



M Ű E G Y E T E M 1 7 8 2



BUDAPESTI MŰSZAKI ÉS GAZDASÁGTUDOMÁNYI EGYETEM

GÉPÉSZMÉRNÖKI KAR

MŰSZAKI MECHANIKAI TANSZÉK

Saját fejlesztésű „motion-tracking” szoftver készítése grafikus felhasználói felülettel Python környezetben

Floch Kristóf

Tudományos Diákköri Konferencia

Konzulens:

Dr. Kossa Attila

Egyetemi docens

BUDAPEST, 2022

Tartalomjegyzék

Kivonat	IV
Abstract	V
1. Bevezetés	1
2. Mozgáskövető és kiértékelő algoritmusok	4
2.1. Mozgáskövetés	4
2.1.1. BOOSTING algoritmus	4
2.1.2. MIL algoritmus	5
2.1.3. MEDIANFLOW algoritmus	7
2.1.4. TLD algoritmus	8
2.1.5. MOSSE algoritmus	9
2.1.6. KCF algoritmus	10
2.1.7. CSRT algoritmus	11
2.2. Numerikus differenciálás	12
2.2.1. Véges differencia módszerek	12
2.2.2. Modell alapú módszerek	13
2.2.3. Total variation regulatization	14
3. Szoftver felépítése	15
3.1. Szoftverarchitektúra	15
3.1.1. Grafikus felhasználói felület	15
3.1.2. Mozgáskövetés és elemzés	16
3.1.3. Eredmények megjelenítése, exportálása	16
3.2. Szoftver funkciók	17
3.2.1. Az applikáció indítása	17
3.2.2. Videók beolvasása és visszajátszása	18
3.2.3. Releváns részlet kijelölése	19
3.2.4. Követendő objektumok kijelölése	19

3.2.5.	Mértékegység kalibrálás	20
3.2.6.	Fókusz beállítása	20
3.2.7.	Mozgáskövetés és adatfeldolgozás	21
3.2.8.	Szögelfordulás mérése	22
3.2.9.	Eredmények megjelenítése és mentése	22
4.	Alkalmazási példák	25
4.1.	Mesterséges videók elemzése validálási célokra	25
4.2.	Biaxiális húzókísérlet elemzése	28
4.3.	Airsoft golyó becsapódásának vizsgálata	31
4.4.	Gépjárműforgalom vizsgálata drónfelvételek elemzésével	34
5.	Konklúzió	36
	Hivatkozások	37

Kivonat

Az utóbbi évek folyamatos fejlődése az optikai eszközöket sem kerülte el. Azzal párhuzamosan, hogy ma már mobil eszközeink segítségével is kiváló felvételeket tudunk készíteni, a professzionális, nagy felbontású és magas képkockasebességű (FPS) kamerák ára is folyamatosan csökken. Ennek következtében egyre felkapottabbak az olyan mérési eljárások, melyek során dedikált szenzorok alkalmazása helyett, a kísérletről készült videófelvételek feldolgozásával jutnak adatokhoz a mérnökök. A növekvő népszerűség ellenére a videó alapú mozgásanalízis elterjedését nehezíti, hogy a felvételek feldolgozása sokszor körülményes és akár rendkívül időigényes is lehet. Amennyiben valaki nem képes kifizetni a professzionális szoftverek magas licenszdíjait, vagy nem elégedett azok pontosságával vagy testreszabhatóságával, csak saját kód fejlesztésével képes a szükséges adatok kinyerésére.

Jelen dolgozat egy olyan ingyenesen elérhető, nyílt forráskódú, saját fejlesztésű mozgáselemző szoftvert mutat be videóképek feldolgozására, amely precizitásával és kiegészítő funkcióival megfelel a mérnöki elvárásoknak. Az elkészített applikáció grafikus felületének köszönhetően könnyedén használható a programozásban nem jártas mérnökök számára is. A sokoldalú igénybevétel érdekében a szoftver képes több – a felhasználó által a videóban kijelölt – objektum szimultán követésére, majd a rögzített elmozdulás-adatokat felhasználva akár szögelfordulás, vagy méretváltozás számítására. Továbbá, az implementált numerikus differenciáló algoritmusok segítségével az objektumok sebesség- és gyorsulás-értékeinek számítása is egyszerűen elvégezhető és exportálható.

A szoftver megvalósítására a Python programozási nyelvet használtam. Az objektumkövető algoritmusokat az *OpenCV* könyvtár segítségével implementáltam, míg a grafikus felület a *Qt* keretrendszerre épül, amely biztosítja a rendszer platformfüggetlenségét. Emellett a numerikus differenciálást a *PyNumDiff* könyvtárba beépített algoritmusok segítik.

Az elkészített szoftver hatékonyságát és használatát több saját készítésű videófelvétel elemzésével mutatom be. Ezen felvételek között szerepelnek mechanikai anyagvizsgálatokhoz tartozó videók, nagysebességű kamerás felvételek és emberi mozgás analíziséhez rögzített felvételek. Emellett felhasználók mesterségesen legyártott referencia videóképeket is az általam készített algoritmusok validálásához.

A kapott eredmények azt igazolták, hogy a szoftver nagy pontossággal és megfelelő sebességgel képes kinyerni a szükséges kinematikai mennyiségeket a videóképekből. Azon túl, hogy az elkészült applikációt a Műszaki Mechanikai Tanszéken folyó tudományos kutatásokban már most használják, a szoftvert ingyenesen elérhetővé tettem a kutatók számára, és egyúttal tudományos publikációban is ismertetni tervezem.

Kulcsszavak: motion-tracking, mozgáskövetés, anyagvizsgálat, Python, OpenCV, kísérleti mechanika

Abstract

The continuous development of the recent years had a significant impact on optical devices. Nowadays, as the quality of pictures made by small mobile devices are improving, the price of professional, high speed video cameras are also decreasing. Consequently, measurements with dedicated sensors are getting replaced with video-based motion analysis in some experiments. Despite the increasing popularity, this method also has major drawbacks: the data processing can be cumbersome and excessively time consuming. As the use of professional motion-tracking software comes with hefty license prices, there are many researchers who cannot afford it. Another issue often raised is the customization options of these application, which cannot be compared the "tailor-made" image processing scripts, developed by the engineers.

This work presents a self-made, free and open-source motion-tracking software which can satisfy the needs of the researchers with its accuracy and rich selection of implemented functions. Thanks to the graphical user interface, the application can be easily used without prior programming knowledge. To support versatile utilization, the software is capable of tracking multiple objects – selected by the user – simultaneously. From the tracked position data, various properties can be calculated such as rotation between two objects or the change of size. Thanks to the implemented numerical differentiation algorithms, velocity and acceleration level data can also be effortlessly obtained and exported.

The software is written in the Python programming language. The object tracking algorithms are implemented using the *OpenCV* library and the graphical interface is based on the *Qt* framework, to provide a platform independent solution. Furthermore, numerical differentiation algorithms from the *PyNumDiff* package are also utilized for the post processing.

The usage and performance of the developed software is presented with the analysis of self-made video recordings. These include videos of mechanical material tests, high speed camera recordings and videos recorded for human motion analysis. Moreover, artificial reference videos are also used for the validation of the tracking algorithms.

The results prove that the software is capable of extracting the needed kinematic quantities from the videos accurately. Beyond the fact that the application is already in use at the Department of Applied Mechanics, the software presented in this work is completely open-source and available for any researcher interested. In the foreseeable future, a scientific publication about the software is also planned.

1. Bevezetés

A képelemzésen alapuló adatgyűjtés napjaink egyik legnépszerűbb módszere. A kamerák folyamatos fejlődése ugyanis lehetőséget teremtett arra, hogy ezek az eszközök széles körben el tudjanak terjedni, a mindennapi használattól kezdve a professzionális környezetben történő alkalmazásig. Manapság a mérnökök egyre több helyzetben használnak kamerákat a dedikált szenzorok helyett, majd ezek felvételeit felhasználva, képelemzési eljárásokkal jutnak a szükséges információkhoz. Ezeknek a képfeldolgozási eljárásoknak egyik népszerű ága a mozgáskövetés, vagyis a „motion-tracking”. Ezen eljárás során videóknak kijelölt objektumok pozíciójának időbeni változását tudjuk kinyerni, amelyet felhasználva rengeteg információ határozható meg a vizsgált tárgy viselkedéséről.

Az objektumkövetés egy speciális esete a „motion-capture”, amikor dedikált infravörös kamerákat és úgynevezett reflektív markereket használnak a követendő tárgyak megjelölésére. A kamerák számának függvényében így egy adott méretű térben képesek a mozgások háromdimenziós, nagy pontosságú rögzítésére, valós időben. Ennek a technológiának alkalmazása manapság leginkább a szórakoztatóiparban [29] elterjedt, de találunk sok egyéb példát [30], a biomechanikától [32] kezdve, a robotikai alkalmazásokig [12]. Ezeknek a „motion-capture” rendszereknek ugyanakkor hatalmas hátrányaik is vannak. Egyrésztől alkalmazásuk egy fix helyhez kötött, hordozásuk és kalibrálásuk nehézkes, valamint a mérésre használható tér is előre determinált. Másik számottevő negatívum ezen rendszerek ára, amely a több tízezer dolláros nagyságrendbe esik, egy alap struktúra kiépítése esetén is. Éppen ezért az elmúlt évtizedben egyre több törekvés volt az olyan képfeldolgozási eljárások kifejlesztésére, melyek során nincs szükség különleges kamerák és összetett mérőrendszerek alkalmazására.

Ezen dolgozat a „motion-tracking” azon ágával foglalkozik, amely egyszerű, mindennapi eszközök, illetve esetlegesen nagysebességű videokamerák által készített felvételek elemzését tűzte ki célul. Noha ez a terület általában önmagában kétdimenziós esetekre korlátozódik, számtalan felhasználási mód ismert. A különböző mechanikai kísérletek, például anyagvizsgálatok [41] és rezgésvizsgálatok [31] mellett biomechanikai mérések esetén is alkalmazott ez a módszer. A széleskörű felhasználás eredményeképpen a mérnökök körében rengeteg módszer alakult ki a mozgáskövetés kivitelezésére. Az egyik ilyen módszer professzionális videó vágó programok alkalmazása, melyek beépített „motion-tracking” moduljának segítségével kinyerhetők a kívánt pozícióadatok. Ezt a technikát alkalmazták többek között állatok mozgásának vizsgálatára [24], ahol a kutatók az Adobe After Effects programot egészítették ki saját fejlesztésű kóddal. Emellett volt olyan tanulmány [35] is, mely során egy iPhone készülékkel és rá elérhető applikációk segítségével vizsgálták zebrahalak viselkedését. A legtöbb esetben viszont a kutatók saját fejlesztésű, az adott feladatra specializálódott szoftverekkel igyekeztek megoldani az adatgyűjtés problémáját. Bár ez a módszer az esetek többségében megfelelően bizonyult, egy időigényes és összetett folyamat, melyhez elengedhetetlen a megfelelő programozási előismeret.

Mélyebben áttekintve a releváns szakirodalmat hamar rájövünk, hogy amennyiben nem

szeretnénk saját fejlesztésű kódot készíteni, lehetőségeink a mozgáskövetésre igen behatóra voltak. A mozgáskövetéshez jelenleg elérhető szoftvereket vizsgálva több csoportot különböztethetünk meg. Léteznek egyrészt az alapvetően videó vágási célokra szánt szoftverek, mint az Adobe After Effects, a Final Cut Pro, Adobe Premiere Pro. Ezeknek a szoftvereknek azonban meglehetősen magas licenszdíjai vannak és használatuk is igen körülményes, hiszen alapvetően más feladatok ellátására készültek. Bár vannak nyílt forráskódú és ingyenes alternatívák, mint például a Blender vagy a DaVinci Resolve, ezeknél viszont ugyanúgy fennállnak a korábbi problémák, lévén videó vágó programokról van szó. Másik csoport az alapvetően analízis célokra készült szoftverek. Itt is elérhetők professzionális, kereskedelmi megoldások úgy, mint a Xcitex ProAnalyst, a Vicon Motus ezeknél azonban szintén probléma a magas ár és a zárt forráskód, ami kevés teret hagy a testreszabhatóságra. Bár a bemutatott programoknak szintén ismert nyílt-forráskódú ingyenes alternatívája úgy, mint a Tracker, de ennek a szoftvernek a funkcionalitása meglehetősen hiányos. Harmadik csoportba azokat a szoftvereket sorolnám, melyek valamilyen biológiai alkalmazási területre lettek kifejlesztve [34]. Itt beszélhetünk olyan szoftverekről, mint a SwarmSight [4] vagy a ToxTrack [37], amelyek etológiai vizsgálatokat, különböző állatok csoportos mozgásának megfigyelését teszi lehetővé. De léteznek különböző alapvetően sportelemzésre és edzéstudományra készült programok is, mint például a Kinovea vagy a PhysMo, ezek mozgáselemző funkciói azonban eléggé specifikusak. Végül, de nem utolsósorban ismertek kód alapú megoldások is, mint például a Tracktor [39] vagy pathtrackr [18], amelyek olyan Python, illetve R programozási nyelv alapú könyvtárakat kínálnak, amik segítségével könnyedén kivitelezhető a „motion-tracking” a megfelelő testreszabhatóság mellett. Hátránya azonban, hogy programozási ismeretek nélkül ezen megoldások sem használhatóak.

Ebből a rövid összefoglalásból jól látszik, hogy a számos lehetőség ellenére jelenleg nincsen egy olyan, nyílt forráskódú, ingyenesen elérhető és grafikus felhasználói felülettel rendelkező mozgáskövető szoftver, amely ki tudná elégíteni a kutatók és a mérnökök igényeit precizitásával és funkcionalitásával. Dolgozatomban ezért egy olyan saját fejlesztésű „motion-tracking” szoftvert mutatok be, ami kimondottan mérési és elemzési célokra készült, különös tekintettel a mechanikai kísérletekre, melyekre a korábban ismertetett programok igen kevés figyelmet fordítottak. A fejlesztett szoftvernek fontos célja volt, hogy egy könnyen használható, programozási előismeretet nem igénylő, platformfüggetlen lehetőséget kínáljon, melyben kompromisszumok kötése nélkül egy applikáción belül elvégezhető a mozgáskövetéshez kapcsolódó feladatok. Emellett, a nyílt forráskód révén testreszabhatósági lehetőséget is biztosítson, amennyiben az elérhető funkciókon bővíteni kívánnak a jövőbeni felhasználók.

Hogy ezeket az elvárásokat maradéktalanul ki lehessen elégíteni, a szoftver Python programozási nyelven készült. Ez amellett, hogy jelenleg napjaink egyik legnépszerűbb programozási nyelve, számos előnnyel jár: biztosítja a platform-függetlenséget, egyszerű telepítési és frissítési megoldásokat kínál a PyPI repozitórium segítségével. Továbbá utóbbinak köszönhetően számos más fejlesztők által készített szoftvercsomag elérhető, amivel

az implementáció jelentősen gyorsítható. Ebben a dolgozatban is jelentős szerepet kapnak ezek a könyvtárak. Az *OpenCV* [6] Python csomagja segítségével valósultak meg a szoftver képfeldolgozási eljárásai, de emellett fontos szerepe volt a *PyQt5* [25] keretrendszernek is amely a grafikus felület létrehozásához biztosította az alapokat. Az adatfeldolgozási eljárások gyorsítására a *pynumdiff* [7] és a *pandas* [28] [40] szoftvercsomagok kínálnak megoldásokat, míg a grafikonok készítése a *matplotlib* [21] könyvtár felhasználásával lett implementálva.

Jelen dolgozat öt fő fejezetre tagolható. A bevezetés után, a második fejezetben a szoftver működéséhez szükséges képfeldolgozási, valamint adatelemzési algoritmusokat mutatom be. Itt kitérek az *OpenCV* könyvtár adta lehetőségekre, valamint sebesség illetve gyorsulásadatok számításához szükséges numerikus differenciálás problémájára és megoldására. Ezt követően a harmadik fejezetben a szoftver felépítését, valamint az alapvető használat lépéseit ismertetem. A szoftver működésének bemutatását a negyedik fejezetben található alkalmazási példákon keresztül prezentálom. Itt fontos szerepet kap először a kapott eredmények validációja, mesterségesen előállított videók segítségével. Végül az eredményeket az ötödik fejezetben fogalom össze.

