KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

Ernestas Toliatas

KREPŠINIO TRENIRUOTĖS METIMŲ PATAIKYMO AUTOMATINIO ATPAŽINIMO TIKSLUMO TYRIMAS

Baigiamasis magistro projektas

Vadovas
doc. dr. Eimutis Karčiauskas

Kaunas, 2018

KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

KREPŠINIO TRENIRUOTĖS METIMŲ PATAIKYMO AUTOMATINIO ATPAŽINIMO TIKSLUMO TYRIMAS

Baigiamasis magistro projektas

Programų sistemų inžinerija (kodas 621E16001)

Vadovas
(parašas) doc. dr. Eimutis Karčiauskas
(data)

Recenzentas
(parašas) doc. dr. Antanas Lenkevičius
(data)

Projektą atliko
(parašas) Ernestas Toliatas
(data)

Kaunas, 2018

Kauno technologijos universitetas

Informatikos fakultetas

Ernestas Toliatas

Programų sistemų inžinerija (621E16001)

Krepšinio treniruotės metimų pataikymo automatinio atpažinimo tikslumo tyrimas

Akademinio sąžiningumo deklaracija

20 ___ . _____ ____ .
Kaunas

Patvirtinu, kad mano, Ernestas Toliato, baigiamasis projektas tema „Krepšinio treniruotės metimų pataikymo automatinio atpažinimo tikslumo tyrimas“ yra parašytas visiškai savarankiškai ir visi pateikti duomenys ar tyrimų rezultatai yra teisingi ir gauti sąžiningai. Šiame darbe nei viena dalis nėra plagijuota nuo jokių spausdintinių ar internetinių šaltinių, visos kitų šaltinių tiesioginės ir netiesioginės citatos nurodytos literatūros nuorodose. Įstatymų nenumatytų piniginių sumų už šį darbą niekam nesu mokėjęs.

Aš suprantu, kad išaiškėjus nesąžiningumo faktui, man bus taikomos nuobaudos, remiantis Kauno technologijos universitete galiojančia tvarka.

_____ _____
(vardą ir pavardę įrašyti ranka) (parašas)

Turinys

Toliatas, Ernestas. Krepšinio treniruotės metimų pataikymo automatinio atpažinimo tikslumo tyrimas. Magistro baigiamasis projektas / Vadovas doc. dr. Eimutis Karčiauskas; Kauno technologijos universitetas, Informatikos fakultetas.

Studijų kryptis ir sritis (studijų krypčių grupė): Programų sistemų inžinerija

Reikšminiai žodžiai: *Automatinis mestų ir pataikytų metimų atpažinimas, automatinis mestų ir pataikytų metimų skaičiavimas, automatinis objektų sekimas, automatinė krepšinio treniruočių analizė realiu laiku, automatinis spalvų filtravimas, HSV paletė, OpenCV.*

Kaunas, 2018. 60 puslapiai.

SANTRAUKA

Šiame magistro darbe buvo sukurta metimų pataikymo automatinio atpažinimo sistema (MPAAS), kuri veikia realiu laiku arba video įrašą paleidžiant iš failo, taip pat atliktas jos tikslumo tyrimas, bei išanalizuoti objektų sekimo metodai.

Sistema buvo sukurta naudojant atviro kodo biblioteką *OpenCV* ir trečiųjų šalių programinę įrangą *IP WEBCAM,* kuri buvo reikalinga tam kad telefono kameros transliuojamą video medžiagą būtų galima pasiekti, naudojant *OpenCV* ir *C++*. Todėl prieš pasiruošiant eksperimentui, buvo išanalizuoti duomenų perdavimo atvejai, naudojant *wi-fi* bei *bluetooth* tinklus.

Sukurta objektų sekimo technologija naudojant spalvų ir judėjimo atpažinimo metodus, kartu su automatiniu spalvų filtravimu. Sistema gali sekti ribotą kiekį asmenų, bei krepšinio kamuolį.

Išanalizuotas sistemos veikimas, kai kamera pastatyta iš šono ir galo, kai yra sekamas tik krepšinio kamuolys ir kai sekami visus asmenys kartu su žaidimo įrankiu. Analogiškas tyrimas atliktas, kai yra naudojamos dvi kameros. Galutinėje sistemos versijoje, galinėje kameroje yra sekami visi asmenys bei krepšinio kamuolys, o šoninėje sekamas tik krepšinio kamuolys. Yra skaičiuojami mesti ir pataikyti metimai, atliekamas statistinis vertinimas ir tikrinama ar kiekvienas asmuo laikosi nurodytų tikslų, kurie įvedami treniruotės pradžioje.

Ernestas, Toliatas. Accuracy analysis of automatic counter of basketball practice shoots. Master's Final Degree Project / supervisor assoc. doc. dr. Eimutis Karčiauskas; Faculty of Informatic, Kaunas University of Technology.

Study field and area (study field group): Software engineering

Keywords: *automatic recognition of attempted and made shoots, automatic counting of attempted and made shoots, automatic tracking of objects, automatic analysis of basketball practice in the real time, auto filter of colors, pallete of HSV, OpenCV.*

Kaunas, 2018. 60 pages.

SUMMARY

In this master thesis automatic recognition system of made shoots (ARSMS) was created which works in the real time or from video file. Also analysis of ARSMS accuracy and tracking methods was made.

System was created using open source library *OpenCV* and third party software *IP WEBCAM*, which was needed to get video from telephone camera to *OpenCV* and *C++* programing language. Because of that data transferring methods were analyzed using *wi-fi* and *bluetooth* networks.

It was created object tracking technology which used recognition of colors and motion detection alongside with auto filter of colors. The system can track limited amount of people and basketball.

It was analyzed performance of software when one camera was used from side and behind when only ball was traced and ball with people. Analogical experiment was made for case when two cameras were used. The last version of software tracks people and the ball with two cameras. Camera from behind tracks people and ball, camera from side tracks only ball. Attempted and made shoots is being counted for every person and statistical evaluation is made. Also software tracks if every person is keeping up with goals which were set up before practice.

1. ĮVADAS

1.1 Įžanga ir tikslai

Bandymai užfiksuoti žmogaus judėjimą sporto renginiuose prasidėjo maždaug apie 1920 metus, lengvojoje atletikoje [1]. Buvo sukurtas modelis, skirtas nustatyti sprinto bėgimo rungties rezultatus. Nuo tada augo poreikis atlikti detalesnę analizę ir tiksliau stebėti sportininkus, kadangi sporto lygis vis augo. Futbole buvo sukurta vartų linijos stebėjimo technologija *GoalControl* [2]. Šią sistemą galima naudoti ir ledo ritulio varžybose. Tam kad pasiekti aukščiausius rezultatus, stipriausios pasaulio lygos komandos (NBA) treniruotėse naudoja įvairią programinę įrangą. Ji analizuoja varžybas, treniruotes, sportininkų sveikatos būklę ir t.t. . Tokioje programų sistemoje yra nustatomi žaidėjų atrankos kriterijai, bei tikslai, kurios reikia įgyvendinti. Vienas iš pavyzdžių - *FusionSports* [3]. MPAAS buvo sukurta tam kad supaprastinti atliekamų ir pataikytų metimų automatinį skaičiavimą. Ši sistema gali būti naudojama be jokios specialios įrangos. Pakanka kompiuterio ir dviejų telefono kamerų. Taip pat yra siekiama sukurti galimybę atlikti krepšininkų analizę nuotoliniu būdu, bei rezultatams priskirti konkrečią skaitinę išraišką. Tokio pobūdžio treniruotėms analizuoti dažniausiai naudojamos bent dvi kameros. Kuriant sporto analizės sistemą labai svarbu atpažinti sportininkus ir žaidimo įrankį. Andrii Maksai, Xinchao Wang, Pascal Fua (2015) [4] pateikė universalų kamuolio sekimo algoritmą, kuris tinka didžiajai daliai komandinių arba individualių sporto šakų. Kamble, Keskar, Bhurchandi (2017) [5] sugrupavo pagal kategorijas didžiąją dalį technologijų, kurios buvo sukurtos sekti įvairių sportų šakų žaidimo įrankius, bei nurodydami jų galimybes, privalumus, bei trūkumus. Viena iš populiariausių kamuolio sekimo technologijų yra *EyeHawk* (2001) [6], kuri buvo sukurta kriketo transliacijoms, tačiau ilgainiui sistema buvo modifikuota ir pritaikyta futbolui, bei tenisui. Sistema taip pat fiksuoja ir pateikia statistinius teniso ir kriketo varžybose.

Šio darbo pirmame skyriuje yra pristatomi darbo tikslai, bei pateikiami projekto vykdymo etapai. Antrame skyriuje aprašyti sekimo metodai. Trečiame skyriuje nurodyta sistemos projektinė dalis. Ketvirtame skyriuje aprašytas metimų skaičiavimo algoritmas, kai yra naudojama viena ir dvi kameros. Penktame skyriuje suformuluotos darbo išvados. Šeštame skyriuje nurodyta literatūra, o septintame priedai.

Šio darbo tikslai:

1. Sukurti programinę įrangą, kuri atliktų mestų ir pataikytų metimų skaičiavimą kiekvienam žmogui, paleidžiant transliaciją iš video failo arba filmuojant treniruotę tiesiogiai.
2. Sukurti programinę įranga, kuria būtų lengva naudotis, be specialios papildomos įrangos.
3. Ištirti objekto atpažinimo metodus ir nustatyti tinkamiausią transliacijos analizei.
4. Rasti optimalų kamerų kiekį, kad sistema veiktų realiu laiku kompiuteryje, kurio techniniai resursai yra:
    - Procesorius: i3, 2GHZ

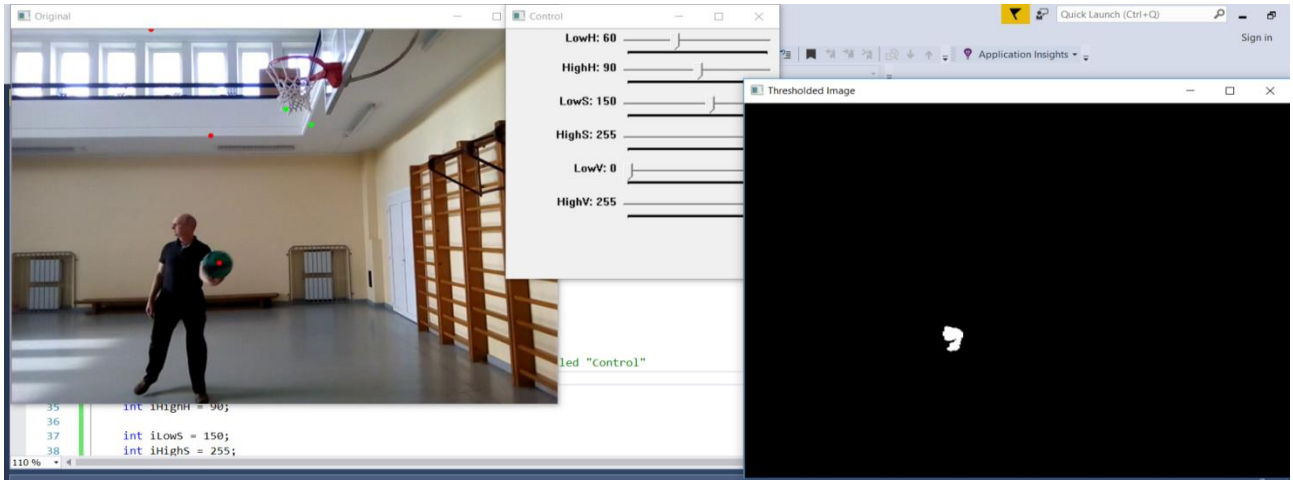
- 8 GB RAM

Darbo uždaviniai:

1. Išanalizuoti atpažinimo metodus, kurie yra siūlomi *OpenCV* bibliotekoje.
2. Išanalizuoti spalvų ir judėjimo atpažinimo technologijas
3. Suprojektuoti ir sukurti sistemą, kuri atpažintų mestus ir pataikytus metimus
4. Sukurti ir užkuoduoti metimų skaičiavimo algoritmą

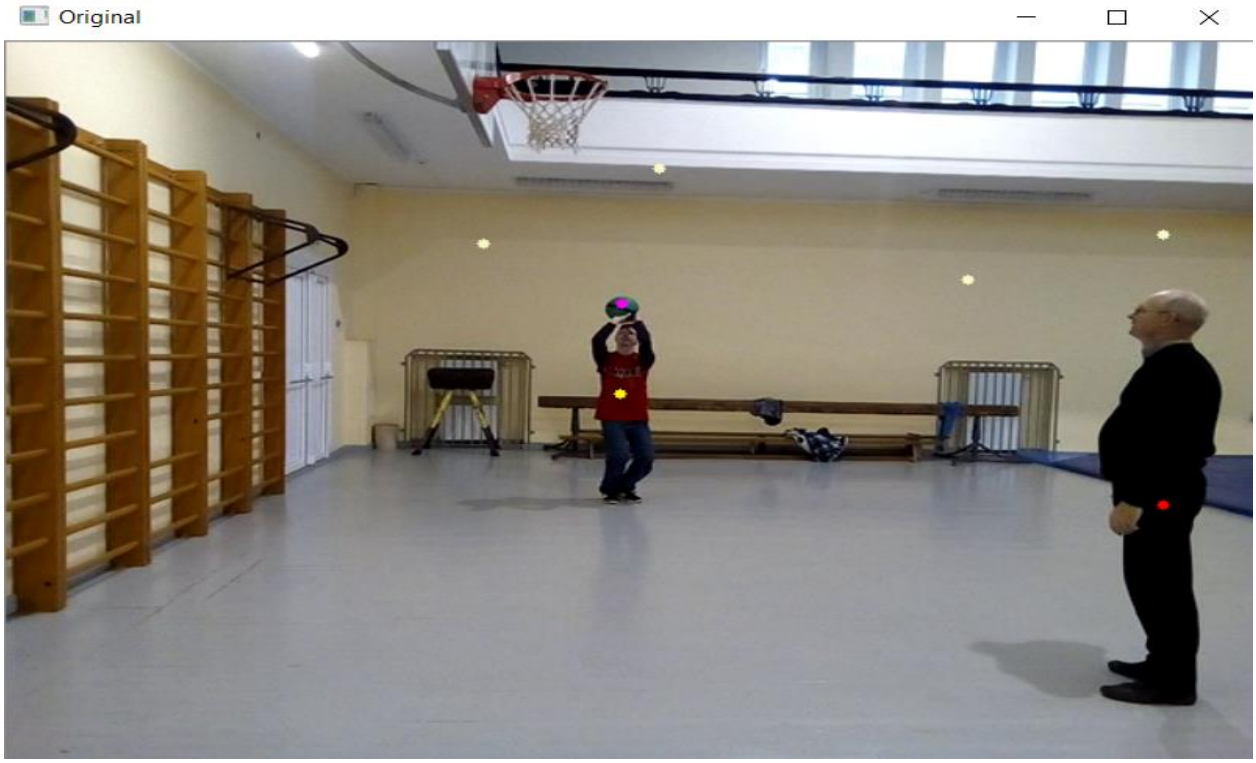
1.2 Projekto vystymo etapai

Atliekant eksperimentą buvo sukurtos keturios sistemos versijos. Pirma sistemos versija skaičiuoja atliktus ir pataikytus metimus, tačiau negali atpažinti asmenų. Todėl buvo sekamas tik krepšinio kamuolys, bei stebimos dvi zonos: viršutinė kadro dalis ir zona po krepšiu. Abi pozicijos yra statiškos, bei apibrėžiamos keturiais pikselių taškais. Stebint viršutinį plotą yra atpažįstamas atliktas metimas, o zoną po krepšiu – pataikytas. Filmuojama iš šono viena kamera (žr. 1.2.1 pav.).



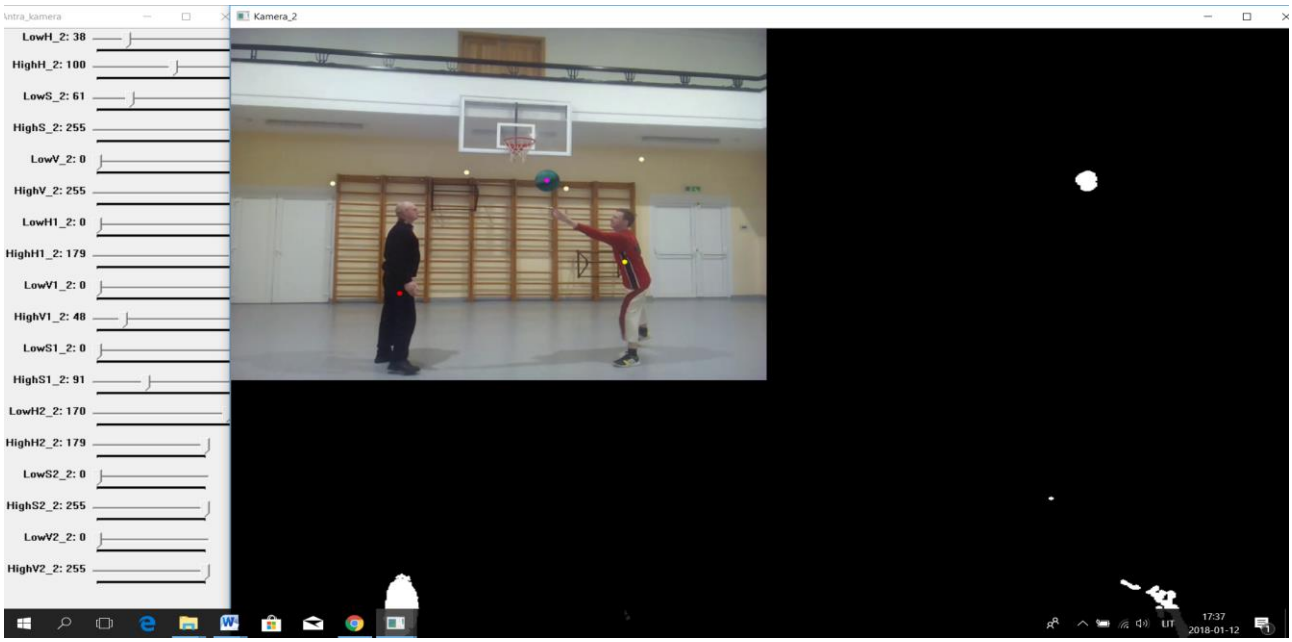
1.2.1 pav. Pirma programos versija

Antroje programos versijoje buvo pradėti sekti ir asmenys. Sistema gali sekti ribotą kiekį žmonių ir krepšinio kamuolį. Dėl to metimo pozicijos yra dinamiškos. Atlikto metimo zona nustatoma, atsižvelgiant į asmens padėtį kadre, o pataikyto metimo nustatoma kaip ir pirmoje versijoje. Transliacija vis dar buvo vykdoma per vieną kamerą, tačiau buvo atlikti bandymai, kai kameros pozicija yra iš šono ir iš galo. Sistemos veikimas buvo išbandytas dviem atvejais, kai filmuojama iš šono ir galo (žr. 1.2.2 pav.).



