoldására már jelen állapotában is, továbbfejlesztve, specializálva pedig további képességekkel ruházható fel. Szerkezete lehetővé teszi a testre szabást, a keretrendszer, kezelőfelület a különböző algoritmusok gyors tesztelésére is lehetőséget ad. A programmal elvégezhető a kutyák gerincpozíciójának érintés nélküli megmérése, anélkül hogy a kutyára bármilyen érzékelőt, jelölőt kellene rakni. Így az etológusok természetes körülmények között vizsgálhatják az állatokat, akiknek nem kell emiatt plusz stresszt elviselnie. A fejlesztések folytatásával pedig még további lehetőségek tárulhatnak fel a kutatók számára.

7 Irodalomjegyzék

1. Moeslund, Thomas B., Adrian Hilton, és Volker Krüger. „A survey of advances in vision-based human motion capture and analysis.” Szerkesztette: A.C. Kak., *Computer Vision and Image Understanding, Volume 104, Issues 2-3*,. Amsterdam: Elsevier, 2006. 90-126. oldal
2. PrimeSense. „Making history - The PS1080”.
<http://www.primesense.com/en/company-profile/114-the-ps1080> (hozzáférés dátuma: 2011. május 7.)
3. Weise, T., B. Leibe, és L. Van Gool. „Fast 3D Scanning with Automatic Motion Compensation.” *2007. CVPR '07. IEEE Conference on Computer Vision and Pattern Recognition, 2007.*: 1 - 8.
4. Corazza, S., L. Mündermann, A. M. Chaudhari, T. Demattio, C. Cobelli, és T. P. Andriacchi. „A Markerless Motion Capture System to Study Musculoskeletal Biomechanics: Visual Hull and Simulated Annealing Approach.” *Annals of Biomedical Engineering Volume 34, Number 6*, 2009.: 1019-1029. oldal
5. Rosenhahn, B., és T. Brox. „Scaled Motion Dynamics for Markerless Motion Capture.” *IEEE Conference on Computer Vision and Pattern Recognition, 2007. CVPR '07. , 2007.*
6. Nicholas D. Butler, Adrian C. Gay, Jack E. Bresenham: „Line generation in a display system” U.S. patent number: 4,996,653, 1987
7. Yuri Pekelný, Craig Gotsman. „Articulated Object Reconstruction and Markerless Motion Capture from Depth Video.” *Computer Graphics Forum, Volume 27, Issue 2*, 2008.: 399–408.
8. Microsoft Corporation. „C# Language specification, version 4.0.” Redmond: Microsoft Press, 2010.: 427-446. oldal
9. „Programming Guide for HLSL.” *Windows Development Center*. 2011. június 9.
[http://msdn.microsoft.com/en-us/library/windows/desktop/bb509635\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/bb509635(v=vs.85).aspx)
(hozzáférés dátuma: 2011. július 22).
10. „Pixel Shader Differences.” *Microsoft Developer Network*. 2011. június 9.
[http://msdn.microsoft.com/en-us/library/bb219846\(v=VS.85\).aspx](http://msdn.microsoft.com/en-us/library/bb219846(v=VS.85).aspx) (hozzáférés dátuma: 2011. október 15).
11. Major Péter „Háromdimenziós mozgáskövetőrendszer jelölőeszközök használata nélkül.” *XXX. Országos Tudományos Diákköri Konferencia*. Baja, 2011. április 27-29.
12. Basler Vision Technologies. *Basler Pylon 2.0 Programmers Guide and API Reference*. 2008.
13. Péter Major „Image Processing Framework for Traffic Monitoring”, *2011 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*. Budapest, 2011. július 4-6.

14. Canny, J. „A Computational Approach To Edge Detection.” *IEEE Trans. Pattern Analysis and Machine Intelligence, Volume 8 Issue 6*, 1986.: 679-698. oldal
15. William H. P., Saul A. T., William T. V., Brian P. F.: *Numerical Recipes*, 3rd edition. Cambridge University Press, Cambridge, 2007.: 48. oldal
16. Mark de B., Otfried C., Marc van K., Mark O.: *Computational Geometry*, 3rd edition. Springer-Verlag Berlin Heidelberg, Berlin, 2008.: 169. oldal
17. Steven S. Skiena: *The Algorithm Design Manual*, 2nd edition. Springer-Verlag London, London, 2008.: 593. oldal
18. Riemer G.: *XNA 3.0 Game Programming Recipes – A Problem-Solution Approach*, Apress, New York, 2009.: 231. oldal
19. Microsoft Corporation. Microsoft Developer Network (MSDN) – <http://msdn.microsoft.com/>