2. Mozgáskövető és kiértékelő algoritmusok

Ebben a fejezetben a mozgáskövetés megvalósításához használt algoritmusok elméleti hátterét, illetve alapjait mutatom be röviden. Mivel a fejlesztés során az *OpenCV* könyvtárat, használtam, így az ott elérhető algoritmusok kerülnek ismertetésre. A képfeldolgozó eljárások után a következő pontban a numerikus differenciálás alapvető problémáit és megoldásait prezentálok. Ezen algoritmusok alkalmazása szintén elengedhetetlen, amennyiben a „motion-tracking” által kinyert pozíció adatokat felhasználva sebesség, illetve gyorsulás adatokat szeretnénk kinyerni. A deriválás implementációjához a *pynumdiff* szoftvercsomag által kínált funkciókat használtam, így ezen dolgozat keretein belül, az itt elérhető algoritmusok kerülnek bemutatásra.

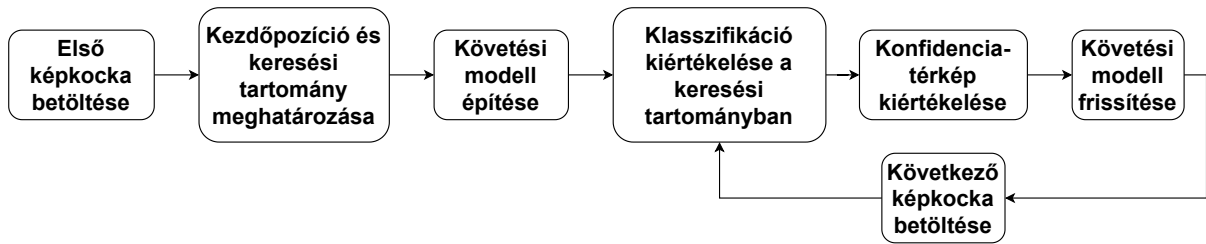
2.1. Mozgáskövetés

Az objektumok követése a gépi látás egyik legalapvetőbb témája. A széleskörű alkalmazási lehetőségeknek köszönhetően, mind ipari, mind kutatási környezetben nagy népszerűségnek örvend. Az legnagyobb nyílt forráskódú, képfeldolgozó eljárások segítségével kidolgozott könyvár, az *OpenCV* számos népszerű és hatékony mozgáskövető algoritmus implementációját magában foglalja, így a fejlesztőknek lehetőségük van egyszerűen egy egységes interfészen keresztül használni ezeket. Jelen dolgozat esetében én is ezt a módszert választottam, így a következőkben az *OpenCV* könyvtár „motion-tracking” algoritmusait mutatom be [36].

2.1.1. BOOSTING algoritmus

A BOOSTING egy valós idejű objektumkövető módszer, melynek az alapja az AdaBoost algoritmus [16]. Ez az algoritmus tulajdonképpen egy bináris klasszifikációt végez a kijelölt objektum és a háttér közt, így meghatározva annak pozícióját, képkockáról-képkockára. Bár ez a megoldás már korábbi módszerekből is ismert [1], azonban a jelen algoritmus képes az osztályozáshoz használt *classifier*-ek valós idejű frissítésére, a robusztusság és a pontosság növelése érdekében.

Az BOOSTING technika működésének blokkdiagramja a 2.1 ábrán látható. A kezdeti lépés a felhasználó által, az első képkockán kijelölt területet, mint pozitív mintát határozza meg az algoritmus. Ezzel párhuzamosan a kijelölt területtel megegyező méretű régiókat, mint negatív mintákat is választ a kijelöléshez közeli háttérből. Ezt követően egy valós idejű BOOSTING algoritmust használva, iterációs eljárásan keresztül, egy stabil modellt hoz létre az objektumról. Ezután az objektumkövetés következik. Az újonnan betöltött képkockán kiértékelésre kerül a létrehozott *classifier*, az objektum előző pozíciójához közeli régióban (*region of interest*). A kiértékelésből kapott konfidenciatérkép segítségével ezután, annak maximum lokációjába mozdul el az objektum pozíciója. Ezt követően a *classifier* frissítésre kerül az új képkocka felhasználásával, hogy az algoritmus hatékonyan



2.1. ábra. A BOOSTING algoritmus blokkdiagramja.

tudjon alkalmazkodni a mozgó objektum és a környezet esetleges változásaihoz. Majd a frissített *classifier* felhasználásával a következő iteráció elindulhat az új betöltött képkockával. Az iteráció során felhasznált valós idejű AdaBoost algoritmust részletesen tárgyalja [15] és [16], ezért jelen dolgozat keretében nem kerül bemutatásra.

Annak ellenére, hogy a BOOSTING algoritmus már több, mint egy évtizede került először bemutatásra, a pontossága ma is megfelelő. Ugyanakkor újabb algoritmusok, különösen a számítási időt tekintve, felülmúlják ezen mozgáskövető módszer hatékonyságát. További hátránya ezen felül, hogy az algoritmus nem képes felismerni azt, ha „eltévedt”, vagyis nem képes megszakítani magát az objektum elvesztése esetén. Ez nagyobb fájlok és hosszabb videók feldolgozásánál kellemetlenséget jelenthet.

2.1.2. MIL algoritmus

A MIL, hosszabb nevén „Multiple Instances Learning” algoritmus megközelítése az előző fejezetben tárgyalt BOOSTING-hoz hasonló, hiszen ez a megoldás is egy modell felépítésével, majd annak felismerésével valósítja meg az objektum detektálását [2]. Azonban a modell felépítésére egy adaptív, tanulás alapú módszert használ.

Az objektum modell felépítése pozitív és negatív minták kijelöléséből áll. Általában a pozitív mintát a mozgáskövető algoritmus aktuális helyzete, negatív mintát pedig a környező terület alkotja. Azonban, ha az objektum helyzete nem ismert pontosan, vagy eltérés van a kijelölt és a valós pozíció között, úgy az algoritmus által használt modell pontatlan lesz, ami követés „elcsúszásához” vezethet. Abban az esetben viszont, ha a modell felépítéséhez több pozitív mintát használunk a környező régióból az algoritmus nehezebben tudja elkülöníteni a háttérrel az objektumtól. Ezért, [2] egy módosított megoldást mutat be az objektum azonosítására. Ennek a tanulási módszernek az alapja, hogy a tanítási folyamat alatt, minták egy megcímkézett halmaza kerül átadásra az algoritmusnak, melyben több potenciálisan pozitív megoldás van. Ez a gyakorlatban úgy néz ki, hogy az objektum kijelölt pozíciójának egy adott sugarában több lehetséges területet határoz meg az algoritmus, majd a tanuló szakasz ezek közül választja ki a legmegfelelőbb megoldást. Így, jelentősen javul az alkalmazkodóképesség, mivel az algoritmus rugalmasan vázoltathatja a vizsgált területet, amelyben az objektum található.

Az objektumkövetés konkrét lépései szakaszokra bonthatók, melyet a 2.2 ábra mutat

be. Az új képkocka betöltése után (1) a következő lépésben (2) lehetséges megoldásokat jelölünk ki, az objektum környezetéből. Ezen képrészek halmazát a következőképpen lehet definiálni:

$$X^s = \{x | s > ||l(x) - l_{t-1}^*||\}, \quad (2.1)$$

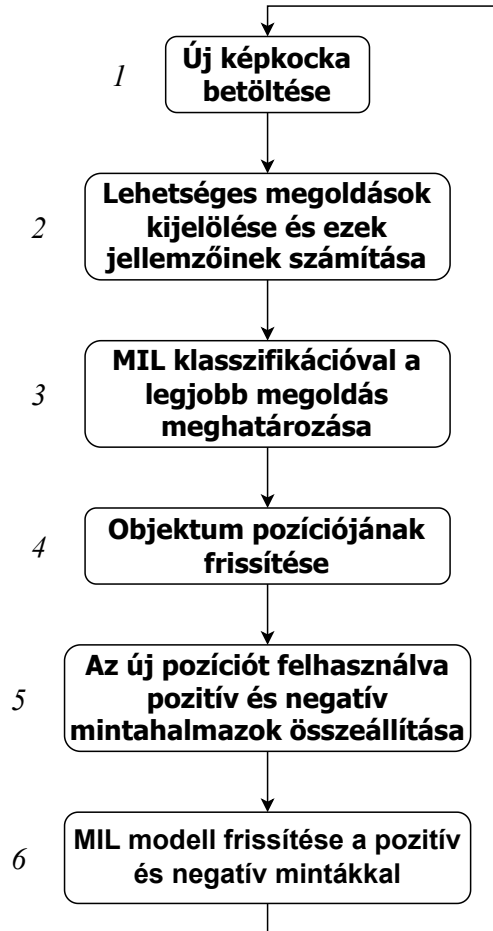
ahol $l(x)$ az x képrészlet lokációja, l_{t-1}^* az objektum pozíciója az előző iterációs lépésben, s pedig a keresési sugár. A halmaz elemein ezután MIL modell segítségével klasszifikációt végzünk (3). A használt diszkriminatív *classifier* a következőképpen jelölhető:

$$p(y = 1|x), \quad (2.2)$$

ahol y bináris változó jelöli, hogy az adott objektum megtalálható-e x képrészletben. Ezt követően egy mohó algoritmus [8] segítségével frissíthető az objektum pozíciója (4):

$$l_t^* = l \left(\arg \max_{x \in X^s} p(y|x) \right). \quad (2.3)$$

Az új pozíciót felhasználva, az objektum környezetéből képrészletek kivágásával pozitív (X^r) és negatív minták ($X^{r,\beta}$) halmazát lehet előállítani(5). Ezen halmazokat felhasználva a MIL modell frissíthető (6), amit egy újabb iteráció követ.



2.2. ábra. A MIL objektumkövető algoritmus lépései.

A MIL probléma megoldására, egy valós idejű boosting alapú algoritmust fejlesztettek ki, amelyet [2] részletesen bemutatja.

Bár a MIL algoritmusnak alapvetően kevesebb kezdeti tanítási információ áll rendelkezésére, [43] megmutatja, hogy az algoritmus képes felülmúlni a korábbi megerősítéses tanulás alapú klasszifikációs objektumkövetőket, melyeket explicit területekből álló mintákkal tanítottak.

Az algoritmus fő előnye, hogy nagyobb robusztusságot mutat zajjal terhelt felvételek esetén, miközben a pontossága is megfelelő. Ugyanakkor kiemelendő, hogy az algoritmus sebessége relatív alacsony más módszerekkel összehasonlítva. Emellett azt is fontos megjegyezni, hogy a BOOSTING algoritmushoz hasonlóan, nem képes futás közben megszakítani magát és jelezni, ha elvesztette az objektumot.

2.1.3. MEDIANFLOW algoritmus

A MEDIANFLOW algoritmus [22] egy olyan objektumkövető megoldás, melynek kidolgozása során nagy hangsúlyt fektettek a hibadetektálásra. Magának a követésnek az alapja a Lucas-Kankade-féle módszer [26], amely egy *optical flow* alapú technika szerint tudja meghatározni adott pixelek elmozdulását. Ez az információ azonban önmagában használhatatlan az objektumkövetésre, így az itt bemutatott MEDIANFLOW algoritmus, egy olyan funkcióval egészül ki, mely képes a követési hibák detektálására, valamint megbízható trajektóriák szelektálására.

A bemutatott módszer lényege, hogy az objektumkövetés időben mind előre, mind pedig hátra megtörténik minden pillanatban. Ennek segítségével az elmozdulásokat leíró pályák hibája számítható, és a megbízható, pontos trajektóriák kiválaszthatók. A hiba számításának lépései a következőképpen írhatók le. Definiáljuk a kövenerő képkockák sorozatát, mint $S = [I_t, \dots, I_{t+k}]$, az adott pont pozícióját pedig mint $\mathbf{x}_t$ a t kezdeti időpillanatban. Első lépésben az időben előre az I_t képkockától az aktuális I_{t+k} képkockáig minden lépésben meghatározása kerül $\mathbf{x}_t$ jövőbeli pozíciója, amely előállít egy trajektóriát:

$$T_f^k = [\mathbf{x}_t, \dots, \mathbf{x}_{t+k}]. \quad (2.4)$$

Ezt követően a lokalizáció visszafelé I_{t+k} -tól I_t -ig is megtörténik ami egy újabb trajektóriát állít elő, időben hátrafele:

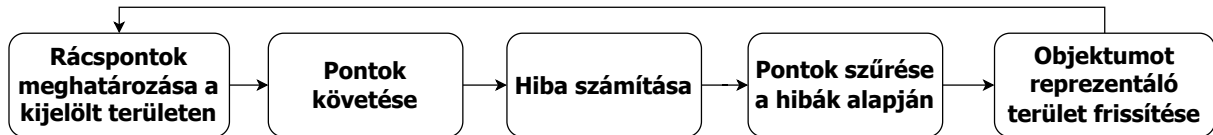
$$T_b^k = [\hat{\mathbf{x}}_t, \dots, \hat{\mathbf{x}}_{t+k}]. \quad (2.5)$$

A hibát ezt követően az euklideszi távolság meghatározásával lehet kiszámítani a validáció végpontjában:

$$FB(T_f^k|S) = \text{dist}(T_f^k, T_b^k) = \|\mathbf{x}_t - \hat{\mathbf{x}}_t\|. \quad (2.6)$$

A hiba számítás ismeretében a következő lépés a követés ismertetése, melynek folyamatábrája a 2.3 ábrán látható. Ellentétben a korábban bemutatott módszerekkel a Lucas-Kankade alapú megoldás pontok elmozdulását és nem teljes régiók pozíció változását

követi. A hatékony követés érdekében ezért az algoritmus először a kijelölt területet berácsozza és potenciális követendő pontokat választ. Ezt követően a pontok elmozdulásának számítása következik. Az elmozdulás adatokból a korábbiakban bemutatott módszerrel minden pontra kiszámítódik a trajektória hibája. A hiba mértéke alapján a pontok 50%-a kiszűrésre kerül, majd a szűrt pontokat felhasználva az objektumot magában foglaló téglalap frissül. A téglalap meghatározása elmozdulás vektorok minden dimenziójában elvégzett medián számításon alapul. Amennyiben azonban a követés során meghatározott trajektóriák hibája túl nagy, a követés leáll.

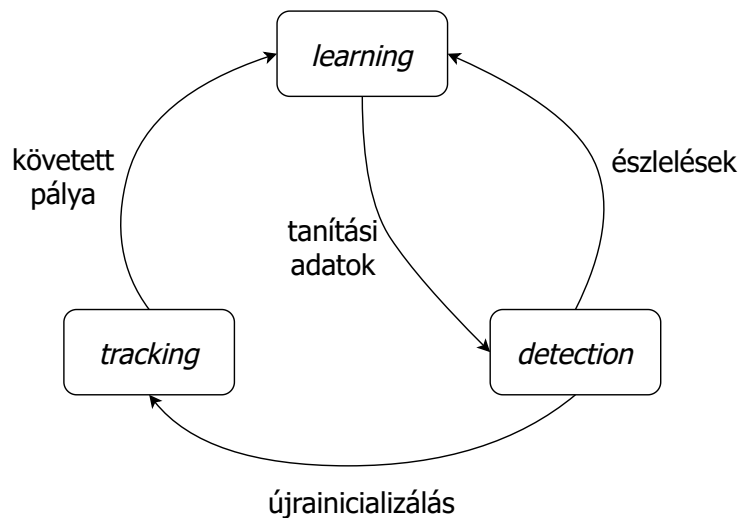


2.3. ábra. A MEDIANFLOW objektumkövető algoritmus lépései.

A MEDIANFLOW algoritmus legnagyobb előnye amellet, hogy megfelelő pontosságot és gyors számítási időt kínál a képessége a követési hibák detektálására és jelzésére, amivel jelentős idő takarítható meg. Ugyanakkor, az alkalmazott kísérletek alapján, a követés nagy sebességgel mozgó objektumok esetén gyakran hibára fut, így ilyen esetekben más algoritmus ajánlott.

2.1.4. TLD algoritmus

A TLD egy komplex mozgáskövető keretrendszer, amelyet ismeretlen objektumok hosszú időn át történő követésére fejlesztettek ki [23]. Az TLD név a *tracking*, *learning*, *detection* szavak rövidítéséből áll össze. Ahogy ezt az elnevezés is sugallja, a mozgáskövetés feladatát ez a technika három részre osztja, ezek láthatók a 2.4 ábrán.



2.4. ábra. A TLD objektumkövetési keretrendszer blokkdiagramja [23].

A *tracking* rész felelős az objektum pozícióváltozásának képkockánkénti meghatározásáért, azzal a feltételezéssel, hogy ez a mozgás limitált határok közt marad és az objektum mindvégig látható. A *detection* rész minden képkockát függetlenként kezel, és egy objektumkeresési eljárást futtat le a teljes képen, a korábban meghatározott objektumjellemzők felismerésére. A detektor természetesen nem működik hibátlanul, fals negatív és fals pozitív eredményeket is közöl. Ezért a harmadik, *learning* rész megfigyeli mind a *tracker* és mind a *detector* eredményeit, hogy abból a felismerés hibáját becsülni tudja és tanítási példákkal tudjon szolgálni a jövőbeni eltérések elkerülésére.