1.2.2 pav. Antra programos versija

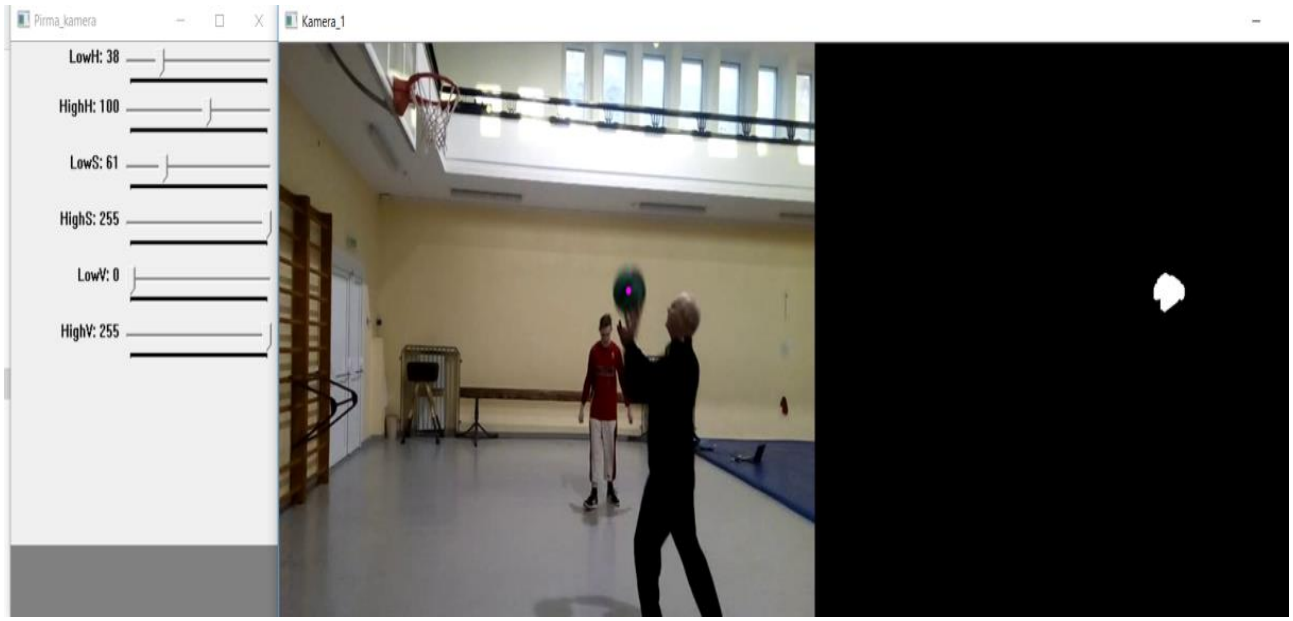
Trečioje sistemos versijoje buvo pridėta antra kamera. Vienu metu filmuojama iš šono ir iš galo. Abiejose transliacijose buvo sekami asmenys ir krepšinio kamuolys, atliekamų metimų pozicijos – dinamiškos, o pataikytų - statiškos. Taip pat buvo pradėtas stebėti sistemos veikimas realiu laiku (žr. 1.2.3 ir 1.2.4 pav.).



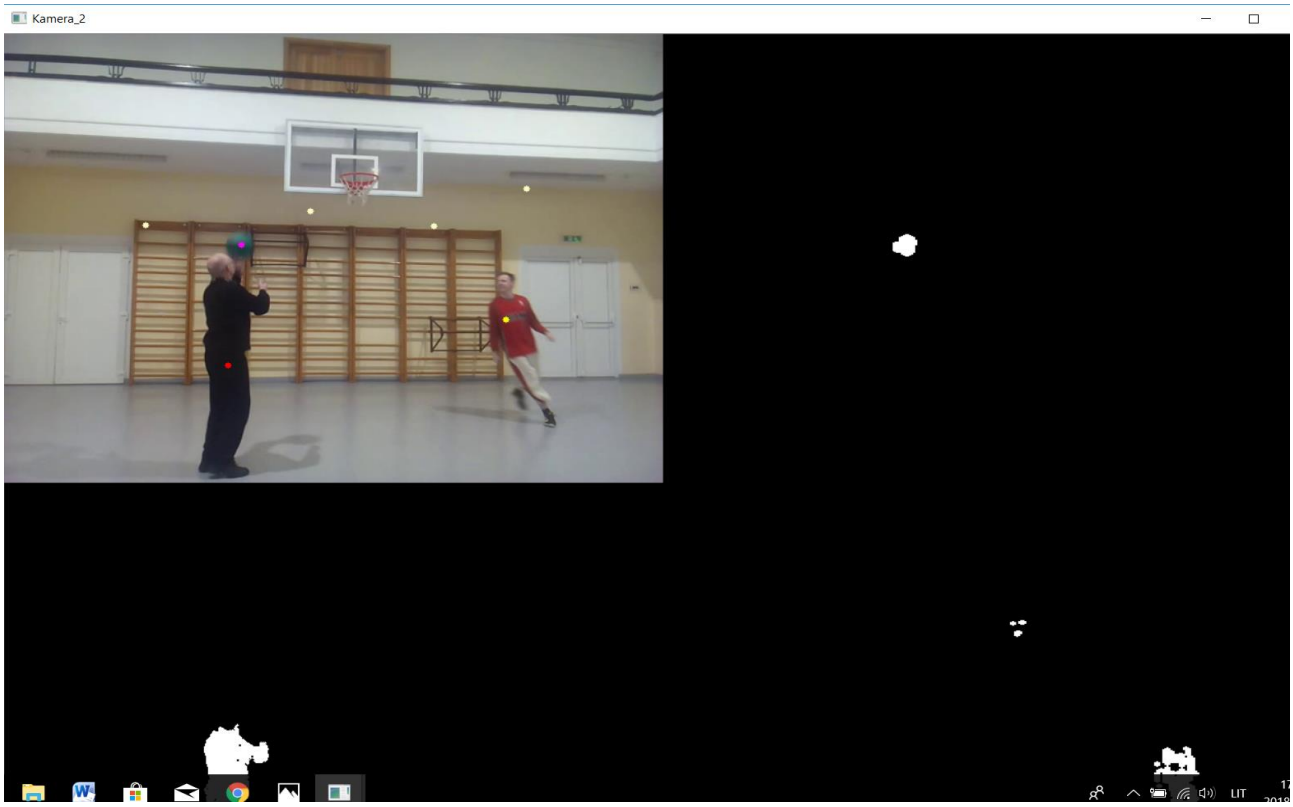
1.2.3 pav. Trečios programos versijos galinė kamera

1.2.4 pav. Trečios programos versijos šoninė kamera

Ketvirtoje sistemos versijoje buvo atliktos korekcijos zonose, kurias stebi kameros. Galinėje kameroje viskas išliko taip kaip ir trečiame variante, tačiau šoninėje metimų pozicijos pradėtos apibrėžti, kaip pirmoje versijoje. Tai buvo nuspręsta padaryti todėl, kad padidinti sistemos veikimo greitį ir palengvinti jos panaudojamumą. Dėl veikimo greičio padidėjimo, rezultatai naudojant programą realiu laiku taip pat pagerėjo (žr. 1.2.5 ir 1.2.6 pav.).



1.2.5 pav. Ketvirtos programos versijos šoninė kamera



1.2.6 pav. Ketvirtos programos versijos galinė kamera

Sistemos greičiui tikrinti buvo naudoti penki video failai ir stebimas laikas per kurį sistema įvykdo transliaciją. Po to gauti rezultatai lyginami su tikromis video įrašo trukmėmis. Reikia pastebėti, kad greitis buvo vertinamas tik 3-ai ir 4-ai versijai.

2. AUTOMATIZUOTI OBJEKTŲ ATPAŽINIMO METODAI

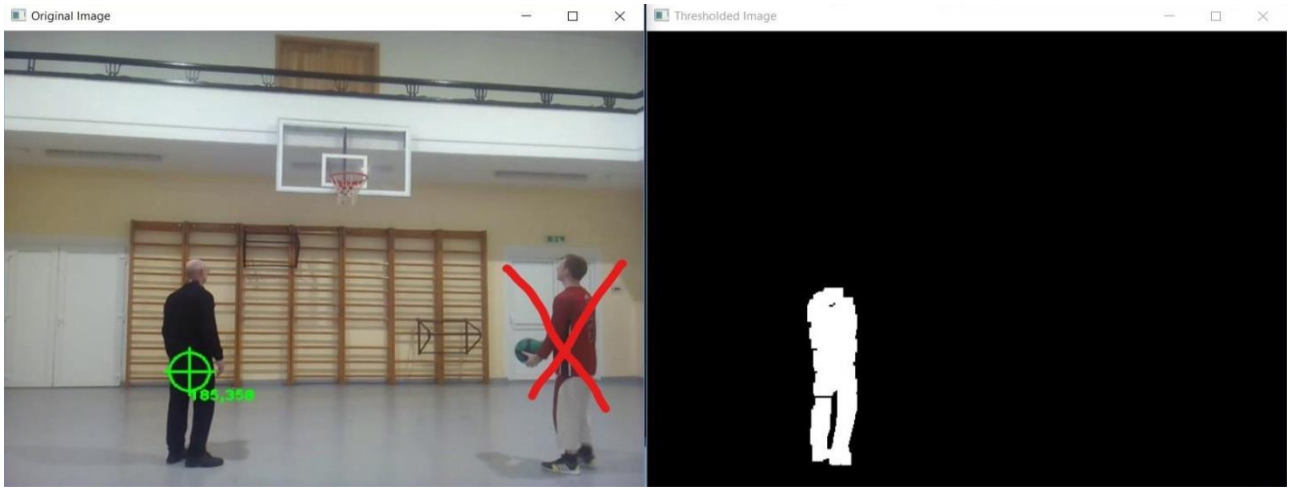
2.1 Įžanga

Automatizuoti asmenų ir krepšinio kamuolio atpažinimo metodai yra įvairūs. Objektus galima sekti pagal jų spalvas arba judėjimą, lyginant esamą kadrą su prieš tai buvusiu, arba apjungti abu algoritmus. Taip pat *OpenCV* bibliotekoje, papildomų modulių kategorijoje, yra siūlomi penki objektų atpažinimo būdai [7] : *KFC, TLD, MedianFlow, Mil, (angl. Boosting),* tačiau šių metodų veikimas nėra tobulas.

2.2 Automatinis ir rankinis HSV spalvų reikšmių filtravimas

Spalvų atpažinimas MPAAS buvo vienas iš pagrindinių objektų sekimo faktorių. *HSV* spalvų paletės reikšmes galima parinkti rankiniu būdu, naudojant slankiklius *(angl. trackbars)*. Toks būdas naudingas, kai reikia sekti mažus objektus arba daiktus, kurie turi keletą atspalvių. Tačiau dėl apšvietimo arba garso trikdžių kai kuriais atvejais teorinės reikšmės praktikoje netiks. Dar vienas trūkumas - *OpenCV* biblioteka atpažįsta *HSV* spalvas ne tradiciniu formatu (žr. [8]). *Hue* priklauso nuo pasirinkto bitų kiekio. Jeigu kadrai yra 16-bitų, tai *hue* reikšmė yra tokia pat, kaip ir standartiniame variante, tačiau jeigu 8-bitų, tai tikrą skačių reikia dalinti iš dviejų ir jo amplitudė kinta nuo 0 iki 180 laipsnių. *Sat* ir *val* reikšmės kinta nuo 0 iki 255 nepriklausomai nuo bitų kiekio, o konvertavimas atliekamas pagal proporcijas.

Tam kad būtų galima naudoti rankinį būdą, reikia papildomų žinių apie spalvų paletę, todėl kaip alternatyvą galima naudoti automatį *HSV* filtrą. Šis būdas yra naudingas, kai kadre esantys objektai yra dideli, bei vienos spalvos (sportininkai su vienodomis aprangomis). Tačiau naudojant tokį metodą yra sudėtinga sekti spalvotus arba mažus objektus, pavyzdžiui, krepšinio kamuolį (žr. 2.2.1 pav.). Taip pat naudojant tokį būdą svarbu, kad fonas ir apšvietimas būtų pastovus, kadangi įrankis parenka optimalų variantą, nustatant mažiausias ir didžiausias reikšmes, todėl net ir nedideli pakitimai sutrikdo objekto sekimą. Taip pat toks metodas reikalauja gilesnių programavimo žinių.



2.2.1 pav. Auto filtravimo pavyzdys

Norint sukurti automatinį spalvų filtrą reikalingi du metodai. Pirmasis turi užfiksuoti sekamo objekto esamą zoną, o antra funkcija turi peržiūrėti visas tos srities pikselių reikšmes. Objekto

atpažinimo algoritmą galima sukurti naudojant specialių *OpenCV* būdą *setmousecallback()* [9]. Šiam metodui reikia nurodyti lango pavadinimą, kuriame vykdoma transliacija, sukurtą papildomą funkciją, kuri grąžina vartotojo veiksmus atliktus pele ir kintamąjį, kuriame yra fiksuojamas kadras. Metodikos realizavimo kreipinys parodytas žemiau.

setMouseCallback(LangoPavadinimas, Vartotojo_veiksmu_nustatymas, &kadro_kintamasis);

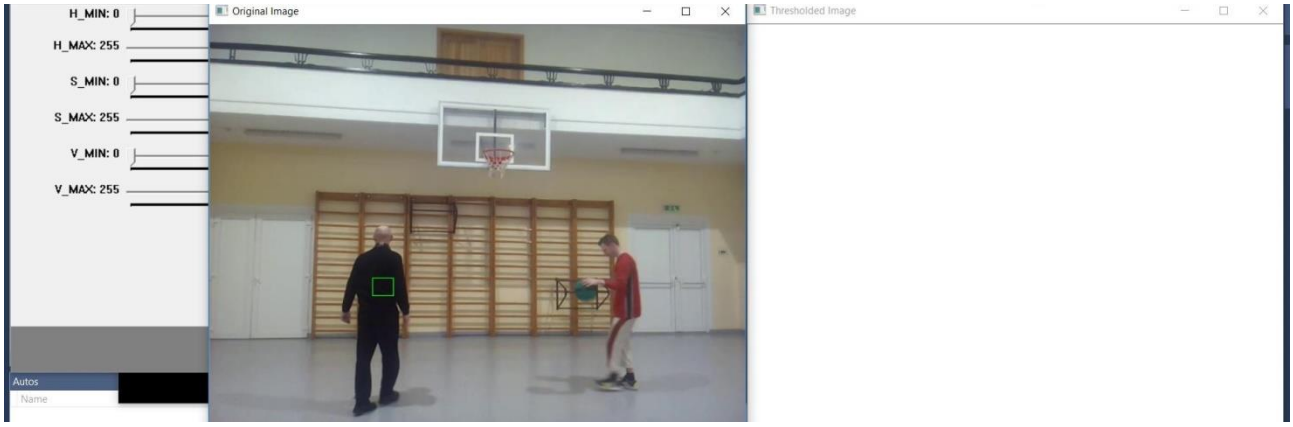
2.2.1

Iš 2.2.1 pvz. matyti, kad yra reikalingas papildomas metodas vartotojo veiksmams nustatyti. Nors į *Vartotojo_veiksmu_nustaymas* funkciją yra kreipiamasi nenurodant nieko, tačiau jos aprašyme kintamieji yra nurodomi standartiškai. Šis būdas yra naudojamas kartu su C++ programavimo kalba, kai reikia nustatyti pelės paspaudimus (žr. 2.2.2 pvz.).

void Vartotojo_veiksmu_nustatymas(int event, int x, int y, int flags, void kadro_kintamasis)*

2.2.2

Pirmi keturi kintamieji yra visada tokie patys. *Event* nustato, į kurią pusę vartotojas pajudino pelę, *x* ir *y* saugo pelės judejimo koordinates, *flags* atskiria kuris mygtukas buvo paspaustas (plačiau žr. [10]). Tada nurodoma kokia nors piešimo funkcija, pavyzdžiui, *Rect()* tam kad pažymėti pasirinktą zoną, o *Point()* [11] metodas parenka tašką, ties kuriuo buvo paspaustas pelės klavišas. Norint pasirnkti tokią sritį daug kartų nestabdant transliacijos reikia sukurti reikšmių atkūrimo sąlygą, kuri įvykdoma paspaudžiant dešinyjį mygtuką. Kadangi reikalingas visas plotas, o ne tik vienas pikselis, reikia atpažinti pelės judėjimą, kuomet yra atliekamas kairysis paspaudimas. Tada dar vienas *bool* kintamasis įgauna *false* reikšmę ir nupiešiamas kvadratas. 2.2.2 pav. parodytas metodo veikimo rezultatas. Taip pat reikia pastebėti, kad pasirenkama zona turi būti be papildomų atspalvių.



2.2.2 pav. Zonos parinkimas auto filtrui

Kai yra užfiksuojama pasirinkta zona, reikia sužinoti visas jos pikselių reikšmes. Vienas iš būdų - „lizdinis“ [12] ciklas. Pavyzdžiui, pirmiausia pasirenkama *x* koordinatė ir surandamos visos įmanomos poros su *y*. Tada tas pats vyksta su antra reikšme ir taip toliau iki pabaigos. 2.2.3 pvz. parodytas kiekvieno pikselio pasiekiamumas.

```
  for (int i = rectangleROI.x; i<rectangleROI.x + rectangleROI.width; i++)
{
  for (int j = rectangleROI.y; j<rectangleROI.y + rectangleROI.height; j++)
{
                Hue.push_back((int)hsv_frame.at<cv::Vec3b>(j, i)[0]);
                Sat.push_back((int)hsv_frame.at<cv::Vec3b>(j, i)[1]);
                Val.push_back((int)hsv_frame.at<cv::Vec3b>(j, i)[2]);
}
}
```

2.2.3

Vec3b [13] reiškia, kad kadras pasižymi trimis spalvų kanalais. Kiekvienas indeksas atitinka *hue*, *sat* ir *val* pikselių reikšmes. Sekančiame žingsnyje reikia surasti mažiausią ir didžiausią dydį, kurie yra sąraše, todėl galima naudoti *min_element()* [14] ir *max_element()* [15] funkcijas (žr. 2.2.4 pvz.).

```
Hue_MIN = *std::min_element(Hue_ROI.begin(), Hue_ROI.end());
Hue_MAX = *std::max_element(Hue_ROI.begin(), Hue_ROI.end());
Sat_MIN = *std::min_element(Sat_ROI.begin(), S_ROI.end());
Sat_MAX = *std::max_element(Sat_ROI.begin(), S_ROI.end());
Val_MIN = *std::min_element(Val_ROI.begin(), V_ROI.end());
Val_MAX = *std::max_element(Val_ROI.begin(), V_ROI.end());
```

2.2.4

Šis metodas buvo įdiegtas ketvirtoje MPAAS versijoje (žr. 1.2 sk.).

2.3 Spalvų atpažinimo technologija

Kiekvieną kadrą *OpenCV* fiksuoja *BGR* paletėje, kurią sudaro trys spalvų kanalai. Formatai be *BGR* gali būti *HSV*, *HSL*, *grayscale* ir kiti (plačiau žr. [16]). Tam kad konvertuoti kadrus iš vienos paletės į kitą yra naudojama *cvtcolor()* [17] funkcija. Metodui yra nurodomas esamo ir naujo kadro kintamasis, kuriame vaizdas bus išsaugotas nauju formatu (esant poreikiui kintamasis gali būti ir tas pats), konvertavimo reikšmė, kuri taip pat turi ir žodinę išraišką, ir spalvų kanalų kiekis, kuris gali būti ir nenurodomas, tada pagal nutylėjimą parenkamas toks pat, kaip ir šaltinio. 2.3.1 yra pateikiamas pakeitimo pavyzdys.

```
VideoCapture cap(transliacijos_saltinis);
Mat esamas_kadras;
bool skaityk_kadra = cap.read(esamas_kadras);
Mat pakeistas_kadras;
cvt(esamas_kadras, pakeistas_kadras, BGR2HSV, 3);
// čia BGR2HSV konvertavimo reikšmė, kurios skaitinė išraiška 6.
```

2.3.1

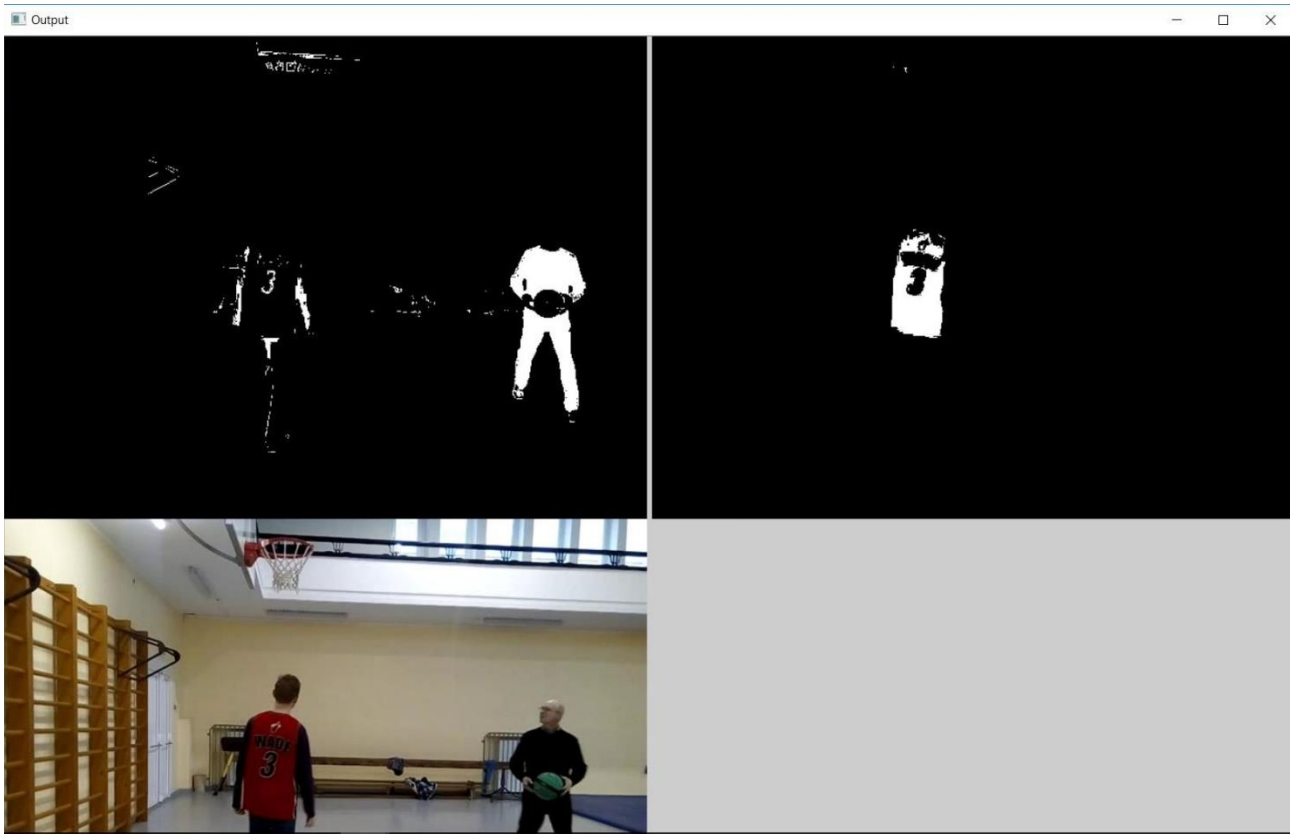
Atlikus paletės transformavimą reikia perduoti spalvų reikšmes, kad būtų galima gauti naują kadrą, kuris sudarytas iš baltos ir juodos spalvos. Tuomet yra tikrinamas paveikslėlio pikselių intensyvumo reikšmės. Zona, kur dydžiai patenka į nurodytus intervalus yra balta, o likusi – juoda.

Esant poreikiui atspalvius galima pakeisti, pakeičiant taškų intensyvumo reikšmes. Svarbiausia, kad kadre vyrautų tik dvi spalvos, priešingu atveju objekto filtravimas gali būti sudėtingas, t.y. *SAT* ir *VAL* reikšmės turi būti nulinės. Tam kad parinkti reikšmes galima naudoti *inrange()* [18] funkciją. Kuriai reikia nurodyti šaltinio kadrą, mažiausią ir didžiausią dydžius visiems kanalams (paprastai pateikiamos vektoriumi) ir kintamąjį, kuriame bus saugomas dviejų spalvų paveikslėlis. (žr. 2.3.2 pvz.)

```
Mat juodaibaltas; // Kintamasis dviejų spalvų kadrui laikyti
inRange(pakeistas_kadras, Scalar(38, 50, 0), Scalar(100, 255, 255), juodaibaltas);
```

2.3.2

Atlikus šiuos veiksmus jau galima išskirti objektą iš aplinkos, tačiau tinkamai sekti vis tiek nepavyktų. Reikia filtravimo algoritmo, kuris pašalintų triukšmo sukeltą neteisingą pikselių intensyvumą. (žr. 2.3.1 pav.)



2.3.1 pav. Objekto išryškimas nepašalinus triukšmo sukeliamų netikslumų

Norint pašalinti triukšmo trikdžius galima naudoti *erode()* ir *dilate()* funkcijas. Tai rinkinys proceso operacijų, kai kadras yra analizuojamas formomis. Abi operacijos priima kadrą ir grąžina išfiltruotą. T.y. panaikina triukšmą, neteisingus pikselių intensyvumus, izoliuoja elementus ir sujungia skirtingus elementus paveikslėliuose. [19] Abiems metodams pateikiami tokie kintamieji:

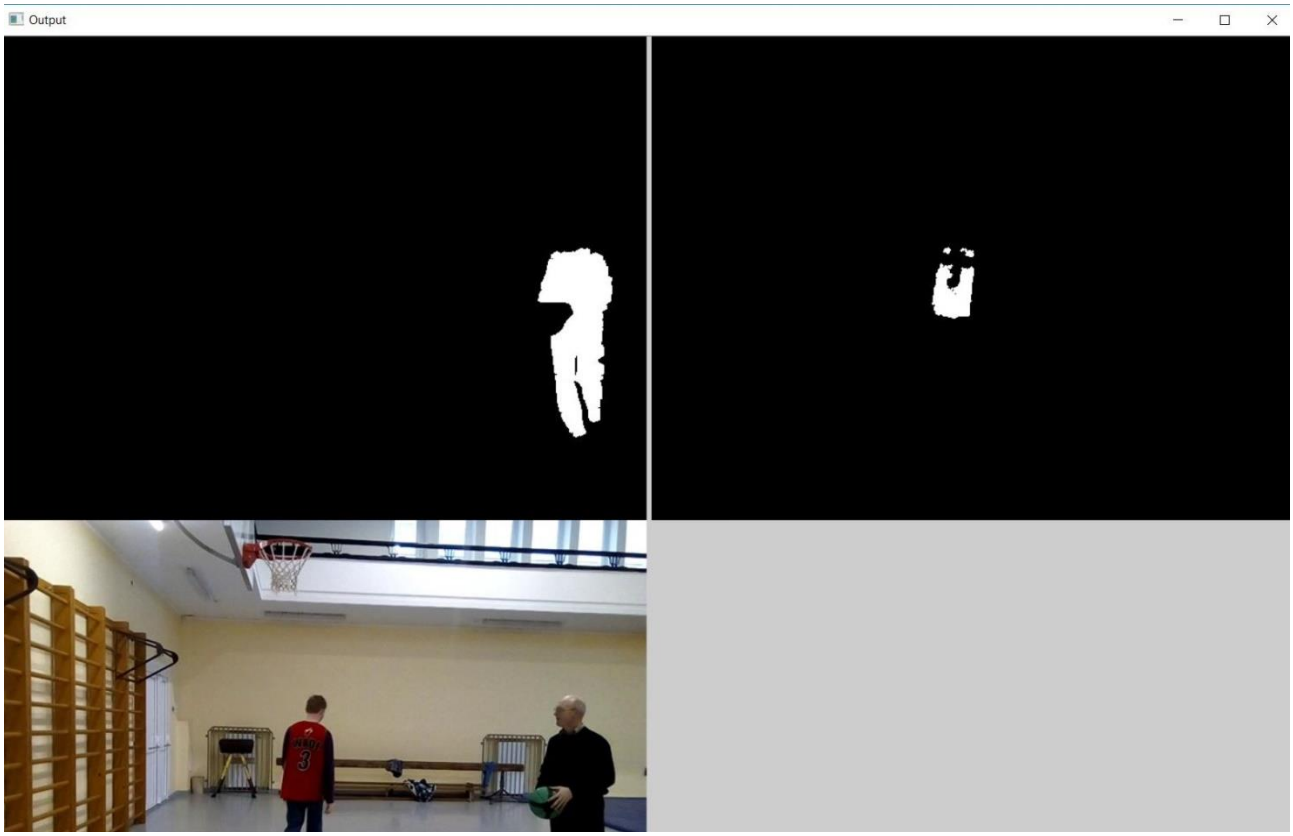
1. Šaltinis. Kadras, kuris gautas iš video transliacijos
2. Gautas_kadras. Kintamasis, kuris saugo išfiltruotą kadrą.
3. Elementas. Nurodomas pikselių plotas spalvų, kurios turi būti pašalintos

4. Iteracijų kiekis.
5. Rėmelių tipas.
6. Rėmelių reikšmė.[20]

erode(Šaltinis, Gautas_kadras, getStructuringElement(MORPH_ELLIPSE, Size(Elementas, Elementas)));

dilate(Šaltinis, Gautas_kadras, getStructuringElement(MORPH_ELLIPSE, Size(Elementas, Elementas)));

2.3.3



2.3.2 pav. Objekto išryškimas pašalinus triukšmo sukeliamus netikslumus

Sekančiame veiksme reikia gauti objektų koordinates. Vienas iš būdų naudoti *Moments* [21] funkciją. Šiam metodui perduodamas galutinai išfiltruotas paveikslėlis. Fiksuojami du momentai ir juodai-baltame lange matomos baltos spalvos plotas, kuris matuojamas pikseliais. Turint šias tris reikšmes galima gauti objekto koordinates ekrane. Tarkime pirmasis momentas yra lygus A, antrasis B, o visa matoma sritis C. Tada objekto X koordinatė bus lygi

$$X = \frac{A}{C},$$

o Y

$$Y = \frac{B}{C}.$$

Taip pat galima pridėti papildomą reikalavimą, kad ieškoti objekto koordinačių, kai plotas yra tam tikro dydžio, pavyzdžiui 10000 pikselių. 2.3.4 pvz. pateiktas programos kodo fragmentas (žr. 2.3.3 pav.).

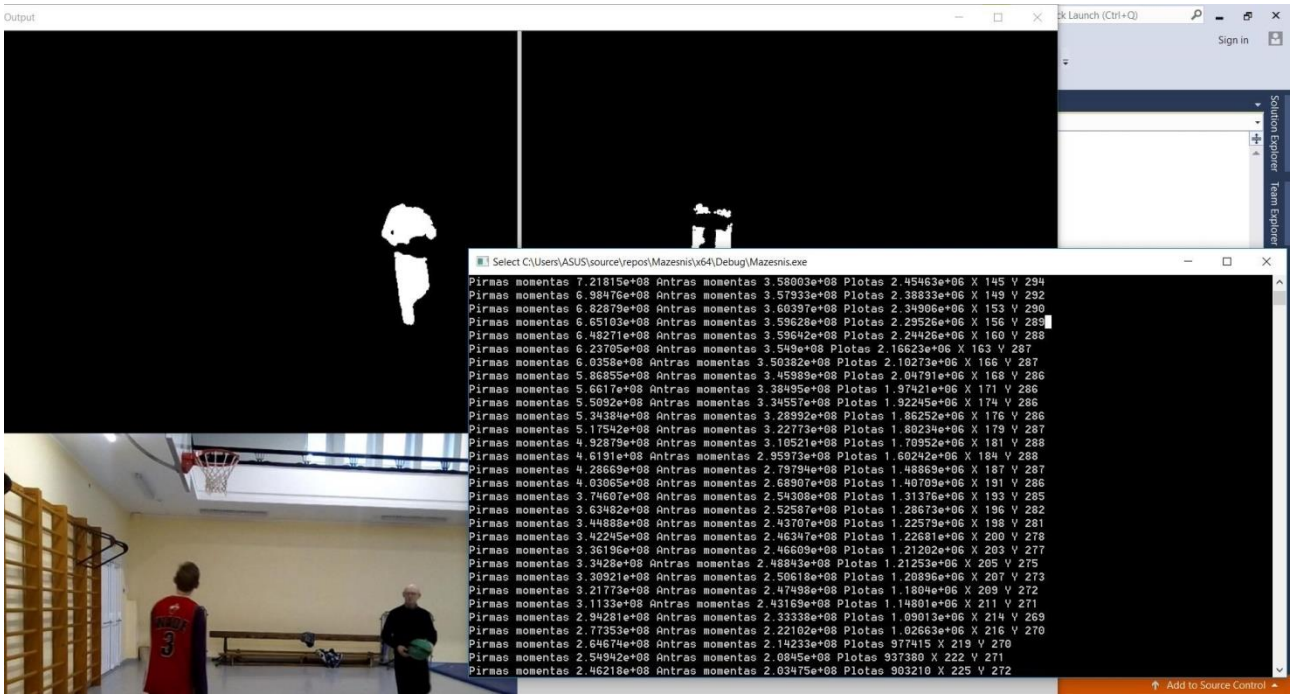
```
Moments oMoments = moments(Gautas_kadras);

double dM01 = oMoments.m01;
double dM10 = oMoments.m10;
double dArea = oMoments.m00;

if (dArea > 10000)
{

int posX = dM10 / dArea;
int posY = dM01 / dArea;
}
```

2.3.4



2.3.3 pav. Objektų sekimui reikalingos reikšmės

Tačiau pasirinkus tokį atpažinimo metodą, programų sistemą yra sudėtinga. Taip pat būtinai reikalingas pastovus fonas, kadangi net dėl menkiausių pakeitimų gali reikėti koreguoti spalvų reikšmes (žr. 1.2.3 pav.). Kitas trūkumas, kiekvienam skirtingo atspalvio objektui atpažinti reikalingas atskiras juodai-baltas langas. Dėl šios priežasties yra prarandamas sistemos veikimo greitis. Jeigu yra parenkamos tinkamos spalvų paletės reikšmės, metodas yra tikslus. Objektai esantys aplinkui esamą objektą yra ignoruojami. Taip pat programų sistema nustato, kada asmuo arba daiktas, nėra matomas, tada ploto reikšmė prilyginama nuliui ir sekimas sustabdomas iki tol, kol nepasirodo.

2.4 OpenCV bibliotekoje esantys atpažinimo metodai

2.4.1 Branduolių koreliacijos filtras (angl. kernels correlations filter)

2.4.1.1 Apibrėžimas. Matrica yra vadinama cirkuliacine arba Teoplitzo, jeigu jos išraiška yra

$$X = \begin{matrix} x_0 & x_{T-1} & x_{T-2} & \dots & x_1 \\ x_1 & x_0 & x_{T-1} & \dots & x_2 \\ x_2 & x_1 & x_0 & \dots & x_3 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{T-1} & x_{T-2} & x_{T-3} & \dots & x_0 \end{matrix} \text{ [22].}$$

2.4.1.2 Apibrėžimas. Failas yra laikomas RAW formato, jeigu jis saugo visą kadro vaizdo informaciją, kuri buvo užfiksuota kameros sensoriumi [23].

2.4.1.3 Apibrėžimas. Kosinusų langai gali būti apibrėžti tokia lygčių sistema

$$w(t) = \begin{cases} \sin\left(\frac{\frac{\pi}{2}}{\frac{t}{0.1T}}\right), 0 \le t < 0.1T \\ 1, \ 0.1T \le t < 0.9T \\ \cos\left(\frac{\frac{\pi}{2}}{\frac{t-0.9T}{0.1T}}\right), 0.9T \le t < 1 \end{cases}$$, čia T nusako lango kadro plotį, ir lygi analizuojamo signalo ilgiui [24].