A tanuló rész alapja az úgynevezett *P-N learning*, ami egy félig felügyelt tanulási módszerként írható le. Alapja, hogy két egymástól független részrendszer felelős a fals pozitív és a fals negatív hibák felismeréséért. Ennek a technikának a lényege, bár mind a két modul képes saját maga is hibás becslésekre, a független működésnek köszönhetően képesek kompenzálni egymást. A P-N tanulás folyamatát és részleteit [23] ismerteti.

A TLD követő keretrendszer legfőbb előnye, hogy a felvételeken képes az olyan objektumok követésére is, amelyek néha takarásba kerülnek egyéb zavaró tényezők által. A detektor modul ugyanis képes az újra megjelenő objektumok azonosítására, így a követés nem szakad meg. Hátránya viszont, hogy a detektor a tanulási módszerek ellenére sokszor fals pozitív eredményeket szolgáltat, ha a követendőhöz hasonló objektumok találhatók a felvételen. Ezáltal a követés instabillá válhat.

2.1.5. MOSSE algoritmus

A MOSSE egy adaptív korrelációs szűrő alapú mozászkövető algoritmus [5], amely megfelelő robosutusságot mutat válzotó megvilágítású, deformálódó és forgó objektumok esetén is. A korrelációs szűrés alapú mozgáskövető technikák során az objektum detektálására szűrőket használnak, amelyek a követendő objektum modelljét mintaképek segítségével tanulják meg.

A MOSSE blokkdiagramja a 2.5 ábrán látható. Első lépésben az algoritmus inicializálása történik, egy kis keresési ablak segítségével az objektum körül. Ezután a következő betöltött képkockán egy adott keresési ablakon belül a korrelációs szűrő kiértékelése következik: az eredmények közül a maximális értékhez tartozó lokáció lesz az objektum új pozíciója. Annak érdekében, hogy az algoritmus kellően gyors legyen, a korreláció a Fourier tartományban kerül kiértékelése, mivel itt a konvolúciós tétel értelmében a korreláció számítása egyszerű elemenkénti szorzás művelettel számítható. Legyen $F = \mathcal{F}(f)$ a betöltött kép, $H = \mathcal{F}(h)$ a szűrő két dimenziós gyors Fourier transzformációval (FFT) meghatározott értéke. Ekkor

$$G = F \circledast H^*, \quad (2.7)$$

ahol G a korreláció Fourier tartománybeli értéke, $\circledast$ a konvolúció műveleti jele és $*$ pedig a komplex konjugáltat jelöli. A korreláció meghatározása után inverz FFT alkalmazásával számítható a térbeli korrelációs eloszlás. Ennek a módszernek köszönhetően az algoritmus

bonyolultsága legfeljebb $O(P \log P)$, ahol P a keresési ablak pixeleinek száma. Ez gyors futási sebességet tesz lehetővé a korábbiakban bemutatott technikákhoz képest.

A megfelelő performancia érdekében fontos a számításhoz használt szűrő modelljének megfelelő inicializálása, valamint frissítése. Ezért a MOSSE algoritmus a szűrőt egy optimalizációs eljárás során határozza meg: a valós korrelációs eredmény és a modell által kapott eredmény négyzetes hibáját minimalizálja. Ennek a formalizmusa

$$\min_{H^*} \sum_i |F_i \circledast H^* - G_i|^2, \quad (2.8)$$

ahol i a tanításhoz felhasznált minták (kezdeti képkockák képkockák) száma. Az optimalizáció megoldása zárt alakban megadható:

$$H^* = \frac{\sum_i G_i \circledast F_i^*}{\sum_i F_i \circledast F_i^*}. \quad (2.9)$$

Az inicializálás után fontos még megvizsgálni az algoritmus alkalmazkodóképességét adó modellfrissítés metódusát. A MOSSE szűrő esetén a szűrő modellje a következő egyenletek szerint frissül, minden egyes iterációban:

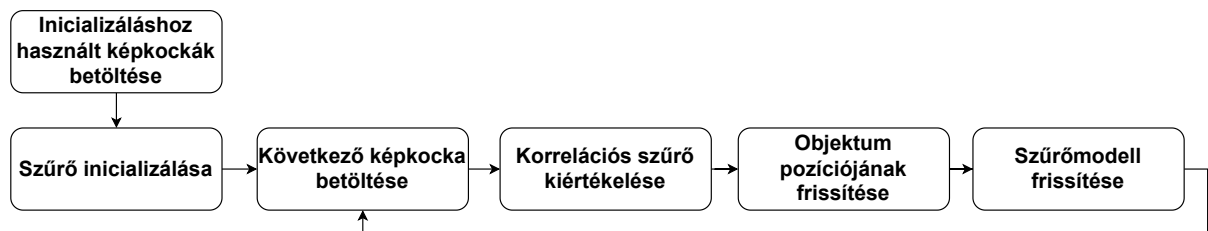
$$H_i^* = \frac{A_i}{B_i}, \quad (2.10a)$$

$$A_i = \eta G_i \circledast F_i^* + (1 - \eta) A_{i-1}, \quad (2.10b)$$

$$B_i = \eta F_i \circledast F_i^* + (1 - \eta) B_{i-1}, \quad (2.10c)$$

ahol η a tanulási ráta A és B a modellhez tartozó kumulatív összegek és i jelzi az aktuális iteráció számát. Gyakorlatban [5] a $\eta = 0.125$ értéket találta a legoptimálisabbnak.

A MOSSE legnagyobb előnye a korábban bemutatott módszerekkel szemben sokkal nagyobb futási sebességre képes és az implementációja is meglehetősen egyszerű. Ugyanakkor hátrányként felhozható, hogy hiba detektálásra ez az algoritmus sem képes, valamint a pontosságát egyéb korrelációs szűrő alapú megoldások felülmúlják.

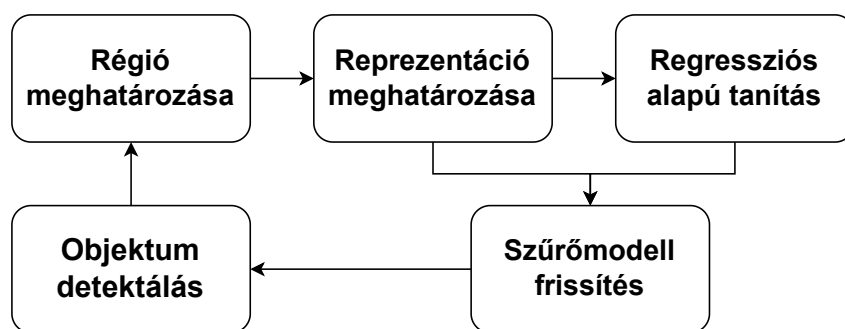


2.5. ábra. A MOSSE szűrő blokkdiagramja.

2.1.6. KCF algoritmus

A KCF [19], hosszabb nevén „Kernelized Correlation Filter” a korábbiakban bemutatott BOOSTING és MIL módszereket és a korrelációs szűrő alapú objektumkövetés technikáját kombinálja.

A KCF algoritmus blokkdiagramját a 2.6 ábra mutatja be. Első lépésben a követendő objektum manuális kijelölése következik. Ezt követően a KCF algoritmus a korábban bemutatott MIL módszer szerint több lehetséges pozitív mintát jelöl ki az objektum környező területéről. Kihasználva azt, hogy az így kijelölt minták között jelentős átfedés tapasztalható, egy analitikus modell hozható létre, keringő mátrixok segítségével [10]. Ezen keringő mátrixok diszkrét Fourier transzformációval diagonalizálhatók, így a tároláshoz szükséges számítási és tárhely igény jelentősen csökkenthető. A Fourier tartományban, úgynevezett lineáris rigde regresszió [17] alapján történik egy speciális, kernel alapú [19] modell tanítása, valamint frissítése. Ezt követően a szűrő kiértékelhető az újonnan betöltött képkockán, amely egyben az objektum új pozícióját is meghatározza és újratekinthető az iteráció. Az algoritmus során alkalmazott tanítási és detektálási módszereket [19] foglalja össze, jelen dolgozat, a terjedelemtől tekintettel, ezt nem részletezi.



2.6. ábra. A KCF algoritmus lépéseit bemutató blokkdiagram [14].

A többi korrelációs szűrő alapú megoldáshoz hasonlóan a KCF sebessége is kimagasló, bár nem éri utol a korábban bemutatott MOSSE algoritmust. Ugyanakkor cserébe annál nagyobb pontosságot kínál és képes a követés hibájának detektálására.

2.1.7. CSRT algoritmus

A CSRT az itt bemutatott, valamint a szoftverben implementált algoritmusok közül a legfrissebb és legnagyobb pontosságot nyújtó megoldás [36]. A CSRT algoritmus a diszkriminatív korrelációs szűrő alapú objektumazonosítási és követési módszert egészíti ki további elemekkel, a performancia és a pontosság növelése érdekében [27]. Az algoritmus alapvető felépítése a 2.7 ábrán látható.

Az egyik legfontosabb kiegészítő elem, amelyre a CSRT módszer épít, a térbeli megbízhatósági térkép alkalmazása. Ez a megbízhatósági térkép képes az objektumhoz tartozó képrészletek megbecsülésére, így a tanításra alkalmas régió kiterjeszthető, valamint a nem téglalap alakú objektumok követése is megbízhatóan kivitelezhető. A másik fontos része az algoritmusnak csatornánkénti megbízhatósági szintek definiálása. Ennek köszönhetően egy többcsatornás, különböző jellemzőkre specializált, komplex korrelációs szűrő tervezhető, mely a csatornák külön eredményeit a megbízhatóság szintek szerint súlyozza. A CSRT

algoritmus matematikai formalizmusát [27] mutatja be, jelen dolgozat a terjedelemre való tekintettel erre a témára nem terjed ki.



2.7. ábra. A CSRT algoritmus felépítése [27].

A CSRT legnagyobb előnye a kimagasló pontosság a korábbi algoritmusokhoz hasonlítva. Emellett megfelelő robusztusságot mutat a zavaró hatások, valamint esetleges átlapolódások esetén is. Az egyetlen negatívum, amit fel lehet hozni a futási sebessége, mely alulmarad nem csak a korrelációs szűrő alapú megoldásokkal, de egyéb, korábban bemutatott algoritmusokkal szemben is.

2.2. Numerikus differenciálás

Bár a fentebb bemutatott algoritmusok képesek a nagy pontosságú pozíciómeghatározásra, a mérnököknek sok kísérlet esetében további, sebesség, illetve gyorsulás adatokra is szüksége lehet. Ezen adatok kiszámításához a szoftver numerikus módszereket használ, melyek lehetővé teszik a pozíció deriváltjainak becslését, vagyis a sebességet és a gyorsulást.

Ebben a fejezetben három különböző megközelítést [7] mutatok be a numerikus differenciálás kivitelezésre. Első lépésben a véges differencia alapú technikákra és az ezeket kiegészítő szűrési eljárásokra térek ki. Ezt követi a lokális modell illesztésére épülő számítási mód, majd a *total variation regularization* alapú optimalizációs derivált becslés.

2.2.1. Véges differencia módszerek

A véges differencia számítása a legegyszerűbb módszer a numerikus differenciálásra [9]. Amennyiben ismert a pozíció két egymást követő időpillanatban, a derivált első rendű

becslése

$$\dot{x}^b[k] = \frac{x[k] - x[k-1]}{\Delta t}, \quad (2.11)$$

ahol k index jelöli a mintavételi pontokat és Δt a mintavételek közt eltelt időt. Ez az úgynevezett *backward finite difference*, vagyis $\dot{x}$ meghatározásához az aktuális és az egyvel korábbi időpillanatban ismert értéket használtuk fel. Emellett természetesen lehetőség van az indexek elcsúsztatásával a *forward finite difference* ($\dot{x}_k^f$) kiszámítására, valamint a kettő ötvözése után a *central finite difference* ($\dot{x}_k^c$) meghatározására is:

$$\dot{x}^f[k] = \frac{x[k+1] - x[k]}{\Delta t}, \quad \dot{x}^c[k] = \frac{x[k+1] - x[k-1]}{2\Delta t}. \quad (2.12a)$$

A legnagyobb probléma ezen algoritmusok alkalmazásakor a mérési pontatlanság. Bár az objektumkövető algoritmusok precízen működnek, a rögzített értékek zajjal terheltek. Ezt a zajt a numerikus differenciálás kiemeli, ami pontatlan, illetve a későbbiekben használhatatlan eredményekhez vezethet.

A hibák a csökkentése, illetve ellensúlyozása elérhető a mért pozícióadatok differenciálás előtti simításával, melyet különböző algoritmusok segítségével valósíthatunk meg.

A legegyszerűbb megoldás egy egyszerű **mozgóátlag-szűrő** alkalmazása:

$$\tilde{x}[k] = \frac{1}{M} \sum_{i=0}^M x[k+i], \quad (2.13)$$

ahol $\tilde{x}$ a simított jel, x a mért jel és M a mozgóátlag ablakmérete. Ha jobban megfigyeljük, ez a művelet tulajdonképpen egy dimenziós diszkrét idejű konvolúcióként értelmezhető, ahol a konvolúcióhoz használt kernel vektor M hosszúságú és elemeinek értéke $1/M$. Ennek a kernelnek a módosításával egyszerűen lehet előállítani más szűrőalgoritmusokat is, úgy mint a **Gauss-szűrőt** vagy a **Friedrichs-szűrőt**. Emellett természetesen egyéb szűrési és simítási módszerek is alkalmazhatók, úgy mint a **Butterworth-szűrő** vagy a **Spline-szűrő**, melyek implementációját [7] ismerteti.

2.2.2. Modell alapú módszerek

Ezen módszerek első lépésben egy lokális modellt építenek fel, az adathalmaz egy szakaszán, lineáris regresszió segítségével [42]. A modell ismeretében pedig a deriváltak számítása analitikusan is történhet.

A modell meghatározására az egyik legegyszerűbb mód a polinomillesztés, egy eltolt ablak segítségével, ez az úgynevezet **LOESS algoritmus** [13]. Hatékonyabb lehet azonban, ha polinom illesztése a frekvencia tartományban történik, erre képes a **Savitzky-Golay szűrő** [38]. Mivel a csúszó ablak miatt az egymást követő polinomok eltérhetnek, a deriváltban nagy ugrások jelennek meg, így a kapott eredményt utólagosan simítani érdemes.

Az itt kiemelt algoritmusokat, valamint további módszereket [7] részletesen tárgyalja.

2.2.3. Total variation regularization

A *total variation regularization* egy optimalizáció alapú eljárás a deriváltak becslésére [3] [42]. Itt a differenciálási feladat megoldásához egy költségfüggvény kerül definiálásra, melyet különböző megoldóprogramokkal minimalizálva kinyerhetők a derivált értékek. Az optimalizáció költségfüggvénye

$$L = ||x - \hat{x}||_2 + \gamma * TV(\dot{\hat{x}}), \quad (2.14)$$

ahol $\hat{x}$ a pozícióadatok becsült simított értéke, γ egy optimalizációs paraméter. A $TV(\dot{\hat{x}})$ a totális variáció:

$$TV(\dot{\hat{x}}) = \frac{1}{m} ||\dot{\hat{x}}_{0:m-1} - \dot{\hat{x}}_{1:m}||_1, \quad (2.15)$$

ahol m az adatok számát jelöli. Ezen az optimalizációs feladat eredményét iteratív megoldóprogramokkal lehet meghatározni [3], melyek a Python környezetben is implementálva vannak [11].

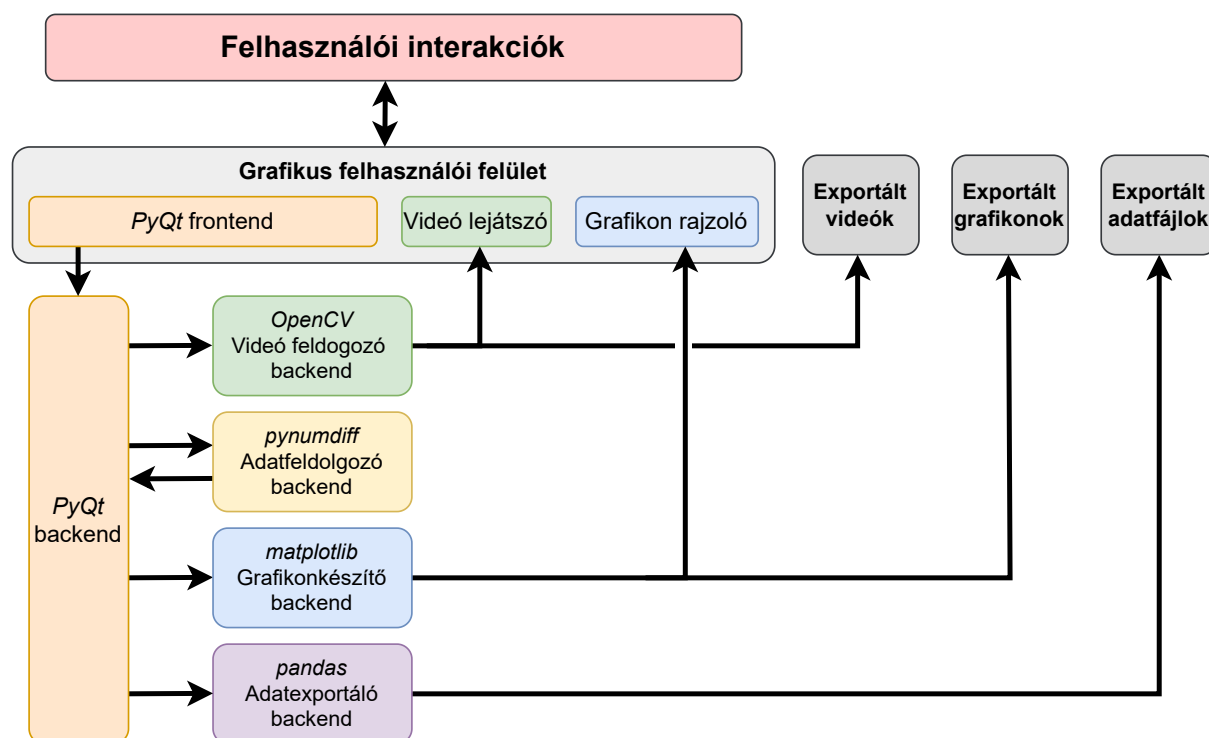
Nagy előnye az eljárásnak, hogy a számításhoz egyetlen külső paramétert kell megadni (γ), ezzel az értékkel állítható az eredmények simítása. Minél nagyobb a γ , annál erősebben igyekszik az optimalizáció elnyomni a differenciálás zaját, így ez a módszer egyszerű hangolást tesz lehetővé.

3. Szoftver felépítése

A dolgozat ezen fejezte az elkészített szoftver felépítésének és használatának bemutatásával foglalkozik. Először a felépített szoftver architektúra kerül bemutatásra kiemelve a legfontosabb komponenseket. Ezt követően pedig a applikáció funkcióit mutatom be, egy egyszerű mintapéldán keresztül.

3.1. Szoftverarchitektúra

Az applikáció három fő részre bontható: grafikus felhasználói felület, mozgáskövetésért és elemzésért felelős funkciók, valamint eredmények megjelenítéséért és exportálásáért felelős programrészek. Az elkészült szoftver felépítését a 3.1 ábra mutatja.



3.1. ábra. A szoftver felépítése, a legfontosabb részegységekre bontva.

3.1.1. Grafikus felhasználói felület

A sokoldalú felhasználás érdekében egyértelmű volt, hogy a fejlesztett objektumkövető alkalmazásnak rendelkeznie kell grafikus felhasználói felülettel. Ennek a legegyszerűbb előnye, hogy így lehetővé válik a szoftver alkalmazása olyan kollégák számára is, akik ugyan szeretnének mozgáselemzéssel, vagy erre épülő kutatásokkal foglalkozni, de nem rendelkeznek megfelelő programozási ismeretekkel az algoritmusok önálló implementációjához. Mindemellett, szoftverfejlesztésben jártas, a témában jobban elmélyedt mérnökök számára is nagy segítséget kínál egy jól felépített grafikus felület. A szoftver használatával

ugyanis a mozgáskövetés és analízis egyszerűen és gyorsan elvégezhető, szemben az olykor körülményes és mindenképp időigényes egyéni videófeldolgozó megoldásokkal.

A grafikus felhasználói felület és egyben az egész applikáció „motorja” a *PyQt5* [25] keretrendszer segítségével készült el. A *PyQt5* tulajdonképpen egy Python-hoz egyszerűen letölthető szoftvercsomag, amely lehetővé teszi az alkalmazások készítéséhez egyik leggyakrabban alkalmazott keretrendszer, a *Qt* programozását Python környezetből. A *Qt* használatának több előnye is van szemben más keretrendszerekkel és közvetlenül Python-hoz készített grafikus megoldásokkal szemben. A legfontosabb, a szoftverfejlesztés szempontjából kiemelt tulajdonsága, hogy a *Qt* egy platformfüggetlen rendszer, ami natívan támogatni tudja napjaink legnépszerűbb operációs rendszereit. Emellett nem elhanyagolható, hogy a *Qt* C++ programozási nyelven készült implementálva, így futási ideje jelentősen kedvezőbb, mint egy szintisztán Python nyelvre épülő alkalmazás esetében.

Mindemellett elengedhetetlen volt a *PyQt* keretrendszerbe beintegrálni az *OpenCV* könyvtárra épülő videólejátszó, valamint a *matplotlib* alapú grafikonrajzoló funkciókat, egy egységes, áttekinthető applikáció érdekében.

3.1.2. Mozgáskövetés és elemzés

Ahogy azt már a 2. fejezetben bemutattam, a mozgáskövetést az *OpenCV* [6] könyvtár segítségével valósítottam meg. Itt fontos feladat volt egy egységes API kialakítása a követőalgoritmusok számára, hogy a felhasználó könnyedén tudjon választani a különböző algoritmusok közül. Szintén fontos fejlesztés, az algoritmus futásának optimalizálása több követendő objektum esetén a minél gyorsabb számítási sebesség elérése érdekében.

A követőalgoritmusok sikeres lefutása utáni eredmény az egyes objektumok nyers pozícióadata. A további elemzés érdekében számos numerikus módszer áll a rendelkezésre a sebesség valamint gyorsulásszintű adatok meghatározására. Mivel a numerikus deriválás kiemeli a zajt, így fontos szerepet kap a különböző szűrők alkalmazása. Mindezek implementálása a *pynumdiff* [7] szoftvercsomag felhasználásával történt. A *pynumdiff* egy olyan Pythonhoz fejlesztett könyvtár, amelyben számos numerikus differenciálalgoritmus elérhető. Külön kiemelendő, hogy a különböző módszerek által használt paraméterek meghatározására elérhető egy intuitív, optimalizációra épülő eljárás is. Ennek segítségével az algoritmusok képesek behangolni a saját paramétereiket, a minél pontosabb eredmények érdekében.

3.1.3. Eredmények megjelenítése, exportálása

A „motion-tracking” és az adatok feldolgozása után elengedhetetlen a kapott eredmények megjelenítése, kimentése a további munka támogatásához. Az adatok illusztrálását egy *matplotlib* alapú grafikonrajzoló részalkalmazás végzi, amellyel lehetőség van különböző követett objektumok mozgásának szemléletesére és összehasonlítására. Emellett az *OpenCV* videólejátszó segítségével, az objektum mozgása könnyedén visszajátszható, a

bejárt trajektória ábrázolható.

A nyers és feldolgozott adatok mellett természetesen mind a visszajátszott videók, mind a készített grafikonok exportálhatók a legnépszerűbb fájlformátumokba, amelyért a korábban bevezetett részegységeken felül egy *pandas* [40] [28] alapú adatkezelő rendszer a felelős.

3.2. Szoftver funkciók

A következő fejezet egy egyszerű mintavideo elemzésével mutatja be a szoftver összes elérhető funkcióját. A könnyebb érthetőség érdekében, a leírás mellett a funkciók bemutatására egy videót is készítettem amely elérhető online¹.

3.2.1. Az applikáció indítása

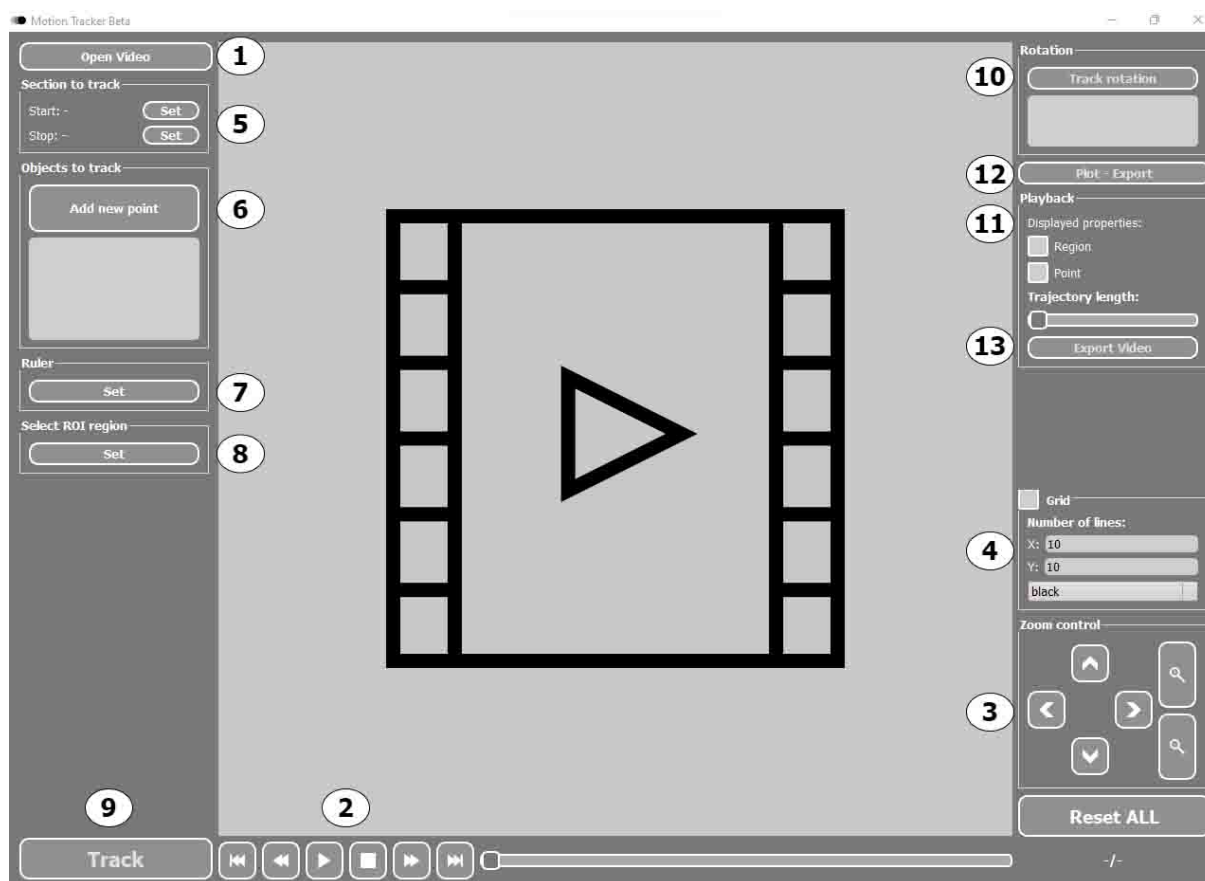
A telepítés módjától függően, eltérő módszerekkel indítható el a program. Amennyiben az egy Python környezetben lett telepítve, egy parancssor segítségével a következő utasítások bármelyikének kiadásával indítható:

```
$ MotionTrackerBeta
$ MotionTracker
$ motiontracker
```

Installer alkalmazás használata esetén pedig még egyszerűbb dolgunk van, ilyenkor elég megkeresni az alkalmazást a telepített programok között.

Sikeres indítást követően az alkalmazás kezdőképernyője fogadja a felhasználót. Ez a grafikus felület a 3.2 ábrán látható, ahol egyben meg megszámoltam a legfontosabb interaktív elemeket. Ezekre a további fejezetekben zárójelben (*szám*) fogok hivatkozni.

¹<https://tinyurl.com/motiontrackerbeta>



3.2. ábra. Az alkalmazás kezdőképernyője.

3.2.2. Videók beolvasása és visszajátszása

Az objektumkövetés első lépése az elemezni kívánt videó betöltése. Az *Open Video* (1) gombra kattintva a szoftver megnyit egy fájlkezelő párbeszédablakot, amely segítségével ki lehet választani a felvételt. Jelenleg a szoftver a következő fájlformátumokat támogatja: MOV, MP4, AVI, MKV.

A megnyitást követően a videó bekerül az applikáció médialejátszó rendszerébe. Itt az irányítógombok (2) segítségével a visszajátszás könnyedén vezérelhető, hasonlóan más lejátszó programokhoz. Hasznos funkció ugyanakkor a beépített nagyítási és mozgatási lehetőség, mely segítségével fókuszálni lehet a felvétel releváns részeire, akár lejátszás közben is. A zoom vezérlésére két mód áll rendelkezésre, Egyrészt lehet használni a számítógéphez csatlakoztatott egér görgőjét, de amennyiben az nem áll rendelkezésre, irányítógombok (3) is elérhetők. Hasonlóan, a nagyított képen történő mozgást is lehet a W, A, S, D billentyűkkel, valamint a grafikus felületen is vezérelni.

Szintén a könnyebb navigálás segítése érdekében a lejátszóba be van építve egy opcionális rácsozási lehetőség, amely állapotát szintén a grafikus felületről lehet szabályozni (4). Annak érdekében, hogy a különböző felbontású és árnyalatú videókhoz is minél jobban alkalmazkodni tudjunk, tetszőleges beosztás készíthető és a rács színe is állítható.

3.2.3. Releváns részlet kijelölése

A mozgáselemzés következő lépése a videó releváns részeinek kijelölése. Mivel még optimalizált objektumkövető algoritmusok alkalmazása esetén is komoly számítási idővel lehet számolni, fontos, hogy a „motion-tracking” csak a videó azon szakaszain fusson, amelyeket analizálni szeretnénk.

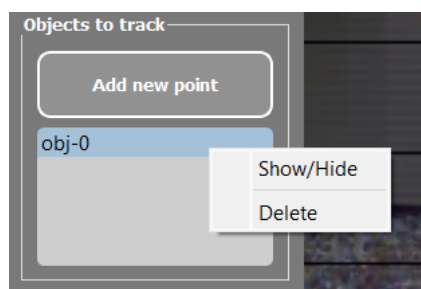
A videórészlet kijelöléséhez a lejátszó vezérlőinek segítségével kell elnavigálni a kívánt időpontba, ahol is a Section to track (5) *Set* gomb megnyomásával lehet a kezdő-, majd utána a záró képkockát rögzíteni.

3.2.4. Követendő objektumok kijelölése

Az elemezni kívánt videórészlet kijelölése után, a követendő objektumok definiálása következik, amelyet szintén a grafikus felhasználoi felület ide kapcsolódó részen (6) lehet megadni. A szoftver alapvetően tetszőleges mennyiségű követendő pontot tud kezelni, de a felhasználónak mindenképpen ajánlott figyelembe venni, hogy a minél több a felvett objektum, annál hosszabban fognak futni a követőalgoritmusok.



(a) Követendő objektum hozzáadása.



(b) Hozzáadott objektumok elrejtése és törlése.

3.3. ábra. Követendő objektumok kezelése.

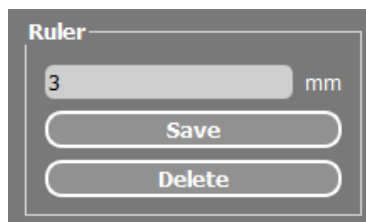
Az *Add new point* gomb megnyomása után a kapcsolódó felület megváltozik, amelyet a 3.3a. ábra mutat. Itt megadható egy egyedi azonosító az létrehozandó objektumnak valamint, láthatóvá válik egy *Point* és egy *Region* gomb is. A *Point* megnyomása után a szoftver kijelölő módba kerül, amit a gomb zöld színűvé válása jelez. Ekkor az egérrel a videóba kattintva megadható a kívánt pont, melyet egy vörös + jelez. A pont megadása mellett azonban a követendő pont körüli régió definiálása is feltétlenül szükséges, ahogy az korábban a 2. fejezetben bemutatásra került. Hasonlóan az korábbiakhoz, a *Region* gombbal lehet a kijelölést elvégezni: a videó aktuális képkockájára, az egeret nyomva tartva, egy téglalapot kell rajzolni, úgy, hogy a korábban már rögzített pont a négyszögön belülre essen. Ezután a létrehozott objektum a *Save* gombbal véglegesíthető.