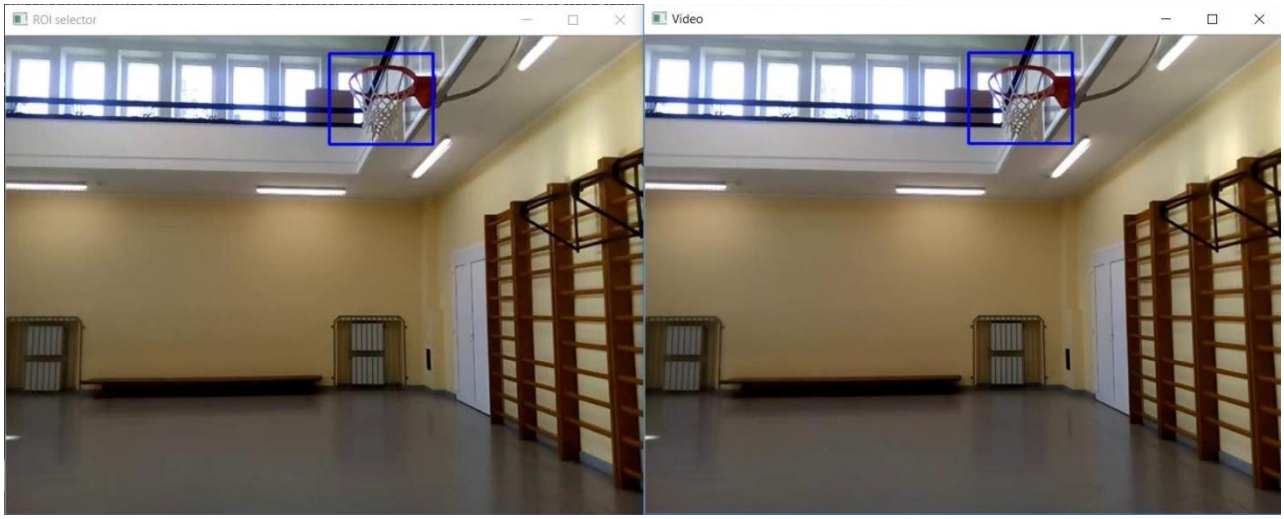
2.4.1.4 Apibrėžimas. Diskreti Forjė transformacija yra toks procesas, kurio metu kompleksinių skaičių seka $\{x_n\}$, $x_0, x_1, \dots, x_{N-1}$, kurios dydis yra N, yra transformuojama į kitą seką $\{X_k\}$, $X_0, X_1, \dots, X_{N-1}$. Transformacija apibrėžiama:

$$X_k = \sum_{n=0}^{N-1} x_n e^{\frac{-i2\pi nk}{N}} = \sum_{n=0}^{N-1} x_n \left[\cos\left(\frac{2\pi kn}{N}\right) - i \sin\left(\frac{2\pi kn}{N}\right)\right] \text{ [25].}$$

OpenCV biblioteka siūlo keletą jau sukurtų automatizuotų atpažinimo algoritmų [7]. Vienas iš jų *KCF* arba (*angl. kernels correlations filter).* Šis metodas naudoja cirkuliacinės matricos savybes, kad padidinti proceso greitį [26]. Paprastai funkcija veikia, kai yra nurodoma tam tikra *m x n* dydžio zona, o tos srities paveikslas sekamas stebint centrinę pikselių koordinatę. Kintamieji *m* ir *n* yra ilgis ir plotis matuojamas pikseliais. Iš nurodytos vietos algoritmas pasirenka skiriamuosius *RAW* kadro bruožus arba kanalų skaičių. Tada kiekvienam iš jų yra priskiriama tam tikra svorinė reikšmė, pasinaudojant kosinuso lango formulėmis. Cirkuliacinė matrica naudojama sužinoti visus įmanomus taikinio pakeitimus iš pradinio paveikslėlio. Koeficientas α_p užkoduoja galimus pavyzdžius, atsižvelgiant į pradinio kadro visus galimus pasikeitimus. Šis koeficientas yra lygus

$$\alpha_p = \frac{y_f}{k_f^{xx'} + \lambda},$$

čia y_f apibrėžia diskrečią Forjė transformaciją, o $k_f^{xx'}$ Gauso branduolio koreliacijos funkcija [27].



2.4.1.1 pav. *KCF* metodo veikimas

Tam kad naudoti *KCF* metodą, reikalingos papildomos *OpenCV* bibliotekos (*angl. contribution reprositories*), kurių nėra pagrindiniame pakete. Taip yra todėl, kad tai metodai, kurie nėra tobuli ir kūrėjai palieka galimybę apsispręsti patiems vartotojams ar jie nori juos naudoti. Šis algoritmas gaunamas iš *TrackerKCF* klasės (žr. [26]). 2.4.1.1 pav. parodytas 2.4.1.1 pvz. rezultatas.

```
Ptr<Tracker> tracker;
tracker = TrackerKCF::create(); // KCF metodas
 VideoCapture video("D:\\MyOpenCv\\Images\\failas46.3gp"); // video šaltinis
Mat frame; // pirmas kadras
bool ok = video.read(frame); // skaitomas kadras
Rect2d bbox(0, 0, 0, 0); //apibrėžiama pirminė zona pagal nutylėjimą
bbox = selectROI(frame, false); // zonos parinkimas iš pirmo kadro
rectangle(frame, bbox, Scalar(255, 0, 0), 2, 1); // piešiamas pasirinktas kvadratas
imshow("Tracking", frame); // rodomas pirmas kadras su nupieštu kvadratu
tracker->init(frame, bbox); // paruošiamas piešimas ant visų kadrų
while (true) // pradedamas transliacijos ciklas
{
   // kodas apie tai ką reikia padaryti kadrų sekai
}
```

2.4.1.1

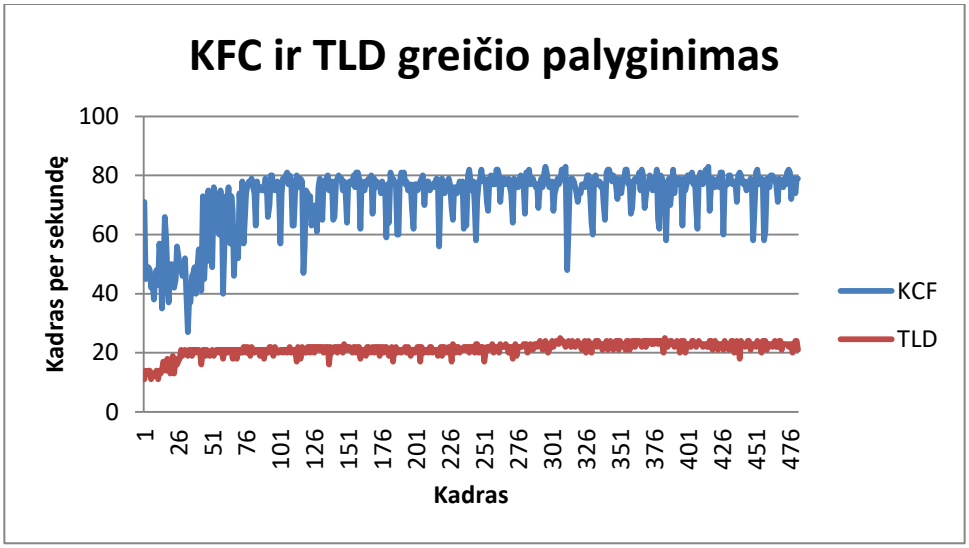
Pastaba. Kad *KCF* algoritmas veiktų, reikalinga ne senesnė nei 3.0 OpenCV versija, bei bent 2015 visual studio.

2.4.2 Išmokstamasis sekimo atpažinimo algortimas (angl. Tracking Learning Detection)

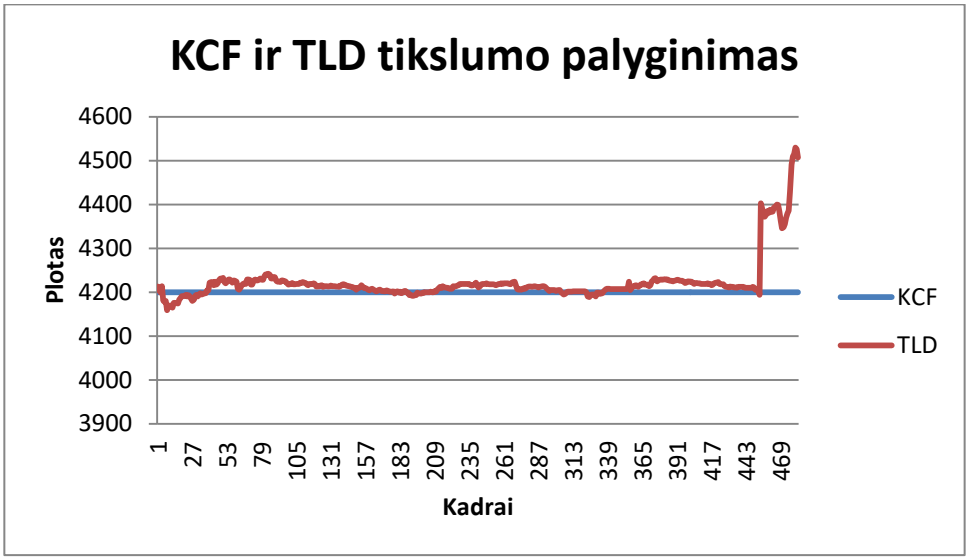
TLD (angl. Tracking Learning Detection) dar vienas atpažinimo metodas, kurį siūlo *OpenCV* biblioteka (žr. [7]). Šis algoritmas naudojamas, kai objektą reikia sekti ilgą laiką, kuris juda savavališkai judančioje neribotoje aplinkoje. Metodas seka taikinį ir palaipsniui kaupia informaciją

apie jo bruožus, tam kad būtų galima nuspėti daikto arba asmens elgseną, jeigu sekimo technologija neranda kadre esančio taikinio. Kiekvienas video kadras yra perduodamas sekimo ir atpažinimo technologijoms esančioms pačioje funkcijoje. Pirmasis blokas analizuoja objekto judėjimo trajektorija, o antrasis – išsiskiriančius bruožus, atlieka korekcijas, jeigu jos reikalingos. Gauta informacija yra perduodama mokymosi blokui ir yra padaromos išvados, kurios bus naudojamos tolimesnei kadrų analizei, bei perduodamos potencialios objekto koordinatės [28].

Tačiau toks metodas kol kas labiau veikia tik teorijoje. Buvo palygintas *TDL* ir *KCF* (žr. 2.4.1 sk.) metodo greitis ir tikslumas. Naudojant abi technologijas buvo paleistas toks pat video įrašas ir nurodyta identiška sekama zona. Šiuo atveju buvo nurodyta stebėti krepšinio lanką (žr. 2.4.1.1 pav.). Greitis buvo matuojamas kadrais per sekundę, o tikslumas nustatomas pagal stebimo objekto koordinačių kitimą (kadangi buvo stebima statiška zona, tai kuo kitimas mažesnis, tuo algoritmas tikslesnis). Atlikus bandymus *KCF* metodas buvo pranašesnis pagal abu kriterijus. Žemiau pateikti greičio ir tikslumo grafikų palyginimai.



2.4.2.1 pav. *KCF* ir *TLD* greičių palyginimas



2.4.2.2 pav. *KCF* ir *TLD* stebimos zonos ploto kitimas

KCF vidutinis greitis 72, o *TLD* 21 kadras per sekundę (žr. 2.4.2.1 pav.). Taip pat *TLD* metodas neišlaikė stabilaus stebimos zonos ploto. Ypatingai didelis nukrypimas įvyksta, kai yra pataikomas metimas. *KCF* atveju zona buvo stebima stabiliai (žr. 2.4.2.2 pav.). Norint kreiptis į *TLD* algoritmą sukuriamas konstruktorius nurodytas 2.4.2.1 pvz..

```
Ptr<Tracker> tracker;
tracker = TrackerTLD::create();
```

2.4.2.1

Tolimesnis kodas yra identiškas kaip ir 2.4.1.1 pvz.. Norint pasinaudoti šiuo metodu taip pat reikalinga, bent Visual studio 2015 programavimo aplinka, ne senesnė nei 3.0 OpenCV versija, ir papildomų bibliotekų paketas *(angl. contrib repo).*

2.4.3 Kelių kadrų išmokstamasis algoritmas (angl. Multiple instance learning)

Sekantis metodas yra *MIL* (*angl. Multiple instance learning*) (žr. [7]). Šis metodas naudoja pavyzdinius kadrų rinkinius, o ne pavienius atvejus. Sistema išrenka kelis objekto išsiskiriančius bruožus, nustatant didžiausią jo pasirodymo tikimybę iš viso paveikslėlių rinkinio ir jame esančių išskirtinių savybių. Tada keli, labiausiai išsiskiriantis, sujungiami į vieną visumą ir sudaro vieną bruožą [29]. Taigi, pagrindinis tikslas – kaip galima greičiau atrinkti kelis bruožus, kad algoritmą būtų galima naudoti realiu laiku. Sistema, kuri naudoja reikiamus faktorius, o ne tiesiogines pikselių reikšmes veikia greičiau.

Pirmiausia sistema atrenka tinkamus ir netinkamus pavyzdžius iš kurių bus atrenkami objektų bruožai. Tarkime, kad x_0 yra tinkamas, esamame kadre *t.* Tada tinkami egzemplioriai bet kurio kadro *x*, generuojami pagal formulę $\|I_t(x) - I_t(x_0)\| < a$. Funkcija $I_t(x)$ atitinka pavyzdžio *x* vietą kadre *t*, o a yra laisvai pasirenkamas skaičius. Netinkami pavyzdžiai gaunami iš likusios dalies, kuri gaunama $\gamma < \|I_t(x) - I_t(x_0)\| < \beta$, $a < \gamma < \beta$. Be to negatyvius egzempliorius reikia apibrėžti tam tikromis spalvomis arba atspalviais, priešingu jų bus gauta labai daug, bei sugrupuoti į atskirus rinkinius [29].

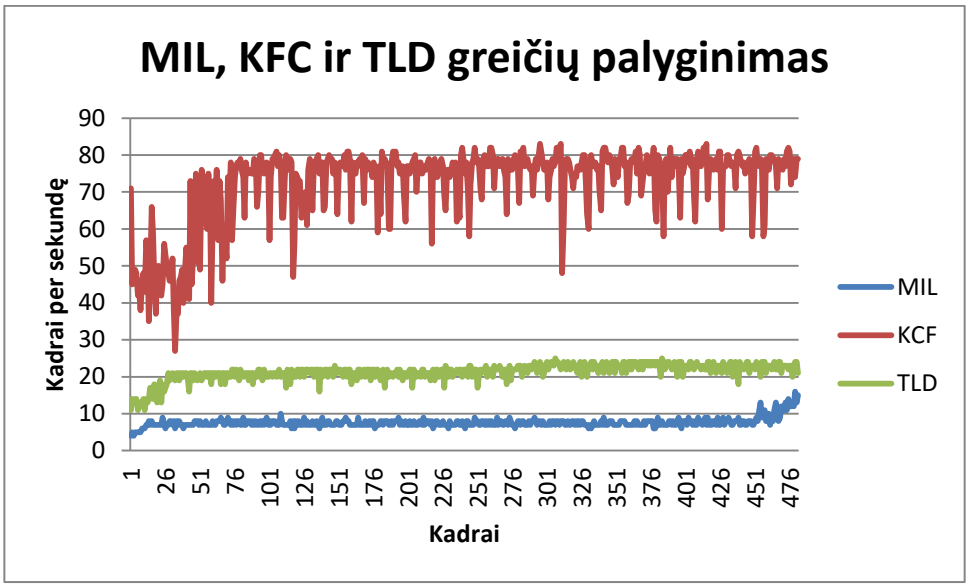
Sekančiame etape iš kiekvieno tinkamo ir netinkamo pavyzdžio yra išrenkami bruožai. Savybės yra skaičiuojamos naudojant kadro integralą. Išvaizdos modelis, gaunamas pagal formulę

$$h_k(x) = \ln\left(\frac{p_t(f_k(x)|y=1)}{p_t(f_k(x)|y=0)}\right),$$

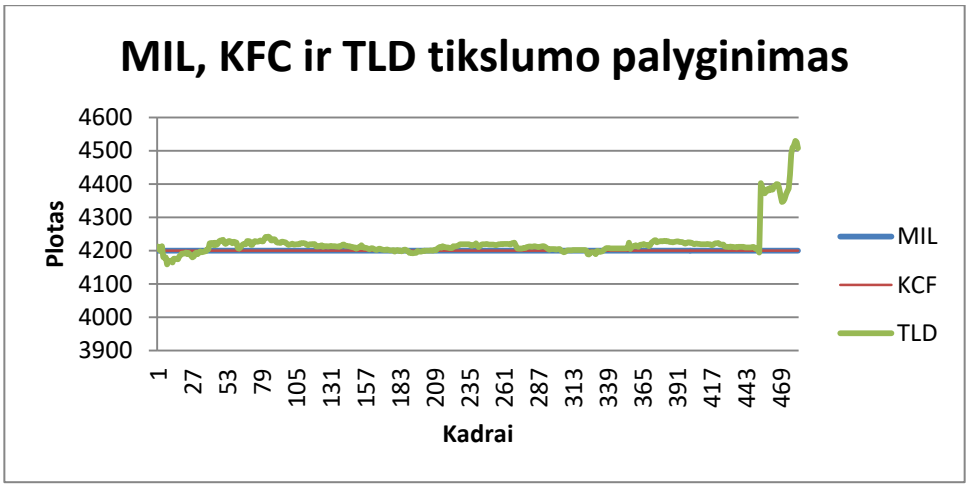
$$H_k(x) = \sum_{k=1}^{K} h_k(x).$$

Čia $H_k$yra atrinktų kelių bruožų visuma, kuri sudaro vieną bruožą, *x* kadro pavyzdys, f_k k-asis bruožas iš pavyzdžio, kai $y = 1$ pavyzdys yra tinkamas, kai $y = 0$ – netinkamas [29].

Tačiau metodo veikimas iškviečiant jį iš sukurtos *OpenCV* klasės (žr. [7]) buvo pats lėčiausias tačiau vienas iš tiksliausių. Bandymas buvo vykdytas identiškai (žr. 2.4.2 sk.). Žemiau pateikiami grafikai, kuriose yra pavaizduotas greičio ir tikslumo palyginimas su *KCF* ir *TLD* metodais.



2.4.3.1 pav. *MIL* greičio palyginimas su *KCF* ir *TLD*



2.4.3.2 pav. *MIL* stebimos zonos kitimo palyginimas su *KCF* ir *TLD*

MIL metodo vidutinis greitis buvo 8 kadrai per sekundę, o tai yra beveik tris kartus lėčiau nei *TLD* ir devynis nei *KCF* (žr. 2.4.1 ir 2.4.2 sk.). Tačiau tikslumas buvo kaip ir *KCF* atveju nekintantis visos transliacijos metu, todėl šiuo aspektu *TLD* metodas veikė prasčiau. Norint kreiptis į šį metodą yra sukuriamas toks konstruktorius,

```
Ptr<Tracker> tracker;
tracker = TrackerMIL::create();
```

2.4.3.1

Tolesnis fragmento kodas yra toks pat žr (2.4.1.1 pvz.). Reikalavimai metodo naudojimui yra tokie patys, kaip ir prieš tai buvusiems metodams (žr. [13] ir 2.4.1 ir 2.4.2 sk.).

2.4.4 Medianos kitimo algoritmas (angl. Median Flow)

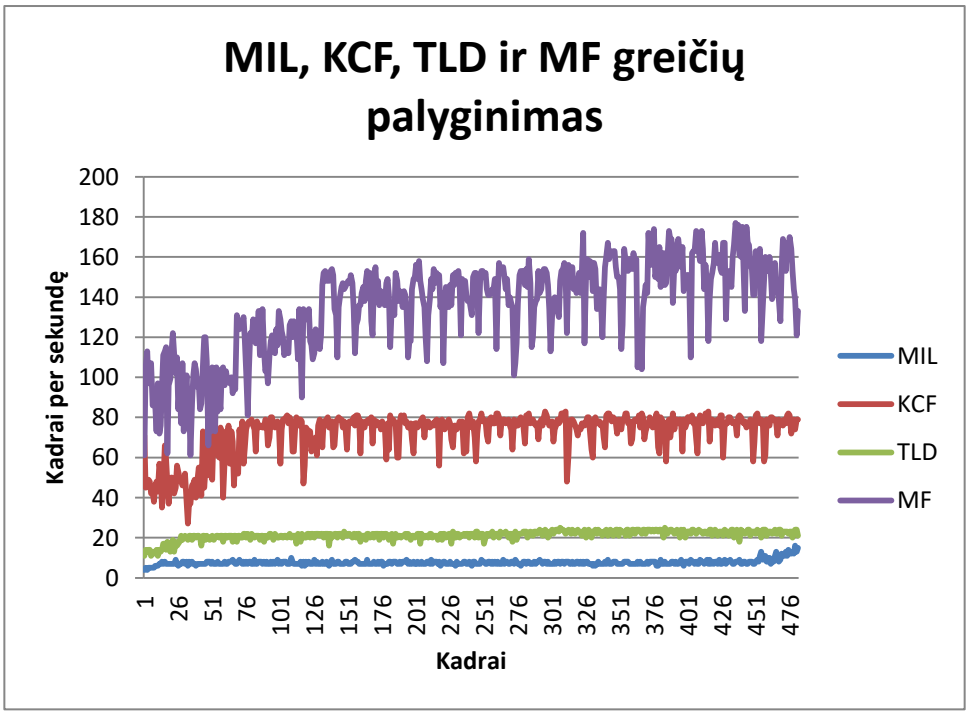
2.4.4.1 Apibrėžimas. Mediana, tai požymio reikšmė, kuri dalija variacinę eilutę, populiacija arba tikimybinį skirstinį į dvi lygias dalis. Lygiai pusę reikšmių yra mažesnės arba lygios medianai ir lygiai pusę didesnės arba lygios. Jeigu imties dydis yra lyginis skaičius, tai surandamas vidurinių reikšmių vidurkis [30].

2.4.4.2 Apibrėžimas. Populiacija**,** tai aibė objektų, turinčių tyrėją dominančių savybių [31].

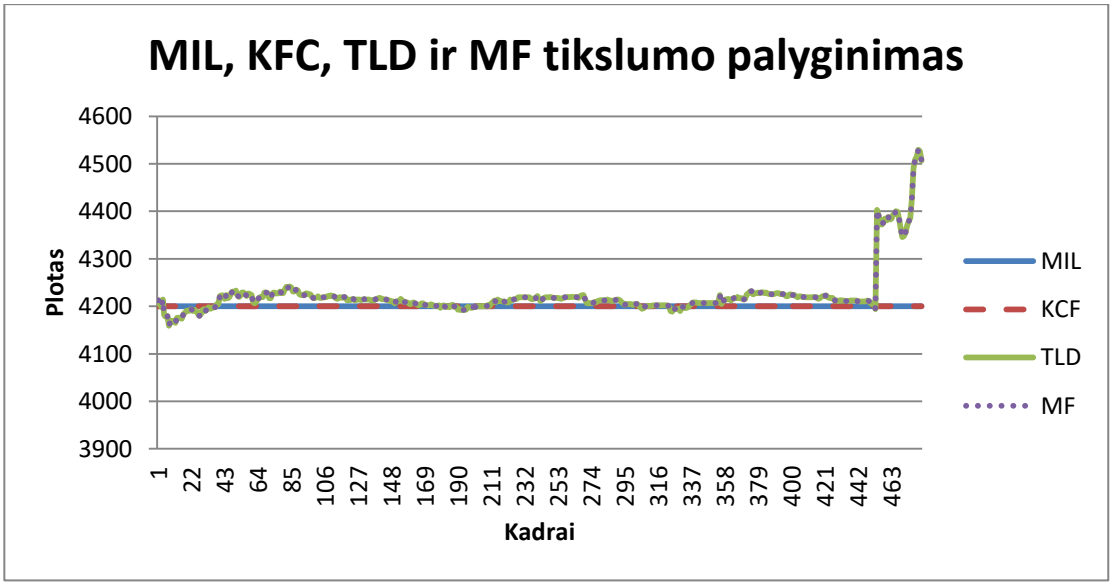
2.4.4.3 Apibrėžimas. Tikimybinis skirstinys, tai funkcija, aprašanti požymio reikšmių ir jų tikimybių tarpusavio ryšį [32].

2.4.4.4 Apibrėžimas. Variacinė eilutė tai nedidėjimo arba nemažėjimo tvarka surašytos kiekybinio požymio reikšmės [33].

Šis algoritmas (žr. [7]) yra vienas iš būdų sekti didesnius objektus esančius kadre, kadangi metodas seka ne vieną, bet krūvą taškų. Sistema priima porą kadrų, kuriose yra apibrėžiama vienoda sekamų pikselių zona. Naudojant *MF* metodą galima įvertinti jų poslinkį. Tam tikslui naudojamas vienas iš statistinių parametrų – mediana. Kiekvienai porai taškų yra skaičiuojamas atstumas tarp esamų ir prieš tai buvusio kadro taškų. Tada ribojama sritis yra gaunama, kaip tų skirtumų mediana [34]. Praktikoje algoritmas veikia efektyviai, kai objekto judėjimas yra lygus ir nuspėjamas [35]. Žemiau esančiose paveikslėliuose pateikiamas greičio ir tikslumo palyginimas su prieš tai analizuotais metodais.



2.4.4.1 pav. *MF* greičio palyginimas su *KCF, TLD, MIL*



2.4.4.2 pav. *MF* stebimos zonos kitimo palyginimas su *KCF*, *TLD* ir *MIL*

Vidutinis metodo greitis buvo 136 kadrai per sekundę. Šis metodas – greičiausias iš visų, o tikslumas identiškas *TLD* metodui. Tam kad realizuoti šį būdą reikalingi tokie patys nustatymai, kaip ir prieš tai aprašytuose variantuose (žr. 2.4.1, 2.4.2 ir 2.4.3 sk.). 2.4.4.1 pvz. nurodytas metodo konstruktorius.

```
Ptr<Tracker> tracker;
tracker = TrackerMedianFlow::create();
```

2.4.4.1

2.4.5 Besimokantis atpažinimo algoritmas (angl. Boosting)

Tai metodas, kuris yra naudojamas sekti objektus realiu laiku. Algoritmas sukurtas *AdaBoost* [36] pagrindu [37]. Pagrindinė funkcijos idėja – sudaryti tokią aplinką, kad veikimas gerėtų, vykdant jo naudojimą. Dėl to objekto pavieniai bruožai (žr. 2.4.3 sk.) yra sudedami į įvairias kombinacijas, kad padidinti atpažinimo-sekimo tikslumą. Dažniausiai yra naudojamas statistinis regresijos modeliavimas [38].

Gradiento padidinimo algoritmo pagrindinė idėja yra optimizuoti statistinio modelio regresijos funkciją, susiejant spėjimo kintamąjį X su baigtimi Y. Tai gali būti užrašyta pagal formulę

$$\hat{f}(*) = \underset{f(*)}{argmin}\{E_{X,Y}[\rho(Y, f(x)]\}$$ [34].

Čia ρ – nukrypimų funkcija, o f(*) – regresijos funkcija.

Sekantis žingsnis – minimizuoti empirinę paklaidą. Tam tikslui naudojama formulė:

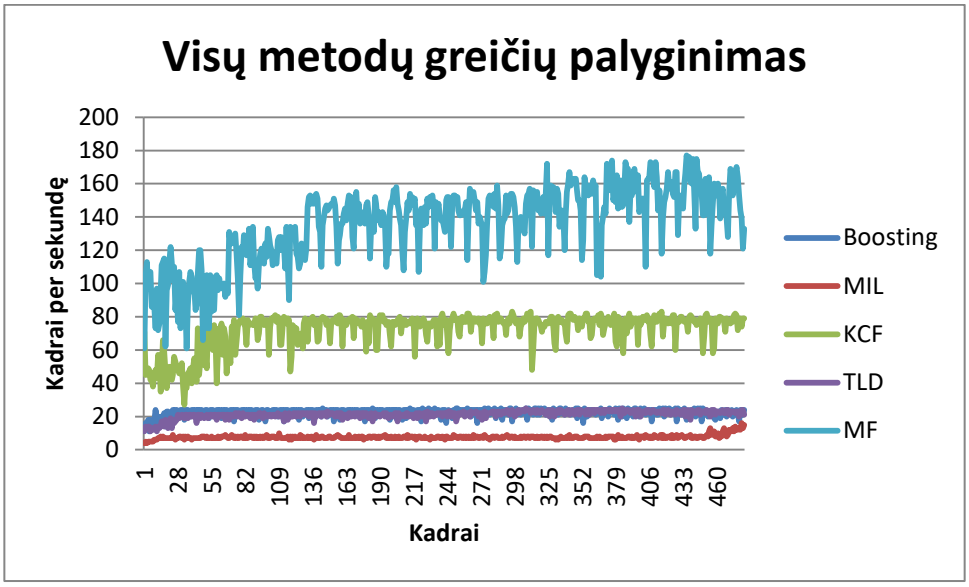
$$\hat{f}(*) = \underset{f(*)}{argmin}\left\{\frac{1}{n}\sum_{i=}^{n}\rho(y_i, f(x_i))\right\}$$ [34].

Veiksmas laužtiniuose skliaustuose yra sąlyginio vidurkio skaičiavimas. Tada reikia surasti neigiamą gradiento reikšmę, nukrypimų funkcijai, paskutinėms m-1 iteracijoms, čia m – iteracijų skaičius.

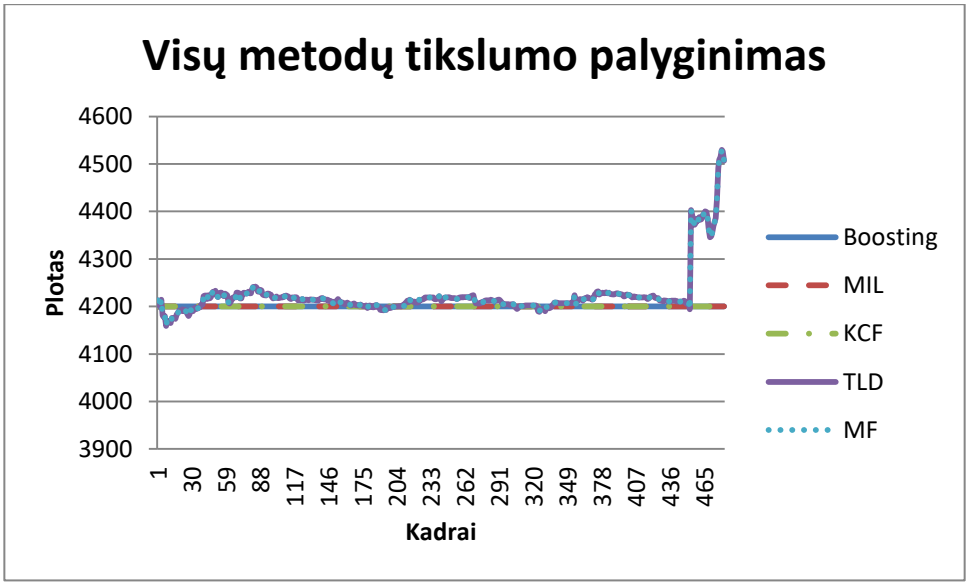
$$u^{[m]} = \left(u_i^{(m)}\right) = \left(-\frac{\partial}{\partial f}\rho(y_i, f)|f = \hat{f}^{[m-1]}(*)\right), i = 1, \dots, n \text{ [34]}.$$

Kitas būdas yra atlikti statistinį modeliavimą. Šios idėjos esmė atrinkti tuos pavyzdžius, kurie pagal tikimybių teoriją yra labiausiai tikėtini ir geriausiai apibūdina sekamą objektą, bei sudaryti statistinį modelį. Tam tikslui pirmiausia nustatomas iteracijų skaičius m, parenkami reikiami pavyzdžiai ir prognozavimo veiksnys $\hat{f}^{[0]}$ su pradine reikšme, bei įvertinama jų tikimybė β. Tada prognozuojama m+1 iteracija, nustatoma paklaidos tikimybė kiekvienam faktoriui, parenkamas komponentas su didžiausia tikimybe, m-1 iteracija prilyginama m iteracijai. Veiksmas yra kartojamas pagal pasirinktą tikslumą. Plačiau apie patį algoritmą žr. [38].

Analogiškai šis metodas buvo išbandytas iš *OpenCV* siūlomas bibliotekos. Žemiau pateikiamas palyginimas su prieš tai analizuotais metodais pagal greitį ir tikslumą.



2.4.5.1 pav. *Boosting* greičio palyginimas su *KCF, TLD, MIL, MF*



2.4.5.2 pav. Visų metodų tikslumo palyginimas

Vidutinis metodo greitis 22 kadrai per sekundę. Algoritmas greičiu lenkia *MIL* metodą, tačiau ženkliai atsilieka nuo *KFC* ir *MF*. Tikslumas nekintantis, tai rodo, kad standartinis nuokrypis iš gautų duomenų buvo lygus 0. 2.4.5.1 pvz. nurodytas algoritmo konstruktorius.

```
Ptr<Tracker> tracker;
tracker = TrackerBoosting::create();
```

2.4.5.1

Reikia pastebėti, kad nė vienas atpažinimo metodas iš *OpenCV* nėra tinkamas naudoti nei krepšinio kamuolio nei krepšininkų sekimui. Tikslumas yra nepakankamas ir reikia papildomo kodo rašymo. Vienintelė šių metodų paskirtis šiame darbe statiškų zonų stebėjimas, pavyzdžiui, pataikytų metimų atpažinimo.

2.5 Judėjimo atpažinimo technologija

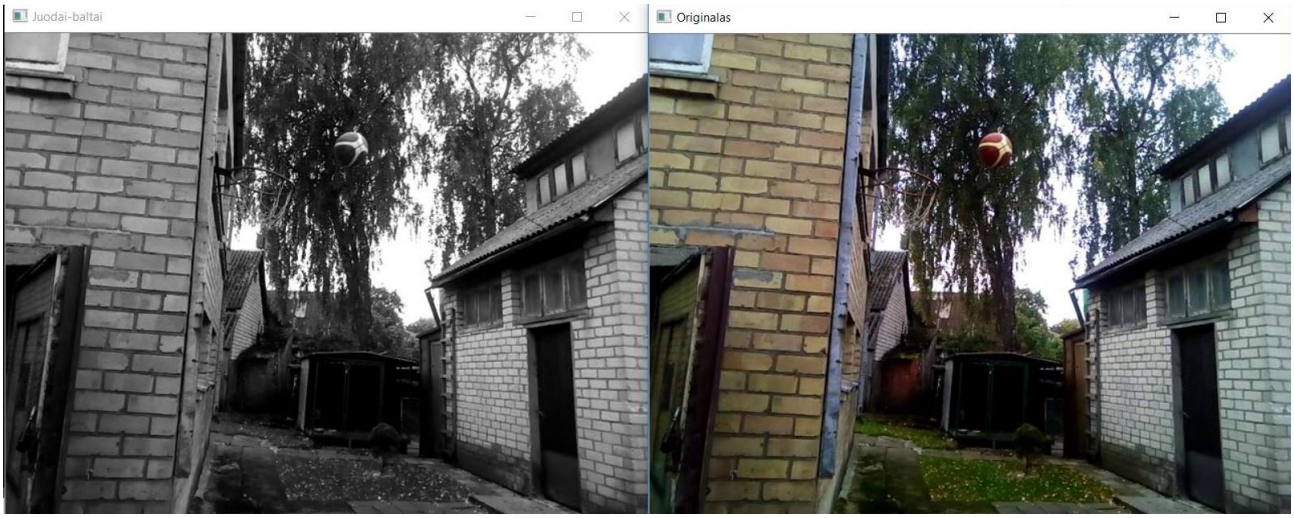
Šis metodas nėra *OpenCV* bibliotekoje, kad į jį būtų galima kreiptis tiesiogiai. Reikalingas kodo užrašymas panašiai, kaip ir spalvų atpažinimo atveju (žr. 2.3 sk.). Algoritmas yra labai efektyvus ir paprastas, kai reikia sekti vieną objektą. Programoje pakanka paleisti video transliacijos šaltinį ir daiktas arba asmuo yra sekamas be papildomų procedūrų. Tačiau kuomet reikia sekti keletą, jų išskyrimas ir atpažinimas yra sudėtingesnis, lyginant su spalvų atpažinimo technologija.

OpenCV kadrus fiksuoja *BGR* formatu. Todėl pirmasis veiksmas užfiksavus kadrą turi būti konvertavimas į juodai-baltą formatą *(OpenCV funkcija COLOR_BGR2GRAY).* Naudojant tik du spalvų kanalus, judėjimas nustatomas tiksliau. Tai galima padaryti naudojant *cvtcolor()* funkciją (žr. 2.3 sk.). Žemiau pateikiamas glaustas programos kodo fragmentas.

```
VideoCapture cap("D:\\MyOpenCv\\Images\\failas6.3gp");
Mat esamas_kadras;
bool bSuccess = cap.read(esamas_kadras);
Mat diff1; // naujas kintamasis juodai baltam kadrui
cvtColor(esamas_kadras, diff1, COLOR_BGR2GRAY);
imshow("Originalas", esamas_kadras);
imshow("Juodai-baltai", diff1);
```

2.5.1

2.5.1 pav. pateikiamas 2.5.1 pvz. rezultatas. Dešinėje yra originalus kadras, o kairėje pakeistas.



2.5.1 pav. Juodai-baltas ir originalus kadrai

Sekantis žingsnis – fiksuoti antrą kadrą, kad juos būtų galima lyginti tarpusavyje ir tai priklausys nuo procesoriaus greičio. 2.5.2 pvz. parodytas patobulintas 2.5.1 pvz. kodo fragmentas, skirtas fiksuoti vieną paskui kitą kadrus.

```
VideoCapture cap("D:\\MyOpenCv\\Images\\failas6.3gp");
Mat esamas_kadras;
while(true)
{
bool bSuccess = cap.read(esamas_kadras);
Mat diff1; // naujas kintamasis juodai baltam kadrui
cvtColor(esamas_kadras, diff1, COLOR_BGR2GRAY);
Mat sekantis_kadras;
bool bSuccess_2 = cap.read(sekantis_kadras);
Mat diff2; // naujas kintamasis juodai baltam kadrui
cvtColor(sekantis_kadras, diff2, COLOR_BGR2GRAY);
imshow("Juodai-baltai_pirmas", diff1);
imshow("Juodai-baltai_antras", diff2);
imshow("Originalas1", esamas_kadras);
imshow("Originalas2", sekantis_kadras);
}
```

2.5.2

2.5.2 pav. pavaizduoti vienas po kito einantys kadrai. Aiškiai matomas atstumas, kurį nueina objektas realiame laike, kol yra užfiksuojamas antras paveikslėlis.