A korábban definiált pontok egy listában gyűjtve láthatók. Amennyiben a felhasználó

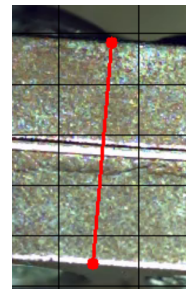
ráklíkkal egy adott objektum nevére, az törölhető, valamint a láthatósága állítható, ahogy azt a 3.3b ábra mutatja.

3.2.5. Mértékegység kalibrálás

Az objektumkövető algoritmusok alapvetően pixelben határozzák meg a kijelölt pontok elmozdulásait, hiszen nincsen információjuk a valós méretekről. Ahhoz, hogy a kapott eredményeket SI mértékegységekben is egyszerűen ki lehessen számolni, a szoftver kalibráló funkcióját lehet használni.



(a) Milliméter megadása.



(b) A „Vonalzó” berajzolása a videóba.

3.4. ábra. A Ruler funkció használata.

A skálázáshoz a szoftver grafikus felületén található Ruler (7) opción belül a *Set* gombot kell megnyomni. Ezt követően a felület átvált és lehetőség van bevinni a kívánt hossz értéket milliméterben, ahogy azt a 3.4 ábra mutatja. Ezután az egeret lenyomva tartva a videó megjelenítő aktuális képkockájára egy vonalat lehet rajzolni. A program ennek a vonalnak a hosszát fogja társítani a korábban bevitt értékhez a belső számítások során.

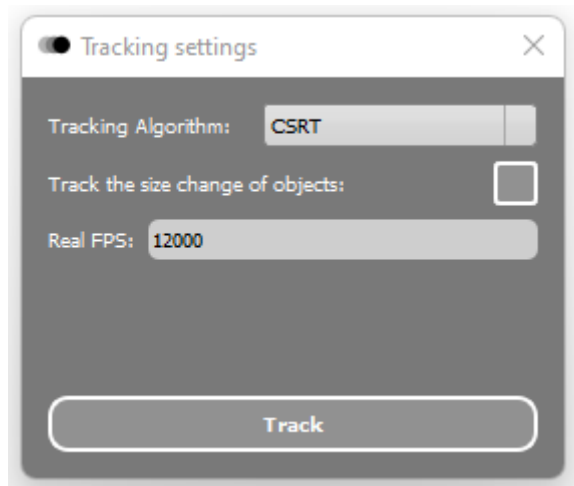
Fontos megjegyezni, hogy ennek a funkciónak a használata nem kötelező, pusztán egy lehetőség.

3.2.6. Fókusz beállítása

Bár az *OpenCV* objektumkövető algoritmusai igen hatékonyak, a futási idő nagy felbontású felvételek esetén jelentősen nőhet. Ez a folyamat gyorsítható opcionálisan, a megfelelő fókusz (Range Of Interest = ROI) beállításával. A beállításhoz a Select ROI region (8) *Set* gombját lehet használni, amelyet követően a már jól ismert egérrel rajzolós módszerrel végezhető el a kijelölés. Fontos, hogy beállított fókusz esetén, az objektumkövető algoritmusok csak a képkockák kijelölt részét töltik be, így amennyiben az objektum elhagyja azt a követés sikertelen.

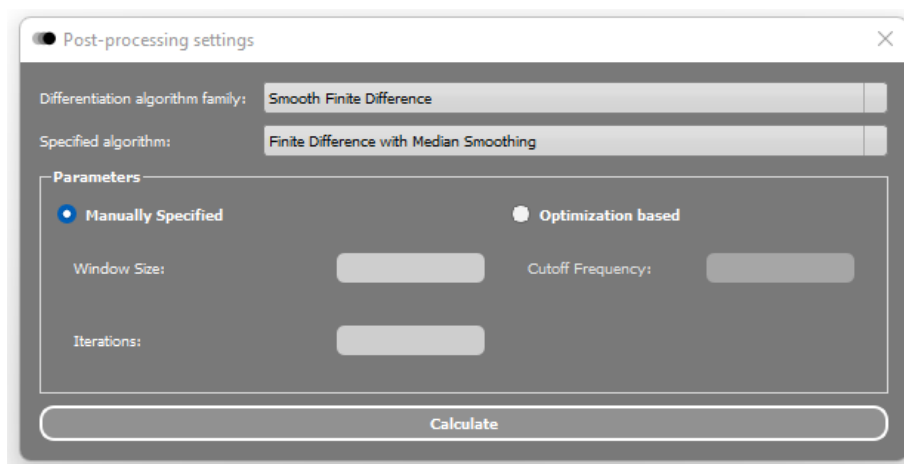
3.2.7. Mozgáskövetés és adatfeldolgozás

Amennyiben a releváns szakasz eleje és vége kijelölésre került és legalább egy pont meg lett adva, az objektumkövető algoritmusok futtathatók. Ezt a *Track* (9) gomb aktiválásával jelzi a program.



3.5. ábra. A „motion-tracking” algoritmusok beállításai.

A *Track* gomb megnyomása után a szoftver egy új párbeszédablakot nyit, melyben a követő-algoritmusok közvetlen beállításai módosíthatók. Ez az ablak a 3.5. ábrán látható. Itt a kívánt követőalgoritmus kiválasztása mellett, lehetőség van a videó FPS átdefinálására, ugyanis ez az érték gyorskamerás felvételek esetén általában eltér a videófájlban rögzítettől. Ezen kívül a program képes a objektumok méret változásait is vizsgálni a megfelelő jelölőnégyzet használata esetén. A szükséges paraméterek megadása után a „motion-tracking” a *Track* gommbal indítható. Ezt követően elindul az objektumkövető algoritmus és egy újabb párbeszédablakban jelzi vissza a szoftver a folyamat előrehaladását.



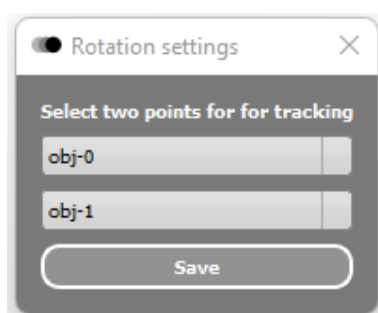
3.6. ábra. A pozícióadatok feldolgozását végző numerikus differenciáló algoritmusok paramétereinek beállítására szolgáló párbeszédablak.

A követés végeztével egy újabb párbeszédablak nyílik meg, amit a 3.6 ábra szemléltet. Ez a felület a rögzített pozícióadatok feldolgozásához szükséges numerikus differenciáló algoritmusok, valamint azok paramétereinek beállítására szolgál. Ahogy azt már korábban a 2. fejezetben bemutattam, rendkívül sok különböző algoritmus közül lehet választani, az adott igényeknek megfelelően.

Amennyiben nem vagyunk elégedettek az adott numerikus módszer eredményével, érdemes másikat választani, vagy a paramétereket hangolni. Fontos azonban, hogy ilyen esetben nem szükséges újra lefuttatni a „motion-tracking” algoritmusokat. A *Track* gomb felett megjelenő *Reprocess*-re segítségével a program képes a korábban rögzített adatokat használni, így jelentős idő takarítható meg.

3.2.8. Szögelfordulás mérése

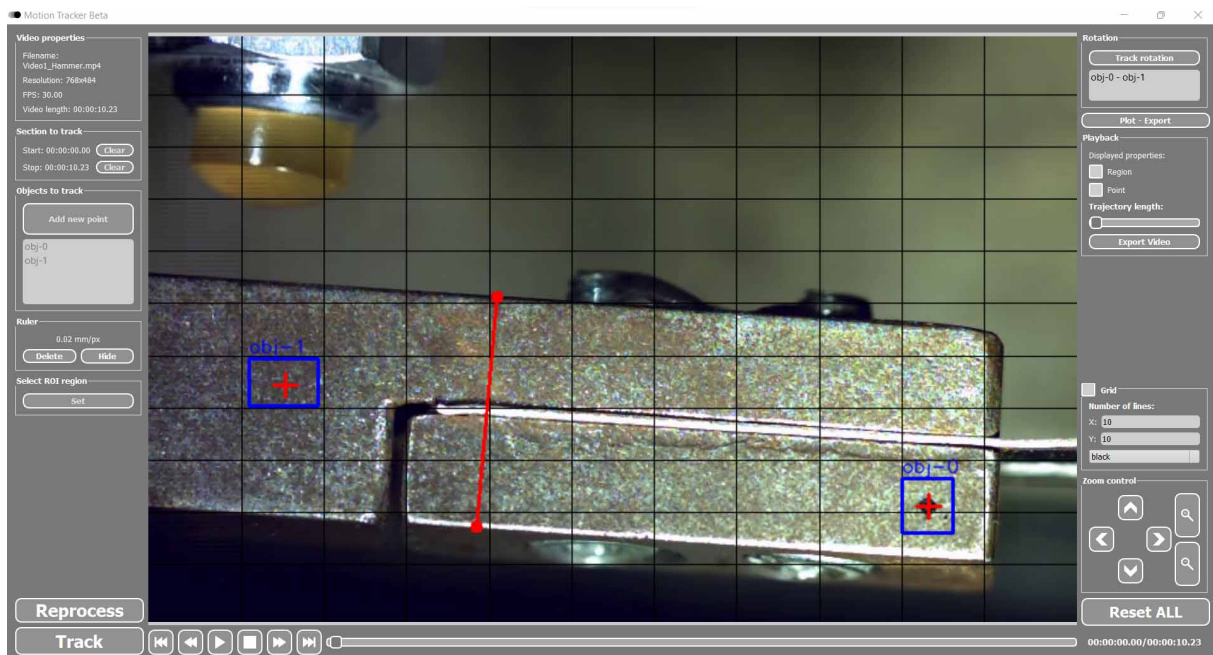
Amennyiben legalább két kijelölt követése sikeresen lezajlott, lehetőség van a két pontot összekötő vonal szögelfordulásának számítására. Ehhez a Rotation (10) modul *Track rotation* gombját megnyomva a két kiválasztott pont megadása szükséges, ahogy azt a 3.7 ábra is mutatja.



3.7. ábra. Referenciapontok megadása a szögelfordulás számításához.

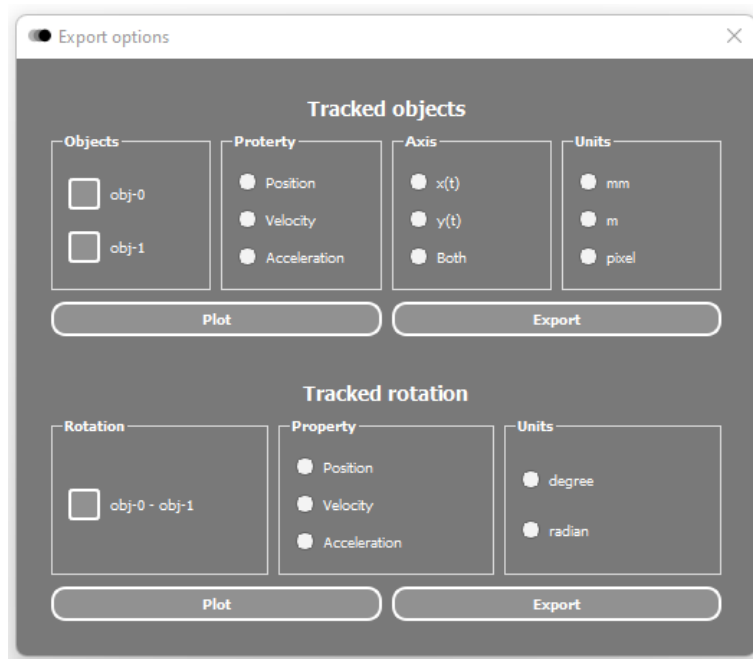
3.2.9. Eredmények megjelenítése és mentése

A „motion-tracking” és az adatfeldolgozás végeztével a szoftver grafikus felülete kisebb mértékben átalakul, ahogy azt a 3.8. ábra mutatja. Több korábbi funkció deaktiválódik, melyeknek pusztán az objektumkövető algoritmusok futtatása előtt van jelentősége, ugyanakkor új lehetőségek válnak elérhetővé.



3.8. ábra. A szoftver módosult kezelőfelülete, az objektumkövető algoritmusok sikeres lefutása után.

Ez egyik ilyen új funkció a videólejátszó modul kiegészülése a visszajátszási opciókkal (11). Itt ki lehet választani, hogy a követett objektumok mely tulajdonságai jelenjenek meg a visszajátszott videón, valamint lehetőség van a befutott trajektória kirajzolására is. Ennek hossza szintén állítható.



3.9. ábra. A szoftver módosult kezelőfelülete, az objektumkövető algoritmusok sikeres lefutása után.

Mindezek mellett a mért, illetve számított adatokat ábrákon is meg lehet jelentetni a beépített grafikonkészítő és exportáló modul segítségével. A *Plot/Export* (12) gombra kattintás után megnyíló párbeszédablakban (3.9. ábra) könnyedén kiválasztható a vizsgált objektum és annak kívánt paramétere. Ezt követően pedig a *Plot* illetve *Export* gombok segítségével lehetőség van a grafikon kirajzolására valamint az exportálásra. Utóbbi opció esetén a létrehozandó fájl elérési útjaa, valamint formátuma is definiálható. Jelenleg a txt, a csv és az xlsx forátum támogatott.

Szintén van lehetőség a program által készített grafikonok, valamint a visszajátszott követett objektumokat tartalmazó videó exportálására is. Előbbit a kirajzolt grafikon mentés gombjával utóbbit a fő ablak grafikus felületén található *Export Video* (13) gombbal lehet véghez vinni.

4. Alkalmazási példák

A következő fejezetben különböző mintapéldákon keresztül szemléltetem a szoftver alkalmazási lehetőségeit. Először egy mesterségesen generált videó elemzésével validálom a szoftver pontosságát. Ezt követően egy húzókísérleten keresztül végzek mozgásvizsgálatot. Ezután egy airsoft golyó becsapódásáról készült nagysebességű felvétel analízisén keresztül mutatom be a numerikus deriváló algoritmusok használatát. Végül, hogy a szoftver sokoldalúságát illusztráljam, felsőnézetes drónfelvételek feldolgozását mutatom be, ami kiválóan prezentálja a követőalgoritmusok robusztusságát összetett, gyorsan változó környezetben.

4.1. Mesterséges videók elemzése validálási célokra

Ahhoz, hogy a fejlesztett szoftver és az alkalmazott követőalgoritmusok pontosságáról meggyőződhessünk mesterséges referenciavideókat alkalmaztam.

Ezeket a videókat az *OpenCV* könyvtár segítségével készítettem. A fehér háttéren egy fekete pont, egy paraméterezett pályát fut be. Az egyes koordináták időfüggvényei a következők:

$$x(t) = 1920 + 1500 \sin\left(\frac{2\pi t}{10}\right), \quad (4.1a)$$

$$y(t) = 1080 + 800 \sin\left(\frac{2\pi t}{10}\right). \quad (4.1b)$$

Ennek ismeretében a deriváltak is egyszerűen meghatározhatók, melyek a sebesség, valamint a gyorsulás értékeknek felelnek meg:

$$\dot{x}(t) = 300\pi \cos\left(\frac{\pi t}{5}\right), \quad (4.2a)$$

$$\dot{y}(t) = 160\pi \cos\left(\frac{\pi t}{5}\right), \quad (4.2b)$$

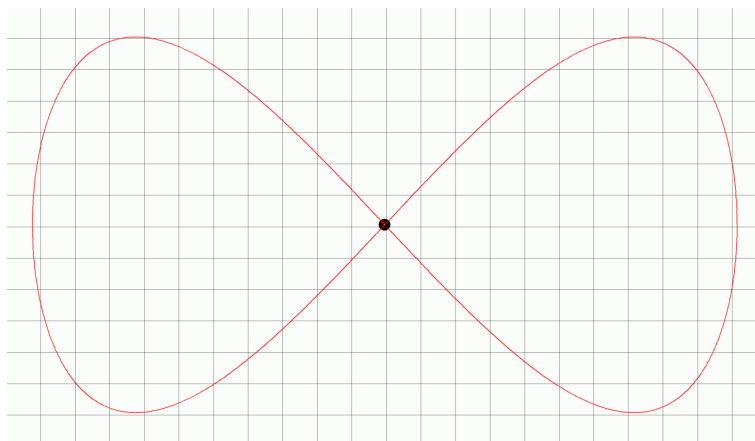
$$\ddot{x}(t) = -60\pi^2 \sin\left(\frac{\pi t}{5}\right), \quad (4.2c)$$

$$\ddot{y}(t) = -32\pi^2 \sin\left(\frac{\pi t}{5}\right). \quad (4.2d)$$

A videó 10 másodperces, vagyis $t = 10$ s, felbontása 4K (3840x2160 pixel) és 120 FPS. A létrehozott fájl elérhető online¹.