2.5.2 pav. Du vienas paskui kitą kadrai

Turint du kadrus galima pradėti jų tarpusavio lyginimą. Tam tikslui *OpenCV* siūlo *absdiff()* [39] funkciją. Ši funkcija skaičiuoja:

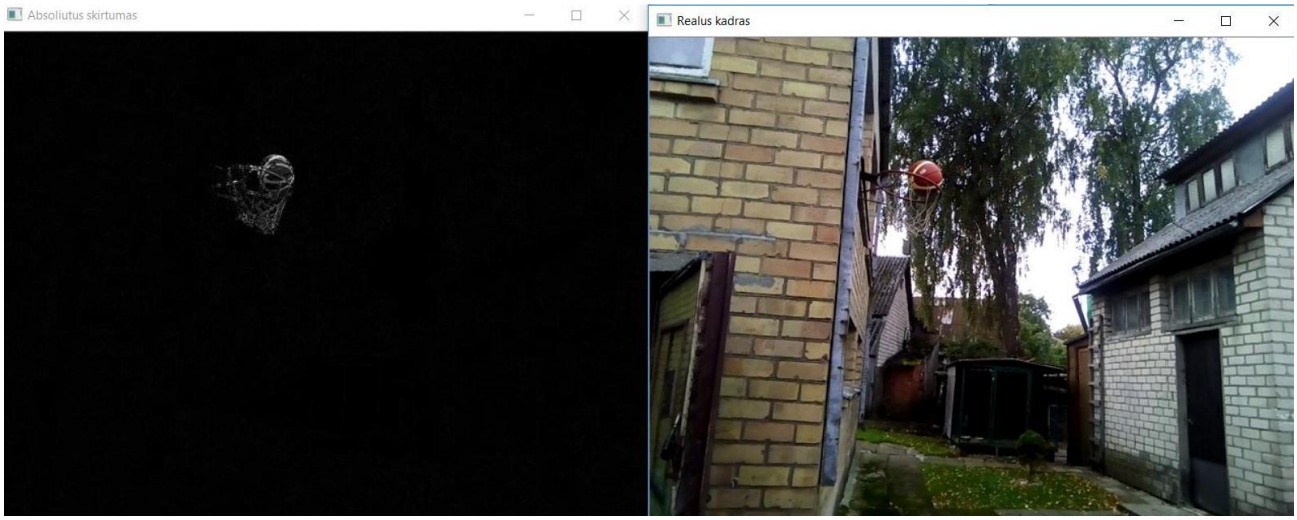
- Absoliutų skirtumą tarp dviejų masyvų, kai jų dydis ir tipas yra vienodi
- Absoliutų skirtumą tarp masyvo ir kintamojo, kai spalvų kanalų kiekis vienodas
- Absoliutų skirtumą tarp kintamojo ir masyvo, kai spalvų kanalų kiekis vienodas

Kad būtų galima naudoti šią funkciją prieš tai yra sukuriamas naujas kintamasis kuris fiksuos tik pikselių pasikeitimus ir yra nurodomas pirmo, antro ir rezultato išvedimo kintamieji. Pakoregavus 2.5.2 pvz. galima gauti metodo darbo rezultatą.

```
VideoCapture cap("D:\\MyOpenCv\\Images\\failas6.3gp");
Mat esamas_kadras;
while(true)
{
bool bSuccess = cap.read(esamas_kadras);
Mat diff1; // naujas kintamasis juodai baltam kadrui
cvtColor(esamas_kadras, diff1, COLOR_BGR2GRAY);
Mat sekantis_kadras;
bool bSuccess_2 = cap.read(sekantis_kadras);
Mat diff2; // naujas kintamasis juodai baltam kadrui
cvtColor(sekantis_kadras, diff2, COLOR_BGR2GRAY);
Mat absoliutus_skirtumas; // kintamasis fiksuoti skirtumui
absdiff(diff1, diff2, absoliutus_skirtumas);
imshow("Absoliutus skirtumas", absoliutus_skirtumas);
}
```

2.5.3

2.5.3 pav. parodytas 2.5.3 pvz. rezultatas.



2.5.3 pav. Absoliutaus kadrų skirtumo pavyzdys

Kairėje pusėje esančio lango vaizdą sudaro du vienas paskui kitą einantys kadrai. Jau galima atpažinti objektą, tačiau vis tiek reikalinga papildoma kadrų filtracija, kadangi dėl aplinkoje vyraujančio garso fono pikselių intensyvumas kinta. Taip pat metodas atpažįsta krepšio tinklelio pozicijos pokyčius, todėl kadre yra matomas ne tik krepšinio kamuolys, bet ir tinklelis. Reikalingas papildomas filtravimo algoritmas.

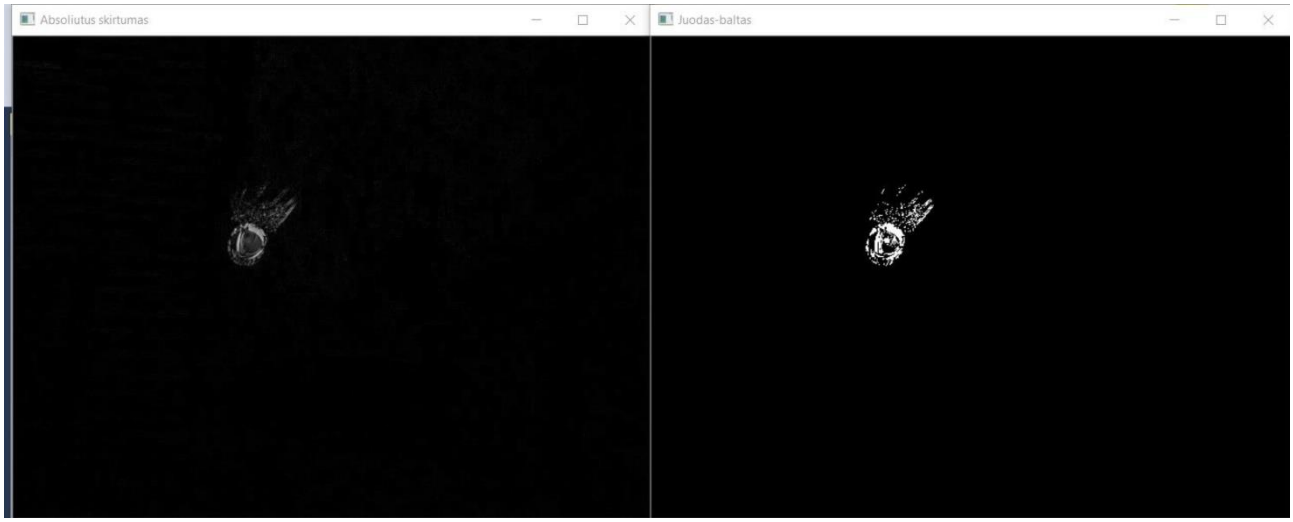
Prieš atliekant kadro filtravimą, reikia padaryti, kad paveikslėlis turėtų tik dvi grynas spalvas be jokių papildomų atspalvių. Paprastai tai būna juoda ir balta, tačiau pasirinkti galima ir kitokias (žr. 2.3 sk., 2.3.1 pav.). Tam tikslui atlikti galima naudoti *threshold()* [40] funkciją. Metodas iškarto suteikia baltą spalvą matomam objektui. Yra nurodomas kadro kurį norima modifikuoti ir į kurį bus išvestas funkcijos darbo rezultatas kintamieji, „jautrumo“ reikšmė, kuri nurodo į kokius objektus funkcija turi kreipti dėmesį, maksimali pikselių reikšmę, kuri paprastai yra 255, bei spalvų suvienodinimo tipas (plačiau žr. [40]). 2.5.4 pavyzdys papildytas su *threshold()* funkcija.

```
VideoCapture cap("D:\\MyOpenCv\\Images\\failas6.3gp");
Mat esamas_kadras;
while(true)
 {
bool bSuccess = cap.read(esamas_kadras);
Mat diff1; // naujas kintamasis juodai baltam kadrui
cvtColor(esamas_kadras, diff1, COLOR_BGR2GRAY);
Mat sekantis_kadras;
 bool bSuccess_2 = cap.read(sekantis_kadras);
Mat diff2; // naujas kintamasis juodai baltam kadrui
cvtColor(sekantis_kadras, diff2, COLOR_BGR2GRAY);
Mat absoliutus_skirtumas; // kintamasis fiksuoti skirtumui
absdiff(diff1, diff2, absoliutus_skirtumas);
```

```
Mat thresholdedimage; // grynai juodos ir grynai baltos spalvos kadro     //kintamasis
threshold(absoliutus_skirtumas, thresholdedimage, 50, 255, THRESH_BINARY);
imshow("Absoliutus skirtumas", absoliutus_skirtumas);
imshow("Juodas-baltas", thresholdedimage);
}
```

2.5.4

2.5.4 pav. parodytas 2.5.4 pvz. esančio kodo darbo rezultatas.



2.5.4 pav. Palyginimas atmetus visus galimus atspalvius

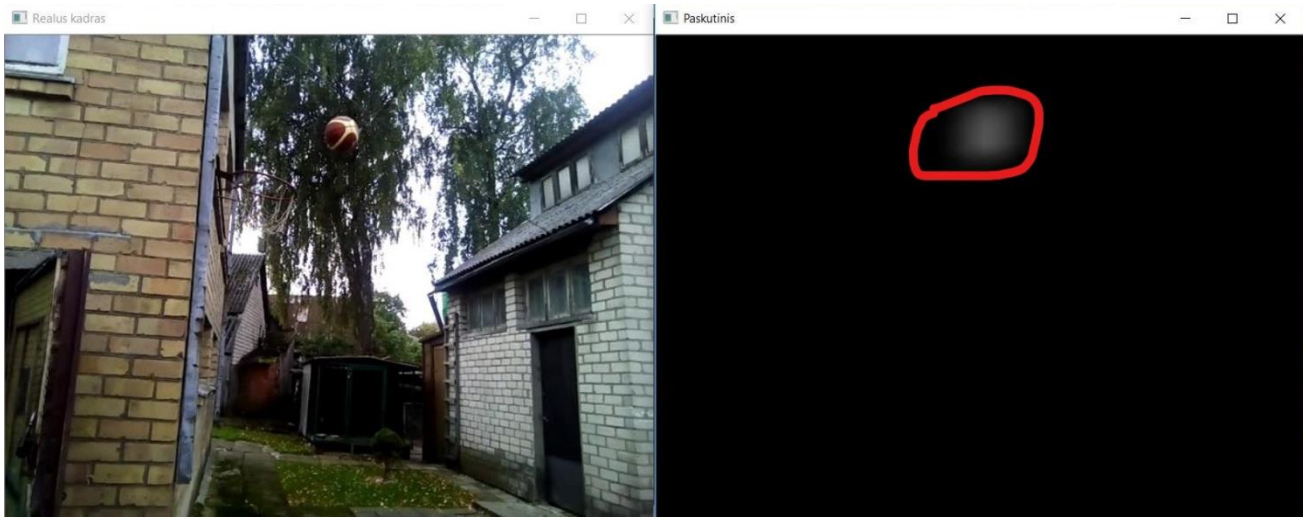
Atlikus tai, kadras yra paruoštas filtravimui. 2.5.3 ir 2.5.4 pav. matomas tinklelio judėjimas ir garso sukeliami pikselių pakitimai. Vienas iš paprasčiausių būdų atskirti kamuolio judėjimą nuo tinklelio *blur()* [41] metodo naudojimas, tačiau tai ne vienintelis būdas (žr. [42] alternatyvas). Funkcijai nurodomi šaltinio ir išvedimo kintamieji ir pageidaujamas dydis. 2.5.5 pvz., *blur()* funkcijos naudojimas.

```
VideoCapture cap("D:\\MyOpenCv\\Images\\failas6.3gp");
Mat esamas_kadras;
while(true)
{
bool bSuccess = cap.read(esamas_kadras);
Mat diff1; // naujas kintamasis juodai baltam kadrui
cvtColor(esamas_kadras, diff1, COLOR_BGR2GRAY);
Mat sekantis_kadras;
bool bSuccess_2 = cap.read(sekantis_kadras);
Mat diff2; // naujas kintamasis juodai baltam kadrui
cvtColor(sekantis_kadras, diff2, COLOR_BGR2GRAY);
Mat absoliutus_skirtumas; // kintamasis fiksuoti skirtumui
absdiff(diff1, diff2, absoliutus_skirtumas);
```

```
Mat thresholdedimage; // grynai juodos ir grynai baltos spalvos kadro     //kintamasis
threshold(absoliutus_skirtumas, thresholdedimage, 50, 255, THRESH_BINARY);
blur(thresholdedimage, thresholdedimage, Size(50, 50)); // blur funkcija
imshow("Absoliutus skirtumas", absoliutus_skirtumas);
imshow("Juodas-baltas", thresholdedimage);
imshow("paskutinis", thresholdedimage);
}
```

2.5.5

Šį kartą įvedimo ir išvedimo kintamasis tas pats. 2.5.5 pav. parodytas *blur()* funkcijos darbo rezultatas.



2.5.5 pav. Rezultatas panaudojus *blur()* funkciją.

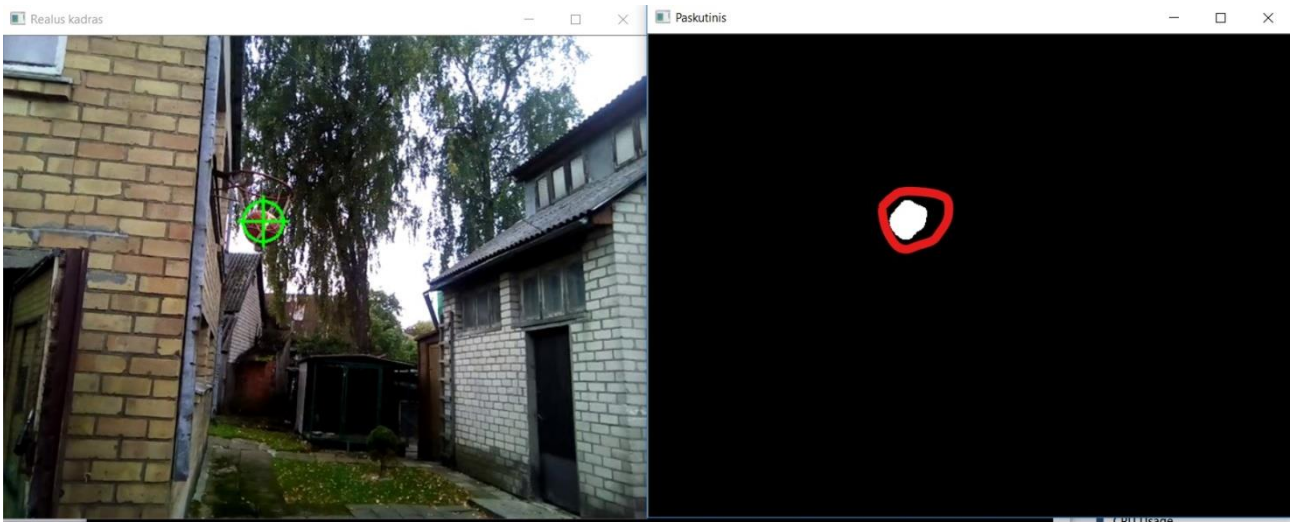
Kadangi buvo pasirinktas gana didelis pikselių intensyvumas (50x50, žr. 3.9.5 pvz.), tai kamuolys nėra matomas pakankamai aiškiai, todėl dar kartą reikia panaudoti *threshold()* funkciją (žr. 2.5.4 pvz. ir 2.5.4 pav.).

```
VideoCapture cap("D:\\MyOpenCv\\Images\\failas6.3gp");
Mat esamas_kadras;
while(true)
{
bool bSuccess = cap.read(esamas_kadras);
Mat diff1; // naujas kintamasis juodai baltam kadrui
cvtColor(esamas_kadras, diff1, COLOR_BGR2GRAY);
Mat sekantis_kadras;
bool bSuccess_2 = cap.read(sekantis_kadras);
Mat diff2; // naujas kintamasis juodai baltam kadrui
cvtColor(sekantis_kadras, diff2, COLOR_BGR2GRAY);
Mat absoliutus_skirtumas; // kintamasis fiksuoti skirtumui
```

```
absdiff(diff1, diff2, absoliutus_skirtumas);
Mat thresholdedimage; // grynai juodos ir grynai baltos spalvos kadro     //kintamasis
threshold(absoliutus_skirtumas, thresholdedimage, 50, 255, THRESH_BINARY);
blur(thresholdedimage, thresholdedimage, Size(50, 50)); // blur funkcija
imshow("Absoliutus skirtumas", absoliutus_skirtumas);
imshow("Juodas-baltas", thresholdedimage);
threshold(thresholdedimage, thresholdedimage, 50, 255, THRESH_BINARY);
imshow("paskutinis", thresholdedimage);
}
```

2.5.6

Atlikus tai gaunamas galutinis kadro vaizdas (žr. 2.5.6 pav.). Tokiu būdu galima atskirti kamuolio judesius nuo krepšio tinklelio judesių, be to yra pašalinami garso sukurti netikslumai. Tačiau norint atskirti judantį žmogų, nuo judančio kamuolio to nepakanka.



2.5.6 pav. Galutinis rezultatas

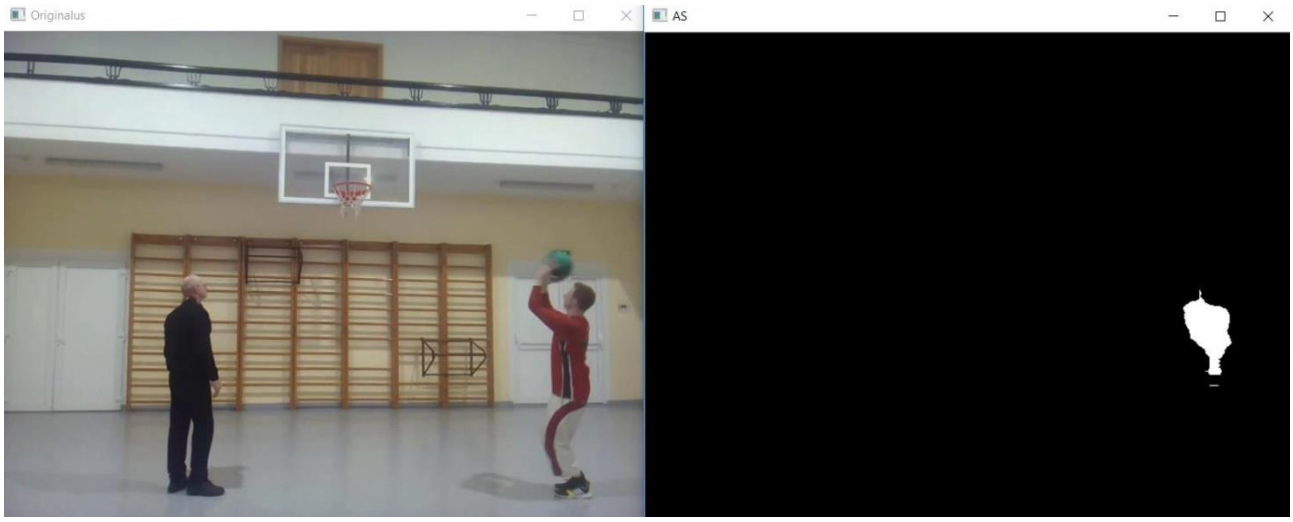
2.6 Objekto atpažinimas, naudojant judėjimo ir spalvų atpažinimo technologiją

Sekantis išbandytas objektų atpažinimo variantas, kuomet yra apjungiamos judėjimo ir spalvų atpažinimo technologijos (žr. 2.3 ir 2.5 sk.). Buvo išbandyti du atvejai. Pirmiausia fiksuojamas judėjimas, o po to pridedamas HSV spalvų atpažinimas, taip patikslinant sekamą objektą. Antras atvejis priešingas, pirmiausia atpažįstamas objektas pagal HSV spalvų paletę, o tada jo atpažinimas patikslinamas naudojant judėjimo sekimo technologija. Gali pasirodyti, kad nesvarbu, kuris būdas pasirenkamas, rezultatai vis tiek bus tokie patys. Tačiau atlikus bandymus paaiškėjo, kad taip nėra. Tam kad butų sudarytos vienodos sąlygos, visi triukšmo trikdžių pašalinimo algoritmai buvo naudoti tokie patys.