A pontos validáció érdekében az összes követőalgoritmust lefuttattam, azonos beállítások mellett, valamint ugyanazt az iterált véges differencia módszert használtam a sebességek és gyorsulások kiszámítására.

¹<https://tinyurl.com/tracker-validation-sample>



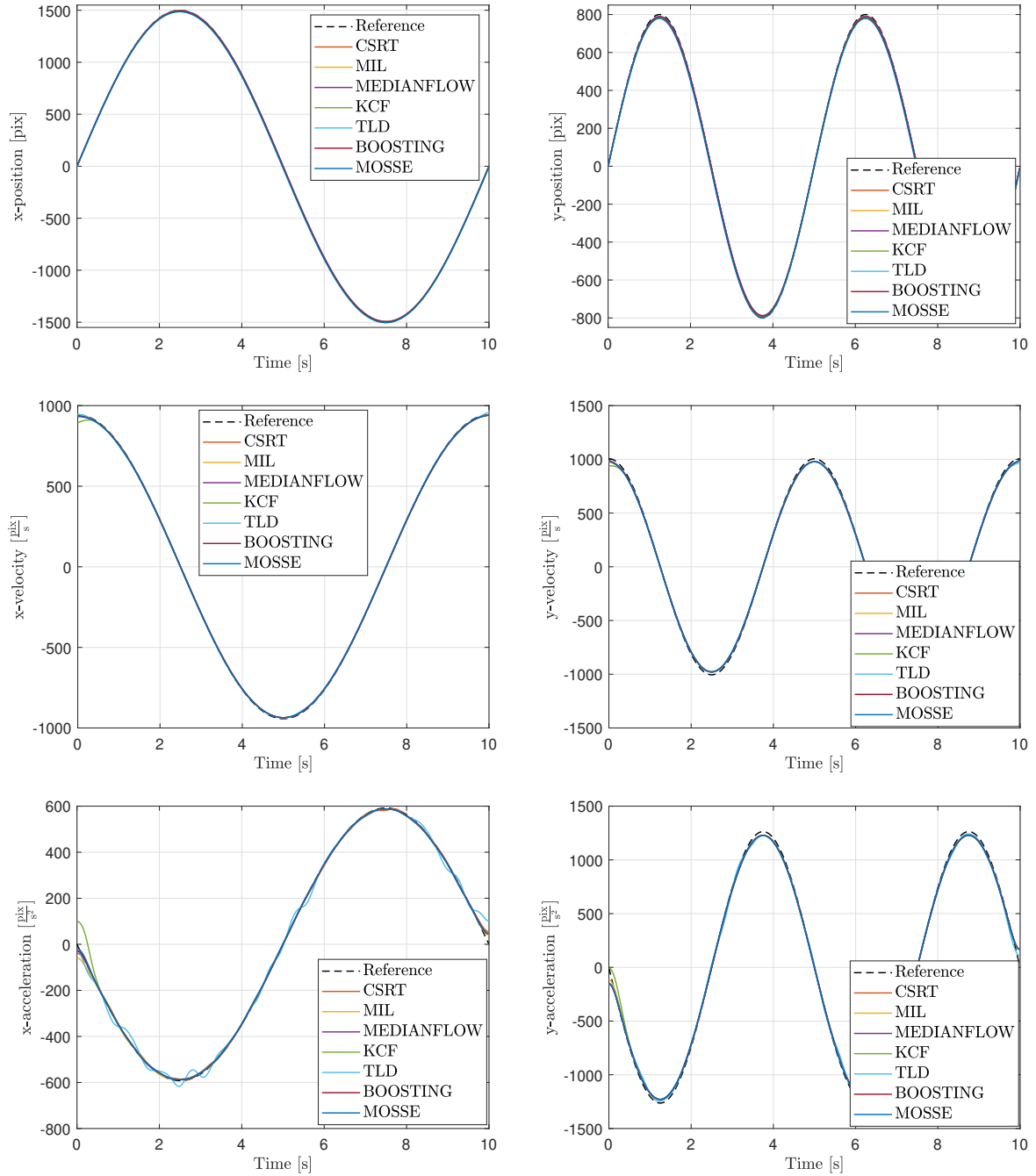
4.1. ábra. Objektumkövetés és a kirajzolt pálya a validációs videón. Az alkalmazott rácsozás lépésköze 175 pixel.

A követő algoritmusok és az adatfeldolgozás után az összes adatot exportáltam, hogy MATLAB segítségével egyszerűen tudjam összehasonlítani a kapott eredményeket a referenciával. Azt, hogy a gyűjtött adatok mennyire illeszkednek a valós értékekre a korrelációs együttható kiszámításával lehet meghatározni. Az eredményeket az 4.1 táblázat tartalmazza.

Algoritmus	CSRT		MIL		MEADIANFLOW		KCF		TLD		BOOSTING		MOSSE	
	x	y	x	y	x	y	x	y	x	y	x	y	x	y
Pozíció	1	0.9998	1	0.9998	1	0.9998	0.9999	0.9996	1	0.9998	1	0.9998	0.9999	0.9994
Sebesség	1	0.9993	1	0.9992	1	0.9993	0.9999	0.9991	1	0.9992	1	0.9993	0.9999	0.9992
Gyorsulás	0.9997	0.9990	0.9997	0.9990	0.9999	0.9990	0.9975	0.9986	0.9980	0.9988	0.9999	0.9990	0.9999	0.9989

4.1. táblázat. Kiszámított korrelációs együtthatók a követőszoftver és a referencia összehasonlítására.

Jól látható, hogy minden követőalgoritmus esetén pontos eredményeket kapunk, és megfelelő beállítások mellett a numerikus differenciálás sem rontja a pontosságot számottevően. A következőkben a gyűjtött adatokat grafikonon kirajzolva hasonlítom össze a referenciával, melyet a 4.2 ábra mutat. A szemléletesség érdekében nem a teljes időtartam került kirajzolásra, hanem csak kis részletek, az eltérések könnyebb megfigyelésére. A pozíció szintű adatokban a „motion-tracking” algoritmusok eltérése a referenciától és egymástól gyakorlatilag minimális. A sebesség szinten már kicsivel nagyobb eltérések láthatók, főleg a kezdeti és végpontokban. A gyorsulások elemzésénél pedig már kiválóan látszik az, hogy melyik algoritmus mekkora zajt vitt bele a követésbe, hiszen a differenciálás ezt kiemeli. Ugyanakkor fontos megjegyezni, hogy megfelelő feldolgozási paraméterek választásával a gyorsulásértékek pontossága is javítható, ebben a fejezetben pusztán az összehasonlíthatóság okán voltak ugyanazok az értékek válaszva.

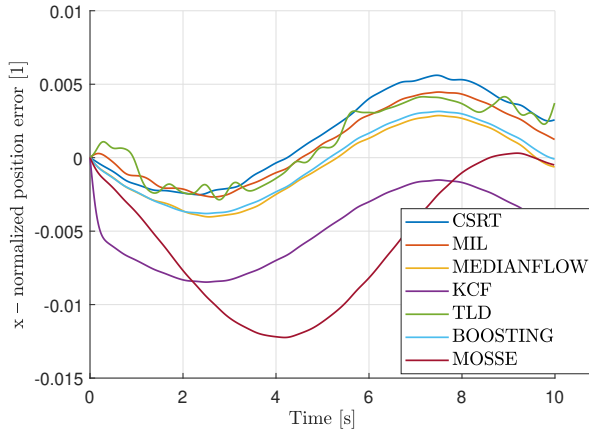


4.2. ábra. A validációs mérések eredményei összehasonlítva a referenciával.

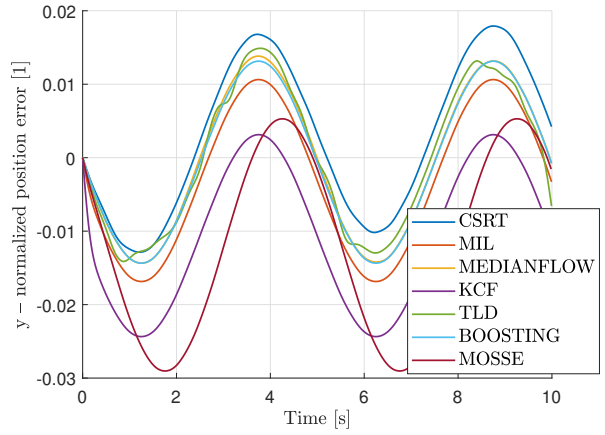
Mivel a kapott ábrákon az eltérés minimális, a könnyebb összehasonlíthatóság érdekében a követés normalizált hibáját is meghatároztam, amelyet a következő képlet definiál:

$$e_{\hat{x}} = \frac{\hat{x} - x}{\max\{x\}}, \quad (4.3)$$

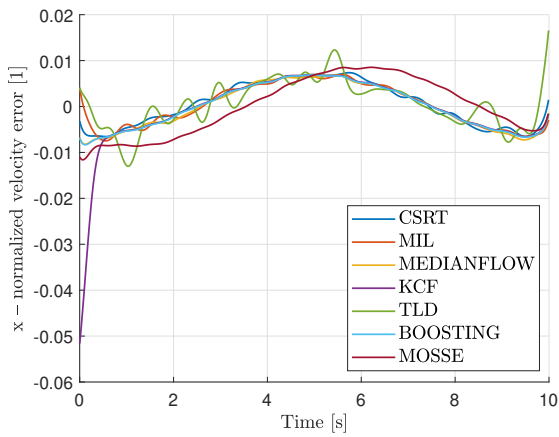
ahol $\hat{x}$ a szoftver által mért, illetve számított eredmény, x pedig a valós érték. A kapott eredmények a 4.3 ábrán láthatók.



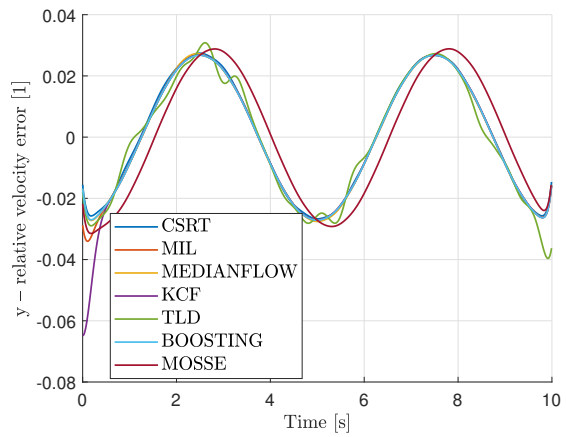
(a) x pozíció normalizált hibája



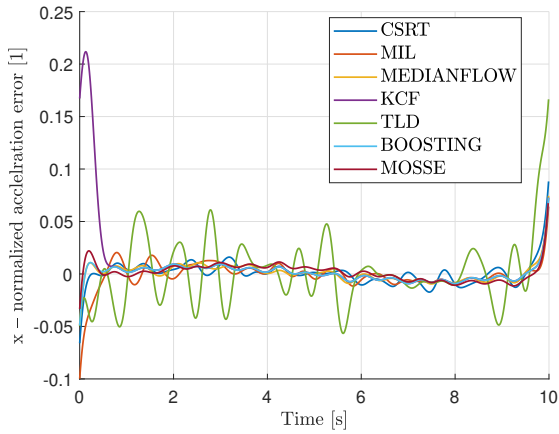
(b) y pozíció normalizált hibája



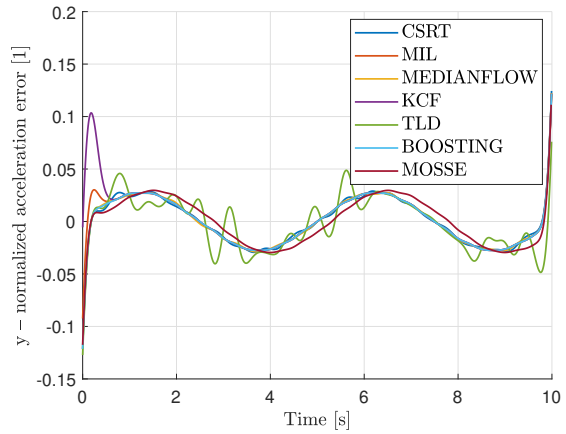
(c) x sebesség normalizált hibája



(d) y sebesség normalizált hibája



(e) x gyorsulás normalizált hibája

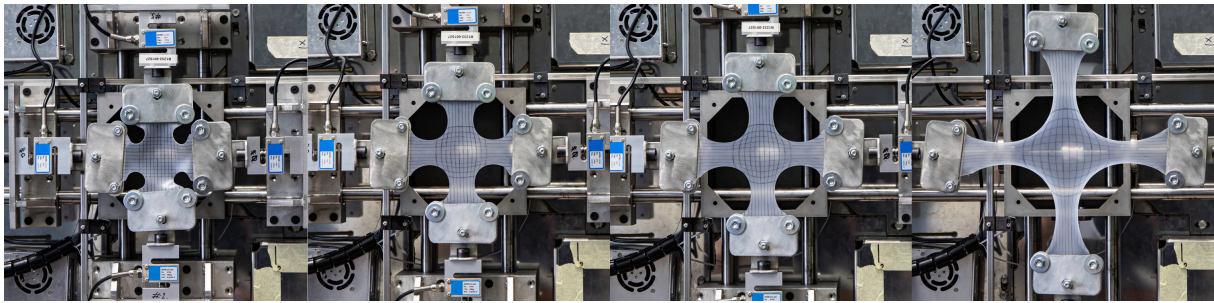


(f) y gyorsulás normalizált hibája

4.3. ábra. A validációs mérések normalizált

4.2. Biaxiális húzókísérlet elemzése

Húzókísérletek fontos eszközei a mérnököknek, különböző anyagok vizsgálatához. Segítségükkel a különböző anyagparaméterek mellett anyagmodellek felállítása is lehetséges.

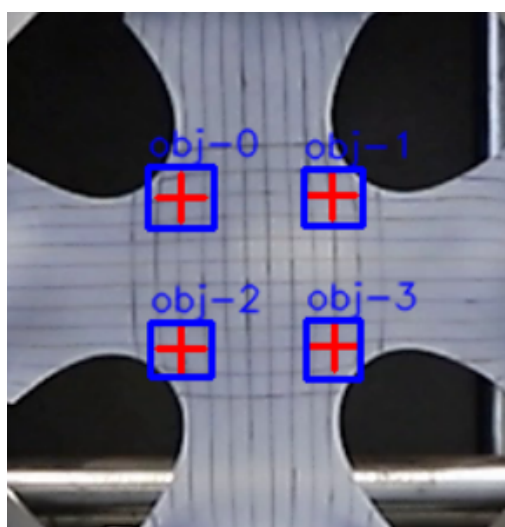


4.4. ábra. A biaxiális húzókísérlet folyamatának szemléltetése 4 állapothoz tartozó fénykép segítségével.

Ebben a fejezetben egy szilikon lemez biaxiális húzókísérletét elemzem ki az objektumkövető szoftver segítségével. Az elemzett videó elérhető online¹, de a jobb szemléltetés érdekében a húzókísérlet folyamata a 4.4 ábrán látható.

Képfeldolgozási eljárások alkalmazása a húzókísérletek elemzésére igen elterjedt [33]. Ennek oka, hogy bár a legtöbb kísérletet végrehajtó gép rendelkezik megfelelő szenzorokkal az elmozdulás mérésére, ezek csak a befogás pontjáról tudnak pontos információkat szolgáltatni. Abban az esetben, ha egy erősen nemlineáris anyagmodell felállítása a cél szükség lehet belső pontok viselkedésének ismeretére is, amely a megfelelő képfeldolgozási eljárásokkal, például *Digital Image Correlation* (DIC) [20] technikával valósítható meg.

Jelen bemutatóban a saját fejlesztésű objektumkövető szoftver felhasználásával igyekeztem kinyerni a szükséges elmozdulás adatokat. A program erejének és sokoldalúságának prezentálása érdekében sok követendő pontot jelöltem ki. Fontos jellemző, hogy ezeket a program egy iteráció alatt gyorsan képes lekövetni, így jelentős idő takarítható meg a DIC technikákkal szemben, amelyek általában jelentős hangolást igényelnek a különböző alkalmazási területen, mielőtt pontos eredményeket tudnának szolgáltatni.

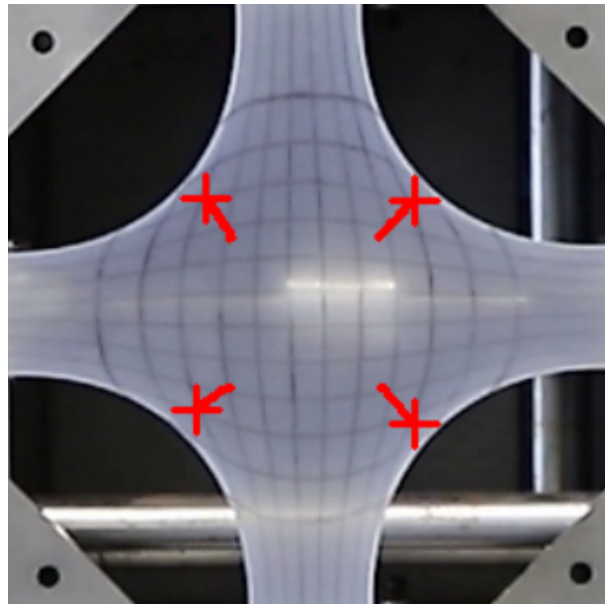


4.5. ábra. Kijelölt objektumok a biaxiális húzókísérletben.

¹<https://tinyurl.com/biaxial-sample>

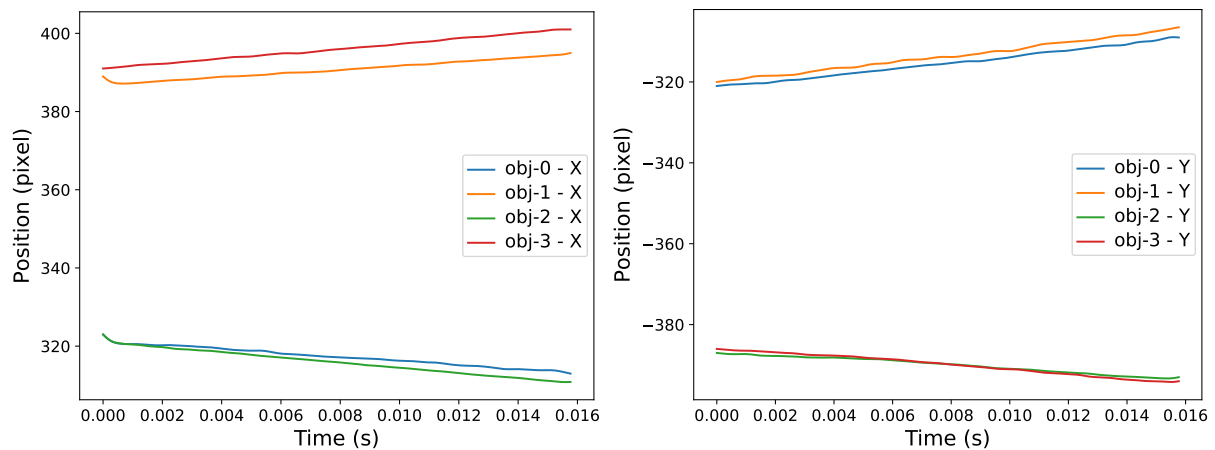
A követőalgoritmusok futtatásához a megfelelő videórész kijelölése után meg kell adni a kívánt objektumokat. jelen példában összesen 4 objektum kerül kijelölésre, melyek a 4.5. ábrán láthatóak. A „motion-tracking” elvégzéséhez az FPS-számot nem változtattam, követőalgoritmusnak pedig a CSRT módszert választottam, ugyanis több próbálkozás után ez bizonyult a legpontosabbnak. Jelen helyzetünkben a sebesség, illetve gyorsulás szintű információ nem annyira lényeges, így az adatfeldolgozási beállítások nem relevánsak.

Sikeres futtatás után első lépésként a videót visszajátszva, a befutott trajektória kirajzolásával a legegyszerűbb ellenőrizni az eredményeket. Egy képernyőkép a visszajátszásról a 4.6 ábrán látható.



4.6. ábra. Biaxiális húzókísérlet eredménye, jelölve a kijelölt pontok elmozdulásával.

Az eredmények kiértékelése érdekében az elmozdulások grafikonon is ábrázolhatók. Ezt a szoftver beépített grafikonkészítő moduljával kiviteleztem. Az eredményeket külön x és y irányú elmozdulásokra lebontva a 4.7 ábra mutatja.

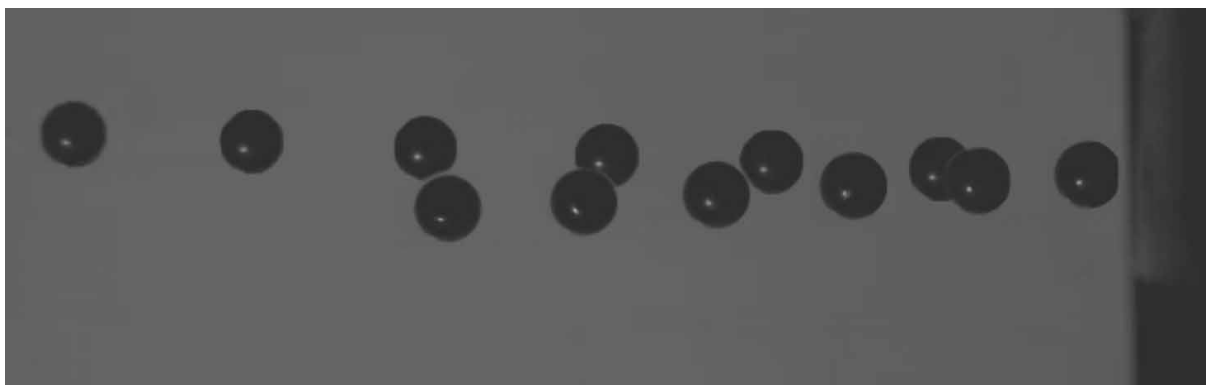


4.7. ábra. Biaxiális húzókísérlet eredményei grafikonon ábrázolva.

4.3. Airsoft golyó becsapódásának vizsgálata

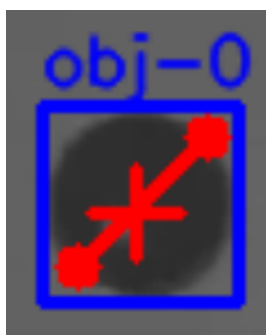
A következő mintapéldában egy airsoft golyó becsapódásáról készült videó elemzését mutatom be. Ennek a példának a célja, hogy demonstrálja a numerikus differenciáló algoritmusok adta lehetőségeket, melyek segítségével sebesség, illetve gyorsulás szintű adatok számíthatók. Ezek az információk kiemelten hasznosak becsapódások vizsgálatakor. A sebességeket vizsgálva információt kaphatunk az ütközés során disszipálódó energiáról.

Az elemzett videó elérhető online¹, de a szemléletesebb bemutatás érdekében egy képsorozat is szemlélteti a folyamatot a 4.8 ábrán.



4.8. ábra. Az airsoft golyó becsapódásának folyamata.

A vizsgálat megkezdéséhez a videó betöltése utáni első lépés itt is annak a videórészletnek a kijelölése, amelyet elemezni szeretnénk. Ezt követi az objektum, jelen esetben a golyó kijelölése. Ezután jelen mérés esetén fontos lépés a kalibrálás is, mely segítségével SI mértékegységekben kaphatjuk meg az eredményeket. Mivel a képen a golyó átmérője ismert, 6 mm, így ezt kell felhasználni a kalibráláshoz. A kijelölt objektumot és a kalibrációs egyenest a 4.9 ábra szemlélteti..



4.9. ábra. A kijelölt objektum és a „Vonalzó” eszköz segítségével berajzolt szakasz a kalibrációhoz.

A kalibrálást az objektumkövető algoritmus futtatása követi, melyhez a következő beállítások fontosak: az FPS értékét 15000-re módosítottam, mivel eredetileg ilyen képkockase-

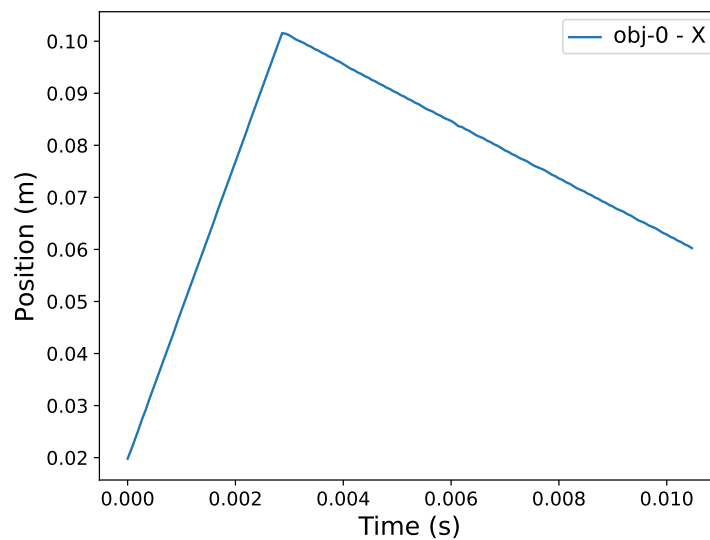
¹<https://tinyurl.com/airsoft-sample>

bességgel készült a videó. Emellett az elérhető algoritmusok közül ebben a mintapéldában a BOOSTING algoritmust választottam. A feldolgozáshoz ezúttal a numerikus differenciáló algoritmusok is fontosak. A megfelelő zajszűrés érdekében a Savitzky-Golay filterre épülő differenciálást választottam, melynek paramétereit optimalizáció útján a program automatikusan határozza meg.



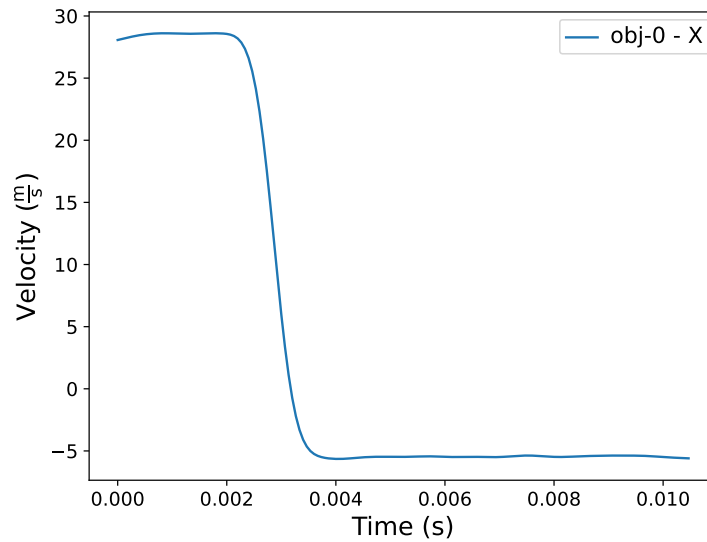
4.10. ábra. Az objektumkövető algoritmus futtatásának eredménye.

Az eredmények kiértékelésekor jelenleg csak az x irányú adatokkal foglalkozunk, de a vizsgálat könnyen kiterjeszthető az y tengelyre is. A követőalgoritmusok által számított pozíció a 4.11 ábrán látható.



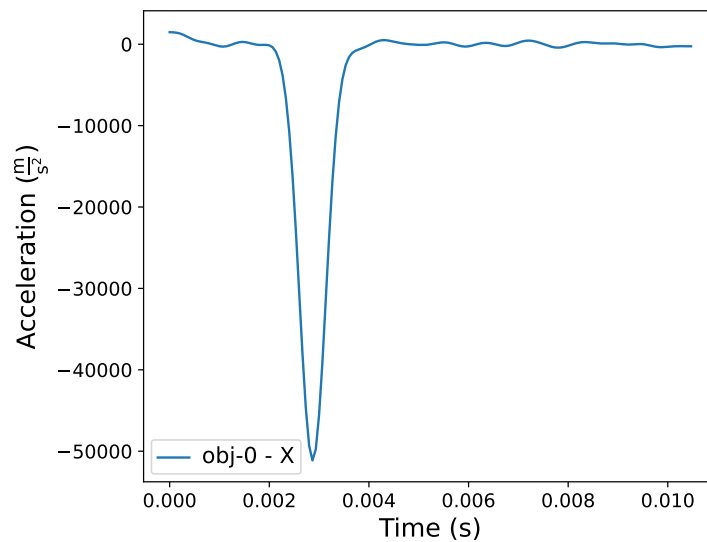
4.11. ábra. Airsoft golyó pozíciója az x tengelyen.

A pozíciót felhasználva numerikus deriválással a sebesség is meghatározható, ezt mutatja a 4.12 ábra. Jól látható, hogy az ütközéssel jelentős energia disszipálódik és a ~ 28 m/s sebességgel érkező golyó a visszapattanás után már csak ~ 5 m/s-mal halad.



4.12. ábra. Az airsoft golyó számított sebessége.

További differenciálással a gyorsulás is számítható. Fontos megjegyezni azonban, hogy erre az értékre jelentős hatással vannak az alkalmazott szűrő-algoritmusok. Amennyiben szűrés nélkül, vagy túl gyenge szűréssel dolgozunk, a zaj elnyomás nem lesz kellően hatékony és a kapott eredmény használhatatlan lesz. Abban az esetben viszont, ha túlságosan erősen simítjuk a sebesség jelet, a számított gyorsulás a valós érték alatt marad, vagyis pontatlan eredményt kapunk.

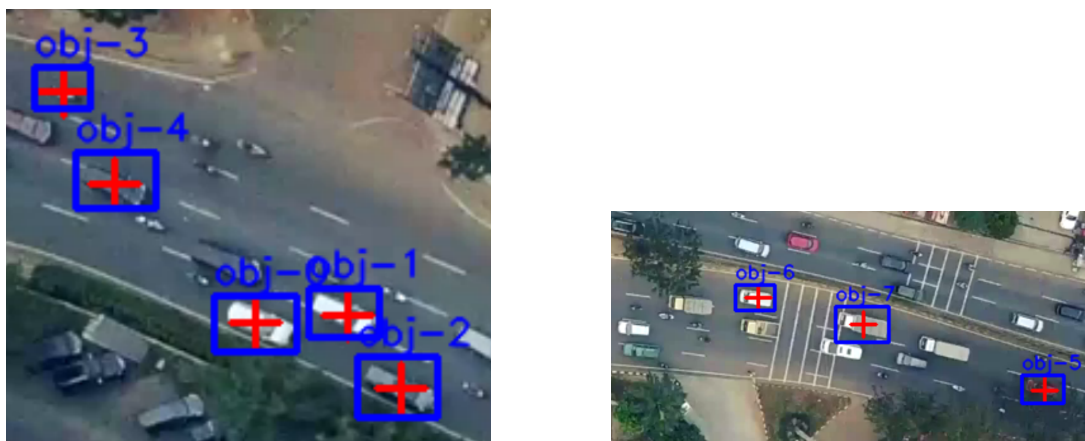


4.13. ábra. Az airsoft golyó számított gyorsulása.

4.4. Gépjárműforgalom vizsgálata drónfelvételek elemzésével

Ebben a fejezetben a mechanikai vizsgálatoktól kicsit elrugaszkodva, egy közlekedési csomópont vizsgálatának bemutatásával igyekszem prezentálni a fejlesztett szoftver sokoldalúságát és lehetőségeit. Jelenleg, többek között a BME Közlekedéstechnológiai és Közlekedésgazdasági Tanszékén is fut olyan kutatás, mely során kereszteződésekről készült videófelvételek elemzésével nyertek ki forgalomspecifikus információkat.