2.6.1 Judėjimo ir spalvų atpažinimo technologija

Pirmiausia yra fiksuojamas pirmas kadras, bei iš karto konvertuojamas į HSV paletę. Analogiškai gaunamas ir antras paveikslėlis. Tada surandamas absoliutus skirtumas naudojant *absdiff()* metodą (žr. [39] ir 3.9 sk). Šį kartą yra praleidžiamas kadrų konvertavimas į juodai baltą

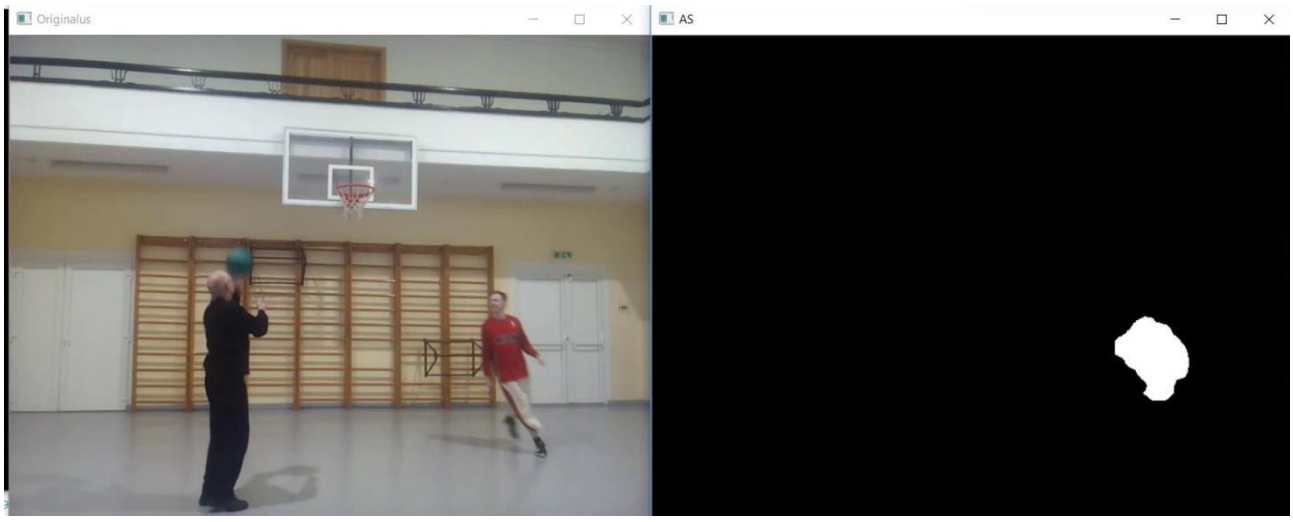
kadrą (žr. 2.5.1 pav.). Taip yra todėl, kad priešingu atveju HSV spalvų reikšmės nepadarys jokios įtakos objekto išskyrimui iš aplinkos. Atlikus tai, sekančiame veiksme reikia perduoti pasirinktas spalvų reikšmes. Tai atliekama naudojant *inrange()* funkciją. Metodas *blur()* pašalina triukšmo ir aplinkos trikdžius. Gautas filtravimo rezultatas parodytas 2.6.1.1 pav.



2.6.1.1 pav. Objekto išskyrimas naudojant judėjimo ir spalvų atpažinimo technologijas

2.6.2 Spalvų ir judėjimo atpažinimo technologija

Naudojant šį variantą yra atliekamas atvirkštinis procesas (žr. 2.6.1 sk.). Skirtumas tik tas, kad HSV reikšmės yra perduodamos abiems kadrams ir yra surandamas absoliutus jų skirtumas. 2.6.2.1 pav. yra parodytas rezultatas.



Pav. 2.6.2.1 Objekto išskyrimas naudojant spalvų atpažinimo ir judėjimo technologijas

2.6.3 Rezultatų palyginimas, naudojant spalvų atpažinimo technologiją, kartu su judėjimo atpažinimu ir atvirkščiai

Tikslumas matuojamas kiekiu kadrų, kuriuose buvo pastebėtas sekamas objektas, o sistemos veikimo greitis lyginamas su realiu laiku (žr. 1.2 sk.). Abiejų rezultatų gavimui buvo naudojamas tie patys video įrašai.

2.6.1 sk. aprašytas atvejis 53 sekundžių video įraše kamuolį užfiksavo 244 kartus. Lyginant su realiu laiku sistema veikė 52 % greičiau. 2.6.3.1 lent. pateikiami greičio rezultatai, kiekvieno iš penkių video įrašų atveju.

2.6.3.1 lentelė. Pavyzdinių video įrašų trukmė

Įrašo trukmė	53s	132s	41s	66s	56s
Įrašo trukmė sistemoje	27s	64s	20s	32s	26s

2.6.2 sk. aprašytas atvejis 53 sekundžių video įraše kamuolį užfiksavo 429 kartus, t.y. 56% daugiau, nei pirmuoju atveju (žr. 2.6.1 sk.). Tačiau lyginant su realiu laiku sistema veikė 51% greičiau (žr. 2.6.3.2 lent.).

2.6.3.2 lentelė. Pavyzdinių video įrašų trukmė sistemoje

Įrašo trukmė	53s	132s	41s	66s	56s
Įrašo trukmė sistemoje	28s	68s	21s	34s	27s

2.7 Kelių objektų sekimas, kurie turi vienodų bruožų

2.3 sk. aprašyta spalvų atpažinimo technologija veikia, jeigu objektai turi ryškius kontrastus. Pavyzdžiui, kai kadre yra matomi du asmenys, kurių apranga yra vienoda, tai atpažinimo metodas mėtysis tarp abiejų žmonių ir neseks nė vieno. Taip pat šis algoritmas yra nepatogus, kai norima sekti keletą objektų. Tokiu atveju kiekvienam iš jų reikalingas atskiras filtravimas. Tačiau remiantis šia technologija, galima sukurti tobulesnį variantą, kuris gali sekti didesnį kiekį daiktų arba asmenų, kurie pasižymi bendrais bruožais, pavyzdžiui, spalva. Koks kiekis asmenų gali būti sekamas priklauso nuo kameros atstumo ir filtravimo algoritmo reikšmių.

Pirmiausia reikia nurodyti ploto intervalą, pagal kurį sistema turės atskirą objektą (pasirenkama didžiausia ir mažiausia reikšmė). Dydžių pasirinkimas priklauso nuo atstumo tarp objekto ir kameros, bei filtravimo algoritmo reikšmių. Kuo didesnis plotas prilyginamas garso trikdžiams, tuo mažesnis skirtumas tarp intervalo dydžių reikalingas ir atvirkščiai. Taip pat reikia nurodyti maksimalų galimą objektų kiekį. Tai leis sistemai veikti tiksliau, bei pranešti kada bus reikalingas tikslesnis filtravimo algoritmas.

Sekantis svarbus etapas, tai sąrašo sudarymas, kuriame bus saugomos užfiksuotų objektų koordinatės. Tam tikslui galima sukurti papildomą klasę su keturiais metodais (tuo atveju, jeigu yra naudojama C++ programavimo kalba). Du metodai skirti užfiksuoti X ir Y koordinates, kiti jas naudoti. Tada galima sukurti vektorių [43], kuris pasižymi klasėje nurodytomis savybėmis (žr. 2.7.1 pvz.).

```
vector <Krepsininkai> raudoni;
```

2.7.1

Toliau reikia suskaičiuoti kadre matomus objektus ir palyginti ar nėra viršijamas matomų asmenų kiekis. Galima naudoti *findCountours()* [44] funkciją. Metodui yra nurodomas šaltinio kadras, aptikti kontūrai, kurie yra saugomi kaip seka taškų, sąrašo tipo kintamasis, kuris atrenka, kurie objekto kontūrai reikalingi jo atskyrimui, kontūrų ieškojimo reikšmė (žr. [45]), bei kontūrų aproksimacijos dydis (žr. [46]) ir kontūrų subalansavimo reikšmė, kuri pagal nutylėjimą gali būti ir nenurodoma. Šios funkcijos naudojimas parodytas. (2.7.2 pvz.)

```
Mat saltinio_kadras;
vector< vector<Point> > konturai;
vector<Vec4i> konturu_hierarchija;
findContours(saltinio_kadras, konturai, konturu_hierarchija, CV_RETR_CCOMP,
CV_CHAIN_APPROX_SIMPLE);
```

2.7.2

Tam, kad suskaičiuoti kiek yra matoma objektų kadre, pakanka surasti vektoriaus *konturu_hierarchija* dydį. Po to yra taikoma *Moments()* funkcija kontūrų sąrašui, kad gauti reikiamas daiktų arba asmenų koordinates. Tada tikrinama ar matoma erdvė yra didesnė už nurodyta plotą (žr. 2.3 sk. ir 2.3.4 pvz.). Sukuriamas klasės *Krepsininkai* sąrašas, kuriam bus priskiriamas kiekvienas gautas kontūras iš *konturai* sąrašo. Kad gauti objektų koordinates panaudojami metodai sukurti *Krepsininkai* klasėje. Metodas *push_back()* [47] prideda gautą kontūrą į nurodytą sąrašą (žr 2.7.1 ir 2.7.3 pvz.).

```
if (konturu_hierarchija.size() < Maksimalus_objektu_kiekis)

{

    for (int i = 0; i >= 0; i = konturu_hierarchija[i][0])
    {
                Moments momentai = moments((cv::Mat)konturai[i]);
                double area = momentai.m00;
                if (area > Minimalus_objekto_dydis)
              {

                 Krepsininkai raudonas;
                  raudonas.setXPos(momentai.m10 / area);
                  raudonas.setYPos(momentai.m01 / area);
                raudoni.push_back(raudonas);

               }

    }

}
```

2.7.3

3. MPAAS PROJEKTINĖ DALIS

3.1 Sistemos paskirtis

Treniruočių procesas krepšinyje, kaip ir kiekvienoje sporto šakoje, yra svarbiausias faktorius siekiant aukščiausių rezultatų, todėl tai yra nuolat tobulinama, pasitelkiant informacines technologijas ir kitus būdus. Yra sukurta daug įvairių programų, kurios analizuoja sportinę veiklą, tačiau tokių, kurios analizuotų sportininkų savybes realiu laiku nėra daug. Sukurtas produktas seka žaidėjus, skaičiuoja atliktus ir pataikytus metimus, bei atlieka statistinį vertinimą. Pavyzdžiui, dabar komandos organizuoja specialias stovyklas, kad galėtų įvertinti žaidėjo pataikymo galimybes. Taip pat reikalingas atskiras pratimas, bei mažiausiai vienas treneris. Sistema sutaupys daug laiko, kurį bus galima skirti didesniam kiekiui draugiškų varžybų arba tiesiog paprastoms treniruotėms rungtyniaujant penki prieš penkis. Taip pat tokia programinė įranga padės krepšininkų skautinge, sumažins komandų išlaidas skiriamas skauto kelionėms, pragyvenimo išlaidoms, kadangi žaidėjų treniruotes būtų galima sekti nuotoliniu būdu. Be to kiekvienas žaidėjas galės atlikti metimų treniruočių pratimus netrukdomai.

3.2 MPAAS uždaviniai

Kuriant tokią sistemą labai svarbu atpažinti sportininkus ir žaidimo įrankį. Kadangi programa skirta treniruotėms, todėl kamuolio sekimas yra lengvesnis. Taip yra todėl, kad realių varžybų metu sviedinys dažnai būna uždengiamas pačių žaidėjų ir judėjimo trajektorija yra sunkiai nuspėjama. Tačiau šiuo atveju yra skaičiuojami treniruočių pataikyti ir mesti metimai, todėl galima nuspėti kamuolio poziciją, kai yra atliekamas metimas ir pataikymas.

Tam kad sekti žmones panaši metodika gali būti taikoma. Dėl to, kad asmenys paprastai yra dideli objektai, jų judėjimo greitis lyginant su kamuoliu yra mažesnis, bei pozicijos kitimo trajektorija yra labiau nuspėjama, sekimo algoritmas yra sudaromas paprasčiau. Vienas iš trūkumų yra kai norima surasti būdą, kad butų galima sekti daugiau negu viena asmenį, reikalingas vienodų bruožų atpažinimas, ko sekant kamuolį dažniausiai nereikia. Taip pat reikia susieti asmenų poziciją su kamuolio pozicija, tam kad nebūtų atlikti nereikalingi metimų priskaičiavimai. Pavyzdžiui, kai vienas žmogus atlieka metimą, tačiau yra kertama ir kito krepšininko metimo zona, metimas yra priskaičiuojamas abiems. Panašios klaidos gali nutikti kai atliekamas perdavimas.

Norint skaičiuoti atliktus ir pataikytus metimus vienas iš būdų yra stebėti atitinkamas zonas. Tuomet kiekvienas metimas yra užskaitomas, kai kamuolys kerta nurodytą sritį, kuri dažniausiai yra apibrėžiama keturiais pikselių taškais. Analogiškai atliekamas ir objektų sekimas, naudojant daugiau nei vieną kamerą. Pagal poreikį yra pasirenkama ar metimas užskaitomas, kai visos kameros pastebi kamuolį arba viena iš jų. Taip pat stebimas plotas nustatomas vienodos arba skirtingos visose kamerose.

3.3 Esminiai reikalavimai sistemai

1. Transliacija paleidžiama iš video failo arba realiu laiku
2. Sistemos video transliacijos kokybė 640x480
3. Transliaciją vykdoma per daugiau nei vieną kamerą
4. Sistemos greitis turi būti lygus realiam laikui

3.4 Gautos sistemos parametrai

1. Kodo eilučių kiekis: 8126
2. Atpažįsta daugiau kaip 90% atliktų ir pataikytų metimų
3. Sistema veikia tolygiai realiu laiku naudojant kompiuterį, kurio resursai: procesorius – i3 2GHZ, 8 GB RAM.

3.5 Sistemos realizavimo technologijos

1. Visual studio community 2017 (C++, C#, GoogleTest, Cunit)
2. OpenCV 3.3.0
3. IPWebcam, pagalbinė programinė įranga, skirta telefono kamerų video medžiagą perkelti į kompiuterį
4. Bluetooth, wi-fi
5. Dvi kameros
6. Kompiuteris: procesorius i3 2GHZ, 8 GB RAM
7. Telefonas, wi-fi tinklo sukūrimui

3.6 Pasirinkti sprendimai

1. Spalvų atpažinimo technologija
2. Judėjimo atpažinimo technologija
3. Spalvų konvertavimas į HSV paletę, naudojant *cvtcolor()* funkciją
4. Centrinių momentų metodas, objekto koordinatėms nustatyti
5. Mesto ir pataikyto metimo zonos apibrėžiamos pikselių koordinatėmis

3.7 Sistemos funkcijos

1. Atpažįsta asmenis ir krepšinio kamuolį
2. Suskaičiuoja išmestus ir pataikytus metimus, kiekvienam asmeniui atskirai
3. Galima nurodyti kiek kiekvienas krepšininkas turi atlikti metimų per tam tikrą laiką. Sistema seka ar sportininkas įvykdė tikslus po treniruotės.
4. Išvedamas pataikymo procentas, atliktų metimų kiekis, pataikytų metimų kiekis, vidutinis laikas tarp pataikytų ir mestų metimų atskirai.
5. Transliacija paleidžiama per dvi kameras

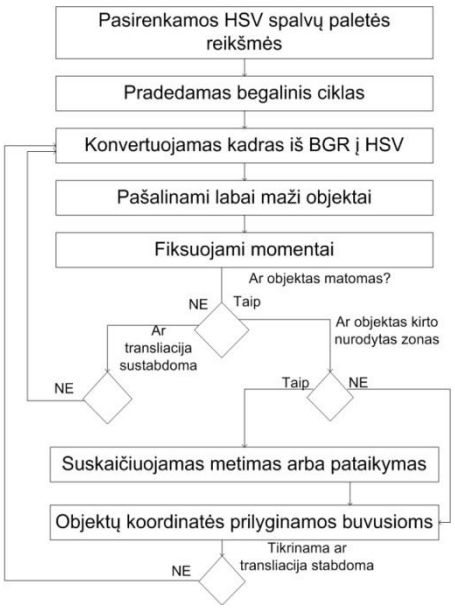
3.8 Sistemos testavimas

1. **Vienetų testavimas.**

- Paleidžiamas pasirinktas video failas, ir visų objektų koordinatės įrašomos į failą. Toliau tas pats įrašas perduodamas testavimo įrankiui, o šis skaito nurodytus failus ir lygina po vieną koordinatę.
- Tikrinamas visų sistemos funkcijų veikimas. Procedūros atliekamos standartiškai, kreipiamasi į metodus, gaunamos rezultatų reikšmės, bei lyginamos su tikėtinomis.
- Tikrinamas kiekvienas sistemos kelias, pasiekus kiekvieną sistemos atšaką įrašomas pranešimas į failą. Tada testavimo įrankis skaito failą ir tikrina ar visi pranešimai įvesti.
- Tikrinamas kiekvienas sistemos mygtuko paspaudimas. Jeigu mygtukas veikia tinkamai yra įrašomas pranešimas į tekstinį failą, tada testavimo įrankis jį skaito ir stebi ar yra visi pranešimai.