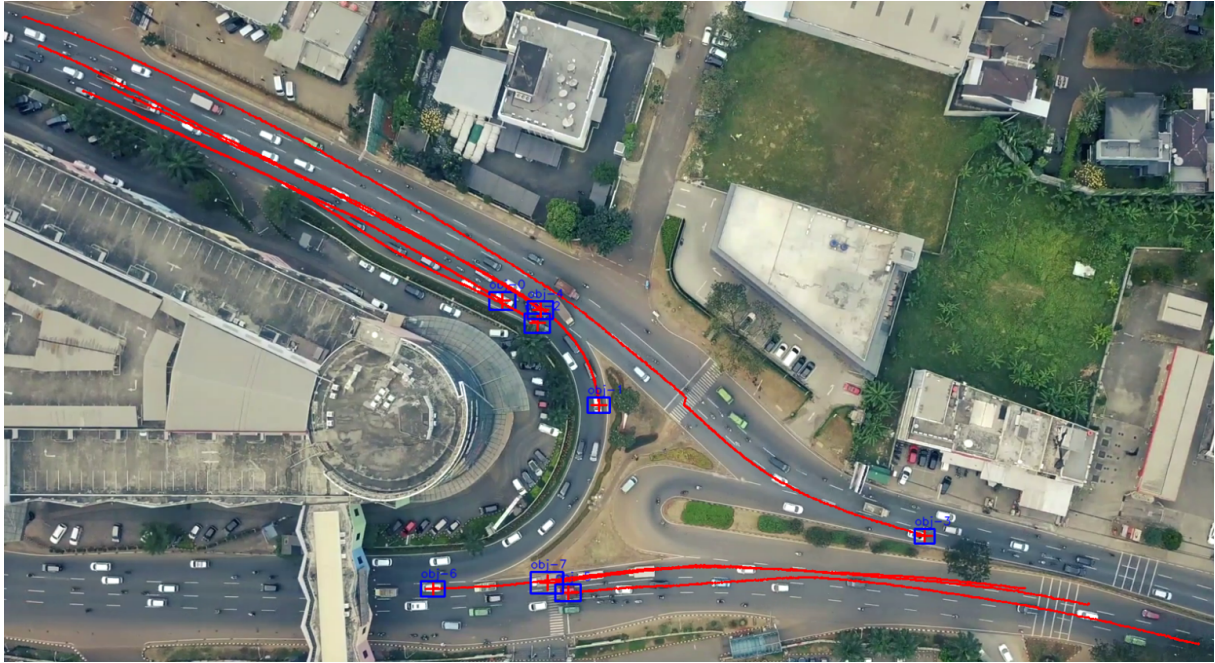
A kiválasztott videó egy nagy közlekedési csomópontot mutat be, melyben több jármű szimultán követése a cél. A kiválasztott videó a korábbiakhoz hasonlóan elérhető online¹. A kezdeti kijelölt járműveket a 4.14 ábra mutatja. A kiválasztásnál fontos szempontnak tartottam, hogy a nagy jól látható autók mellett, kicsi, pár pixelből álló motorosokkal is prezentálni tudjam a szoftver pontosságát és lehetőségeit. Továbbá figyeltem arra, hogy egyes járművek fák, illetve egyéb zavaró tényezők alatt is áthaladjanak, tesztelve ezzel, hogy a program képes-e így hatékony követésre.



4.14. ábra. A kijelölt járművek.

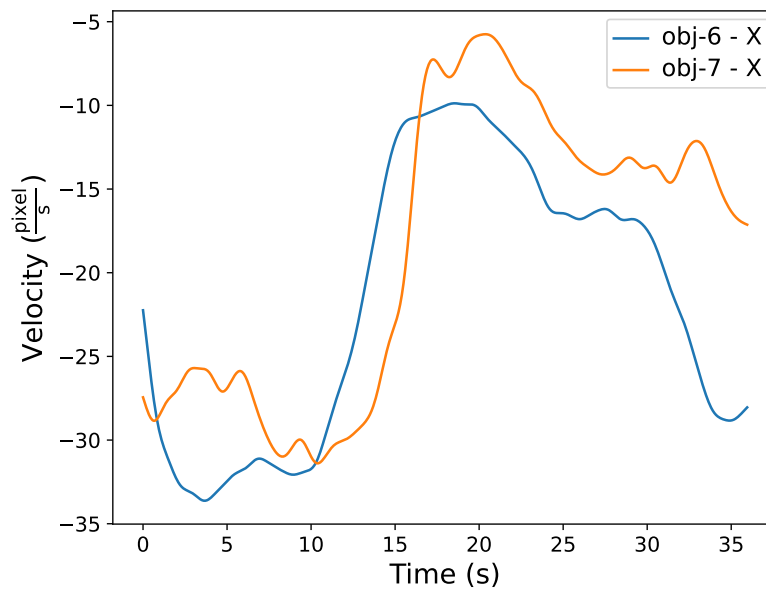
Mivel pontos információval nem rendelkezünk a valós méretekről, így a kalibráció jelen példában kimarad. A „motion-tracking” beállításainál az FPS megfelel a videófájlban tárolt értéknek, a követéshez pedig ezúttal a CSRT algoritmust választottam. Az adatok feldolgozásánál ezúttal egy medián-szűrővel simított véges differencia módszert használtam 31-es ablakmérettel és 22 iterációval. Ahogy az a 4.15 ábrán látható a szoftver ugyanolyan hatékonysággal képes követni a kis objektumokat, mint a nagyobb autókat. Továbbá nem zavarja össze az algoritmust, ha több ugyanolyan színű autó halad egymás mellett, illetve a fák és az útburkolati jelek okozta zavarásnak sincs hatása a performanciára.

¹<https://tinyurl.com/traffic-sample>



4.15. ábra. A járművek kirajzolt pályája a „motion-tracking” eredményeként.

A feldolgozást követően az adatok exportálásával további elemzési lehetőségek adóttak. Megfigyelhető az egyes járművek átlagsebessége, sávokra lebontva, de vizsgálható egymás mögött haladó járművek követési távolsága is, melyekből a közlekedésmérnökök hasznos információkhoz juthatnak. A demonstráció céljából a 4.16 ábrán az 6-os és a 7-es jármű x irányú sebességprofilja van összehasonlítva egymással.



4.16. ábra. Két egymást követő jármű sebességprofilja.

5. Konklúzió

Jelen TDK dolgozat egy olyan objektumkövető szoftvert mutatott be, amely több szempontból is hiánypótlónak mondható. Egy kiterjedt irodalomkutatás után ugyanis kijelenthető, hogy nincsen olyan nyílt forráskódú, ingyenesen elérhető és grafikus felülettel rendelkező mozgáskövető applikáció, amely mérnökök által végzett mérések egyszerű kiértékelésére specializál.

Dolgozatomban a bevezetés után az objektumkövetés elméleti hátterét és a szoftver elkészítéséhez felhasznált mozgáskövető algoritmusokat mutattam be, az *OpenCV* könyvből. Ezt követte a numerikus differenciálás módszereinek és nehézségeinek bemutatása, valamint különböző algoritmusok ismertetése. Ezen algoritmusok rendkívül fontosak, hiszen a szoftver egyik legnagyobb fegyverténye, hogy kiegészítő eszközök használata nélkül, az applikáción belül megvalósítható a sebesség, illetve gyorsulásértékek számítása. A szükséges elméleti háttér ismeretében, a következő fejezetben az implementált szoftver felépítését mutattam be, kiemelve a programozás során felhasznált könyvárakat, keretrendszereket. Ezután egy egyszerű mintapélán keresztül prezentáltam a szoftver alkalmazásának lépéseit, melyről egy videó¹ is elérhető. Annak érdekében, hogy hatékonyan illusztrálni tudjam a szoftver képességeit és lehetséges alkalmazási területeit, mérések és kísérletek kiértékelését mutattam be. Első lépésben egy mesterségesen előállított videó segítségével a szoftver pontosságát validáltam, majd ezt követték az alkalmazási példák a mechanika különböző területei mellett egyéb lehetséges témákban.

A kapott eredmények azt igazolták, hogy a szoftver nagy pontossággal és megfelelő sebességgel képes kinyerni a szükséges kinematikai mennyiségeket a videóképekből. Azon túl, hogy az elkészült applikációt a Műszaki Mechanikai Tanszéken folyó tudományos kutatásokban már most felhasználják, a szoftvert ingyenesen elérhetővé tettem a kutatók számára: a kapcsolódó GitHub linken² a programon felül részletes dokumentáció és telepítési útmutató is elérhető. Emellett a szoftvert tudományos publikációban is ismertetni tervezem.

Természetesen rengeteg fejlesztési irány merülhet fel, a szoftver további funkciókkal történő bővítése tervben van. Viszont ehhez jelenleg szükséges a felhasználói igények és visszajelzések megismerése, amelyek alapján szükséges és hasznos opciókkal tud bővíteni a *Motion Tracker Beta* névre keresztelt objektumkövető alkalmazás.

¹<https://tinyurl.com/motiontrackerbeta>

²<https://github.com/flochkristof/motiontracker>

Hivatkozások

- [1] S. Avidan. „Ensemble tracking”. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*. 2. köt. 2005, 494–501 vol. 2.
- [2] Boris Babenko, Ming-Hsuan Yang és Serge Belongie. „Visual tracking with online Multiple Instance Learning”. *Proceedings of 2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, 983–990. old.
- [3] Arundhati Bagchi Misra. „Nonlocal Total Variation Based Speckle Denoising Model”. 221. köt. 2013. jan., 517–527. old.
- [4] Justas Birgiolas, Christopher Jernigan, Richard Gerkin, Brian Smith és Sharon Crook. „SwarmSight: Real-time Tracking of Insect Antenna Movements and Proboscis Extension Reflex Using a Common Preparation and Conventional Hardware”. *Journal of Visualized Experiments* 2017 (2017. dec.).
- [5] David S. Bolme, J. Ross Beveridge, Bruce A. Draper és Yui Man Lui. „Visual object tracking using adaptive correlation filters”. *Proceedings of 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2010, 2544–2550. old.
- [6] G. Bradski. „The OpenCV Library”. *Dr. Dobb’s Journal of Software Tools* (2000).
- [7] Floris van Breugel, Yuying Liu, Bingni W. Brunton és J. Nathan Kutz. „PyNumDiff: A Python package for numerical differentiation of noisy time-series data”. *Journal of Open Source Software* 7.71 (2022), 4078. old.
- [8] T.H. Cormen, C.E. Leiserson, R.L. Rivest és C. Stein. *Introduction to Algorithms, fourth edition*. MIT Press, 2022.
- [9] Jane Cullum. „Numerical Differentiation and Regularization”. *SIAM Journal on Numerical Analysis* 8.2 (1971), 254–265. old.
- [10] P.J. Davis. *Circulant Matrices by Philip J. Davis*. Chelsea Publishing Series. Chelsea, 1994. ISBN: 9780828403382. URL: <https://books.google.hu/books?id=BDgZwUSJSh8C>.
- [11] Steven Diamond és Stephen Boyd. „CVXPY: A Python-Embedded Modeling Language for Convex Optimization”. *Journal of machine learning research : JMLR* 17 (2016. márc.).
- [12] Matthew Field, David Stirling, Fazel Naghdy és Zengxi Pan. „Motion capture in robotics review”. *Proceedings of 2009 IEEE International Conference on Control and Automation*. 2009, 1697–1702. old.
- [13] Jr. Frank E. Harrell. *Regression Modeling Strategies*. Springer Series in Statistics. Springer Cham, 2015. ISBN: 978-3-319-19425-7.
- [14] Michael George, Babita Jose és Jimson Mathew. „Performance Evaluation of KCF based Trackers using VOT Dataset”. *Procedia Computer Science* 125 (2018. jan.), 560–567. old.

- [15] H. Grabner és H. Bischof. „On-line Boosting and Vision”. *Proceedings of 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’06)*. 1. köt. 2006, 260–267. old.
- [16] Helmut Grabner, Michael Grabner és Horst Bischof. „Real-Time Tracking via On-line Boosting”. 1. köt. 2006. jan., 47–56. old.
- [17] M. Gruber. *Improving Efficiency by Shrinkage: The James–Stein and Ridge Regression Estimators*. Statistics: A Series of Textbooks and Monographs. Taylor & Francis, 1998. ISBN: 9780824701567.
- [18] Aaron M. T. Harmer és Daniel B. Thomas. „pathtrackr: An r package for video tracking and analysing animal movement”. *Methods in Ecology and Evolution* 10.8 (2019), 1196–1202. old.
- [19] João F. Henriques, Rui Caseiro, Pedro Martins és Jorge Batista. „High-Speed Tracking with Kernelized Correlation Filters”. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.3 (2015), 583–596. old.
- [20] Francois Hild és Stéphane Roux. „Digital image correlation”. *Optical methods for solid mechanics. A full-field approach* 367 (2012), 183–228. old.
- [21] J. D. Hunter. „Matplotlib: A 2D graphics environment”. *Computing in Science & Engineering* 9.3 (2007), 90–95. old.
- [22] Zdenek Kalal, Krystian Mikolajczyk és Jiri Matas. „Forward-Backward Error: Automatic Detection of Tracking Failures”. *Proceedings of 2010 20th International Conference on Pattern Recognition*. 2010, 2756–2759. old.
- [23] Zdenek Kalal, Krystian Mikolajczyk és Jiri Matas. „Tracking-Learning-Detection”. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34.7 (2012), 1409–1422. old.
- [24] Alexander Koehnson, Jannis Kambach és Sebastian Büsse. „Step by step and frame by frame – Workflow for efficient motion tracking of high-speed movements in animals”. *Zoology* 141 (2020), 125800. old.
- [25] Riverbank Computing Limited. „PyQt5 Reference Guide”. (2016). URL: <https://www.riverbankcomputing.com/static/Docs/PyQt5/>.
- [26] Bruce D. Lucas és Takeo Kanade. „An Iterative Image Registration Technique with an Application to Stereo Vision”. *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2. IJCAI’81*. Vancouver, BC, Canada: Morgan Kaufmann Publishers Inc., 1981, 674–679. old.
- [27] Alan Lukezic, Tomas Vojir, Luka Cehovin Zajc, Jiri Matas és Matej Kristan. „Discriminative Correlation Filter With Channel and Spatial Reliability”. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017. júl.

- [28] Wes McKinney. „Data Structures for Statistical Computing in Python”. *Proceedings of the 9th Python in Science Conference*. Szerk. Stéfan van der Walt és Jarrod Millman. 2010, 56–61. old.
- [29] Alberto Menache. *Understanding motion capture for computer animation*. Elsevier, 2011.
- [30] Matteo Menolotto, Dimitrios-Sokratis Komaris, Salvatore Tedesco, Brendan O’Flynn és Michael Walsh. „Motion Capture Technology in Industrial Applications: A Systematic Review”. *Sensors* 20.19 (2020).
- [31] Joseph Morlier és Guilhem Michon. „Virtual Vibration Measurement Using KLT Motion Tracking Algorithm”. *Journal of Dynamic Systems, Measurement, and Control* 132.1 (2009. dec.).
- [32] Gergely Nagymáté és Rita M Kiss. „Application of OptiTrack motion capture systems in human movement analysis: A systematic literature review”. *Recent Innovations in Mechatronics* 5.1. (2018), 1–9. old.
- [33] Bing Pan, Kemaio Qian, Huimin Xie és Anand Asundi. „Two-dimensional digital image correlation for in-plane displacement and strain measurement: a review”. *Measurement Science and Technology* 20.6 (2009. ápr.), 62001. old.
- [34] Verónica Panadeiro, Alvaro Rodriguez, Jason Henry, Donald Wlodkowic és Magnus Andersson. „A review of 28 free animal-tracking software applications: current features and limitations”. *Lab Animal* 50 (2021. júl.), 1–9. old.
- [35] Julian T. Pittman és Katie M. Ichikawa. „iPhone® applications as versatile video tracking tools to analyze behavior in zebrafish (*Danio rerio*)”. *Pharmacology Biochemistry and Behavior* 106 (2013), 137–142. old.
- [36] NS Raghava, Kushagra Gupta, Ishita Kedia és Anirudh Goyal. „An Experimental Comparison of Different Object Tracking Algorithms”. *2020 International Conference on Communication and Signal Processing (ICCSP)*. 2020, 726–0730. old.
- [37] Alvaro Rodriguez, Hanqing Zhang, Jonatan Klaminder, Tomas Brodin, Patrik L. Andersson és Magnus Andersson. „ToxTrac: A fast and robust software for tracking organisms”. *Methods in Ecology and Evolution* 9.3 (2018), 460–464. old.
- [38] Abraham. Savitzky és M. J. E. Golay. „Smoothing and Differentiation of Data by Simplified Least Squares Procedures.” *Analytical Chemistry* 36.8 (1964), 1627–1639. old.
- [39] Vivek Hari Sridhar, Dominique G. Roche és Simon Gingins. „Tracktor: Image-based automated tracking of animal movement and behaviour”. *Methods in Ecology and Evolution* 10.6 (2019), 815–820. old.
- [40] The pandas development team. *pandas-dev/pandas: Pandas*. 1.4.2. verzió. 2022. ápr.

- [41] Silvia Todros, Silvia Pianigiani, Niccolò de Cesare, Piero Pavan és Arturo Natali. „Marker Tracking for Local Strain Measurement in Mechanical Testing of Biomedical Materials”. *Journal of Medical and Biological Engineering* 39 (2019. nov.), 764–772. old.
- [42] Floris Van Breugel, J. Nathan Kutz és Bingni W. Brunton. „Numerical Differentiation of Noisy Data: A Unifying Multi-Objective Optimization Framework”. *IEEE Access* 8 (2020), 196865–196877. old.
- [43] P.A. Viola, John Platt és C. Zhang. „Multiple instance boosting for object detection”. *Proc. Adv. Neural Inf. Process. Syst.* 18 (2006. jan.), 1417–1426. old.