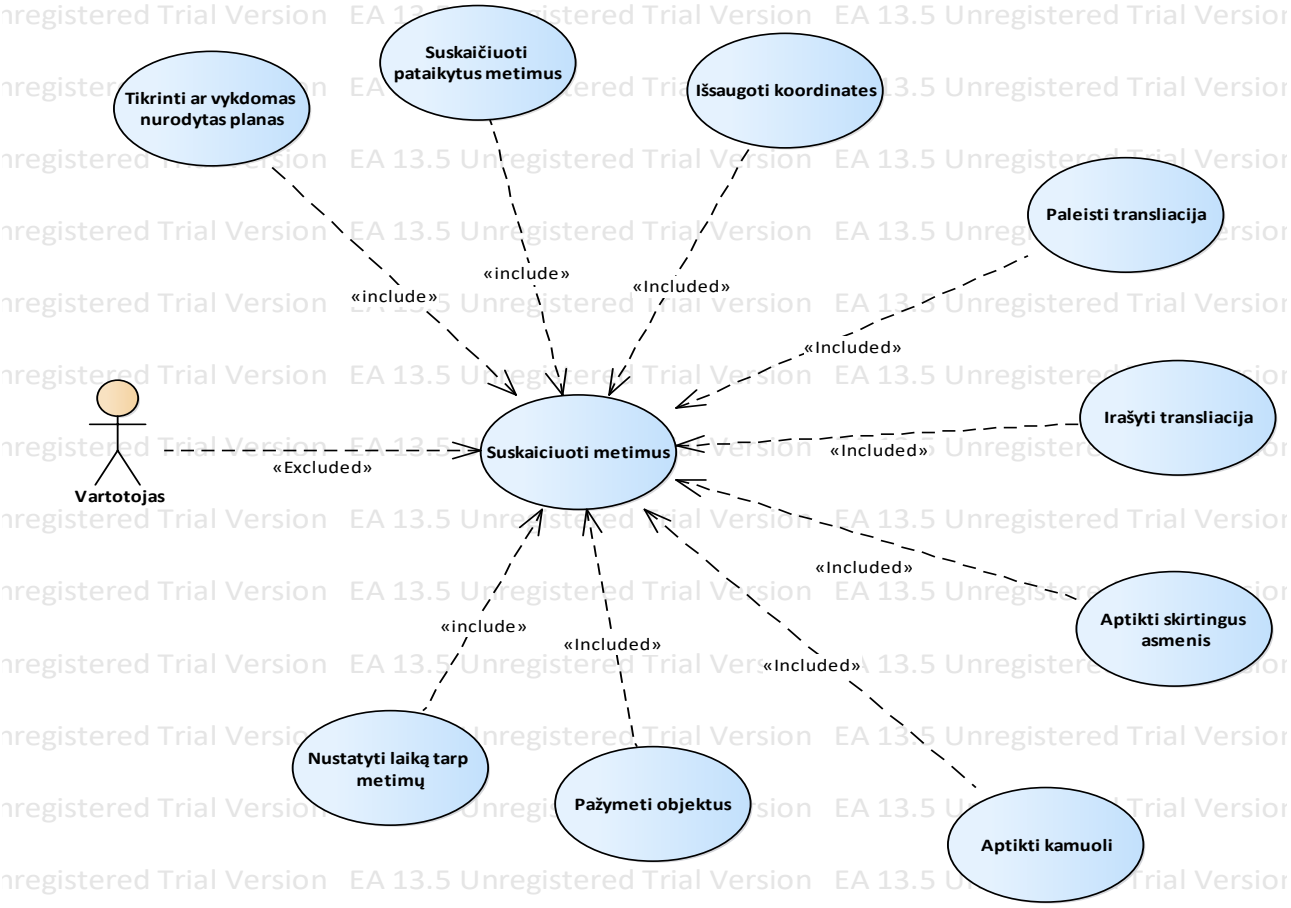
2. **Priėmimo testavimas.**
    - Vykdomas juodosios dėžės principu
    - Tikrinama ar pranešimuose nėra klaidų
    - Imituojami vartotojo veiksmai. Tam buvo sukurtas automatizuotas įrankis, naudojant *GoogleTest* testavimo įrankį.
3. **Aukšto lygio testavimas.**
    - Tikrinamas sistemos greitis
    - Paleidžiama sistema ir skaičiuojamas procesoriaus veikimo laikas
    - Gauti rezultatai lyginami su realiu laiku

3.9 Glaustas sistemos algoritmas



3.9.1 pav. Glaustas sistemos veikimo algoritmas

3.10 Sistemos panaudos atvejai



3.10.1 pav. Panaudos atvejų diagrama

3.10.1 lentelė. PA 1

1. PANAUDOJIMO ATVEJIS: Suskaičiuoti metimus
Vartotojas/Aktorius: Paprastas vartotojas
Aprašas: Skaičiuojami metimai
Prieš sąlyga: Asmuo analizuos treniruotę
Sužadinimo sąlyga: Paleidžiama programinė įranga
Po-sąlyga: Programinė įranga pradeda darbą

3.10.2 lentelė. PA 2

2. PANAUDOJIMO ATVEJIS: Išsaugoti koordinates
Vartotojas/Aktorius: Paprastas vartotojas
Aprašas: Išsaugoti koordinates
Prieš sąlyga: Atsiranda poreikis nustatyti pataikyto metimo apibrėžimą
Sužadinimo sąlyga: Paspaudžiamas pelės kairysis mygtukas
Po-sąlyga: Koordinatės įrašomos į failą

3.10.3 lentelė. PA 3

3. PANAUDOJIMO ATVEJIS: Paleisti transliaciją
Vartotojas/Aktorius: Paprastas vartotojas
Aprašas: Pradėti transliaciją
Prieš sąlyga: Paleidžiama programinė įranga
Sužadinimo sąlyga: Pasirenkamas transliacijos būdas
Po-sąlyga: Pradedamas programos veikimas

3.10.4 lentelė. PA 4

4. PANAUDOJIMO ATVEJIS: Įrašyti transliaciją
Vartotojas/Aktorius: Paprastas vartotojas
Aprašas: Pradėti vaizdo įrašinėjimą
Prieš sąlyga: Atsiranda poreikis įrašyti vaizdą
Sužadinimo sąlyga: Pasirenkama vaizdo įrašymo pradžia
Po-sąlyga: Pradedamas įrašyti vaizdas

3.10.5 lentelė. PA 5

5. PANAUDOJIMO ATVEJIS: Aptikti skirtingus asmenis
Vartotojas/Aktorius: Paprastas vartotojas
Aprašas: Žaidėjų atpažinimas
Prieš sąlyga: Programinė įranga pradėjo veikimą
Sužadinimo sąlyga: Nustatomos spalvų reikšmės
Po-sąlyga: Objektas pažymimas transliacijoje

3.10.6 lentelė. PA 6

6. PANAUDOJIMO ATVEJIS: Aptikti kamuolį
Vartotojas/Aktorius: Paprastas vartotojas
Aprašas: Atpažinti ir sekti kamuolį
Prieš sąlyga: Programinė įranga pradėjo veikimą
Sužadinimo sąlyga: Nustatomos spalvų reikšmės
Po-sąlyga: Objektas pažymimas transliacijoje

3.10.7 lentelė. PA 7

7. PANAUDOJIMO ATVEJIS: Pažymėti objektus
Vartotojas/Aktorius: -
Aprašas: Pažymėti atpažįstamus objektus
Prieš sąlyga: Programinė įranga veikia
Sužadinimo sąlyga: Nustatomos spalvų reikšmės
Po-sąlyga: Pažymimas objektas

3.10.8 lentelė. PA 8

8. PANAUDOJIMO ATVEJIS: Tikrinti ar vykdomas nurodytas planas
Vartotojas/Aktorius: Paprastas Vartotojas
Aprašas: Stebėti ar reikiamas kiekis metimų atliktas per nurodytą laiką
Prieš sąlyga: Programinė įranga veikia
Sužadinimo sąlyga: Nurodytas kiekis metimų atliktas anksčiau arba vėliau
Po-sąlyga: Parodomas pranešimas apie rezultatą

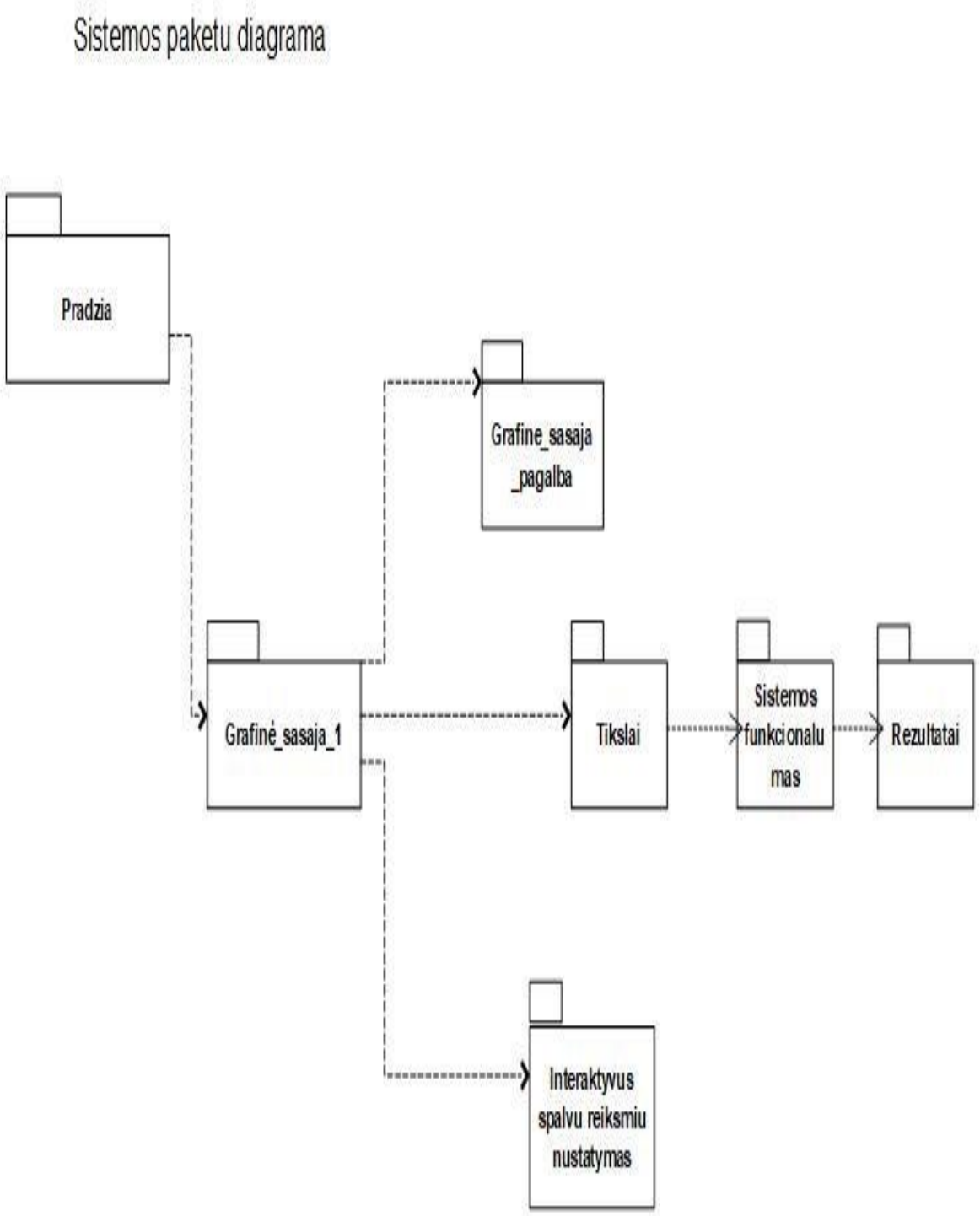
3.10.9 lentelė. PA 9

9. PANAUDOJIMO ATVEJIS: Suskaičiuoti pataikytus metimus
Vartotojas/Aktorius: Paprastas Vartotojas
Aprašas: Suskaičiuoti pataikytus metimus
Prieš sąlyga: Atliekamas metimas
Sužadinimo sąlyga: Kamuolys kerta nurodytas koordinates
Po-sąlyga: Parodomas pranešimas apie rezultatą

3.10.10 lentelė. PA 10

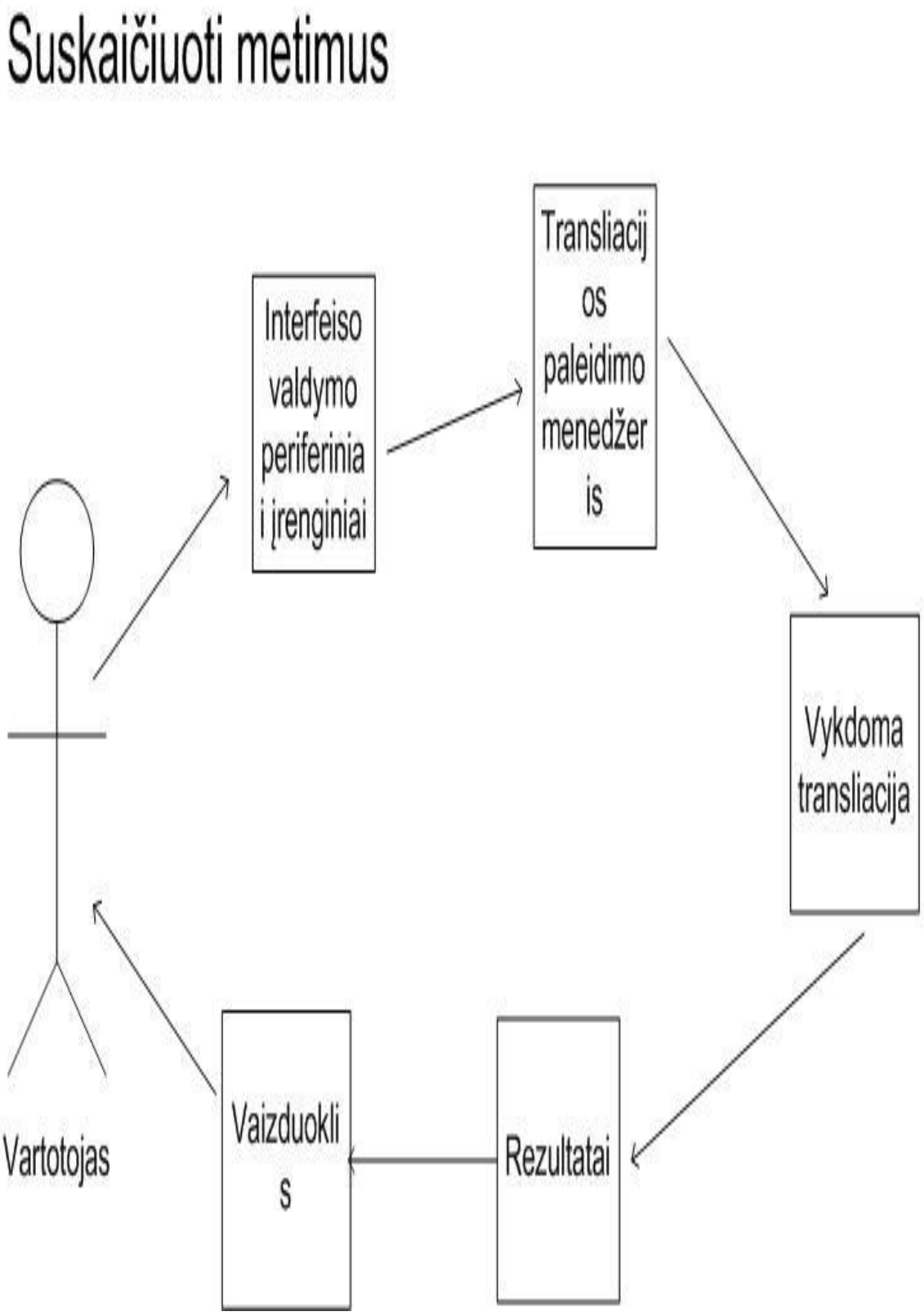
10. PANAUDOJIMO ATVEJIS: Nustatyti laiką tarp metimų
Vartotojas/Aktorius: Paprastas Vartotojas
Aprašas: Nustatyti laiką po kiekvieno atlikto metimo
Prieš sąlyga: Atliekamas metimas
Sužadinimo sąlyga: Kamuolys kerta nurodytas koordinates
Po-sąlyga: Fiksuojamas laiko skirtumas

3.11 Sistemos paketų diagrama



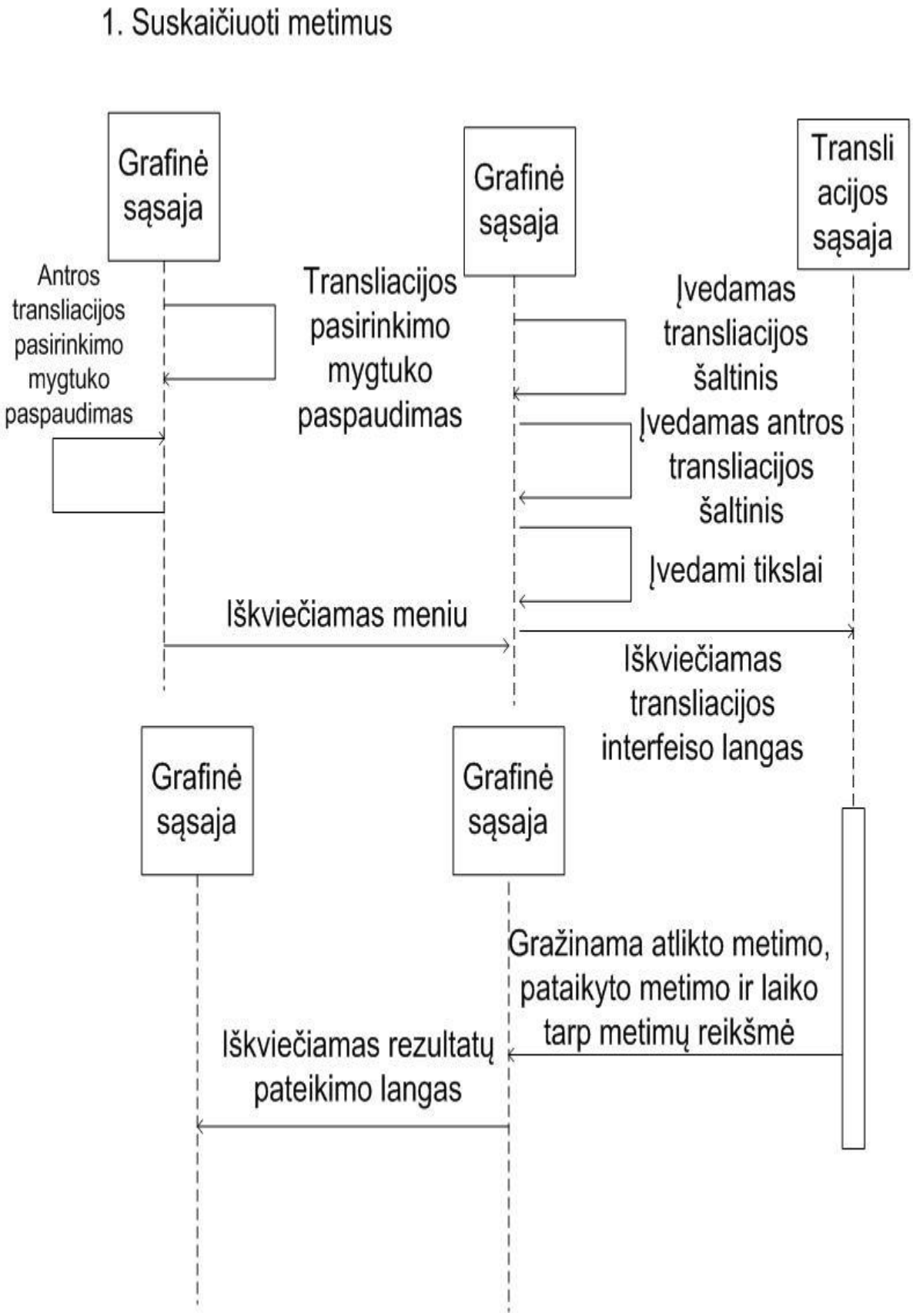
3.11.1 pav. Sistemos paketų diagrama

3.11 Sąveikų diagrama, sistemos panaudos atvejams



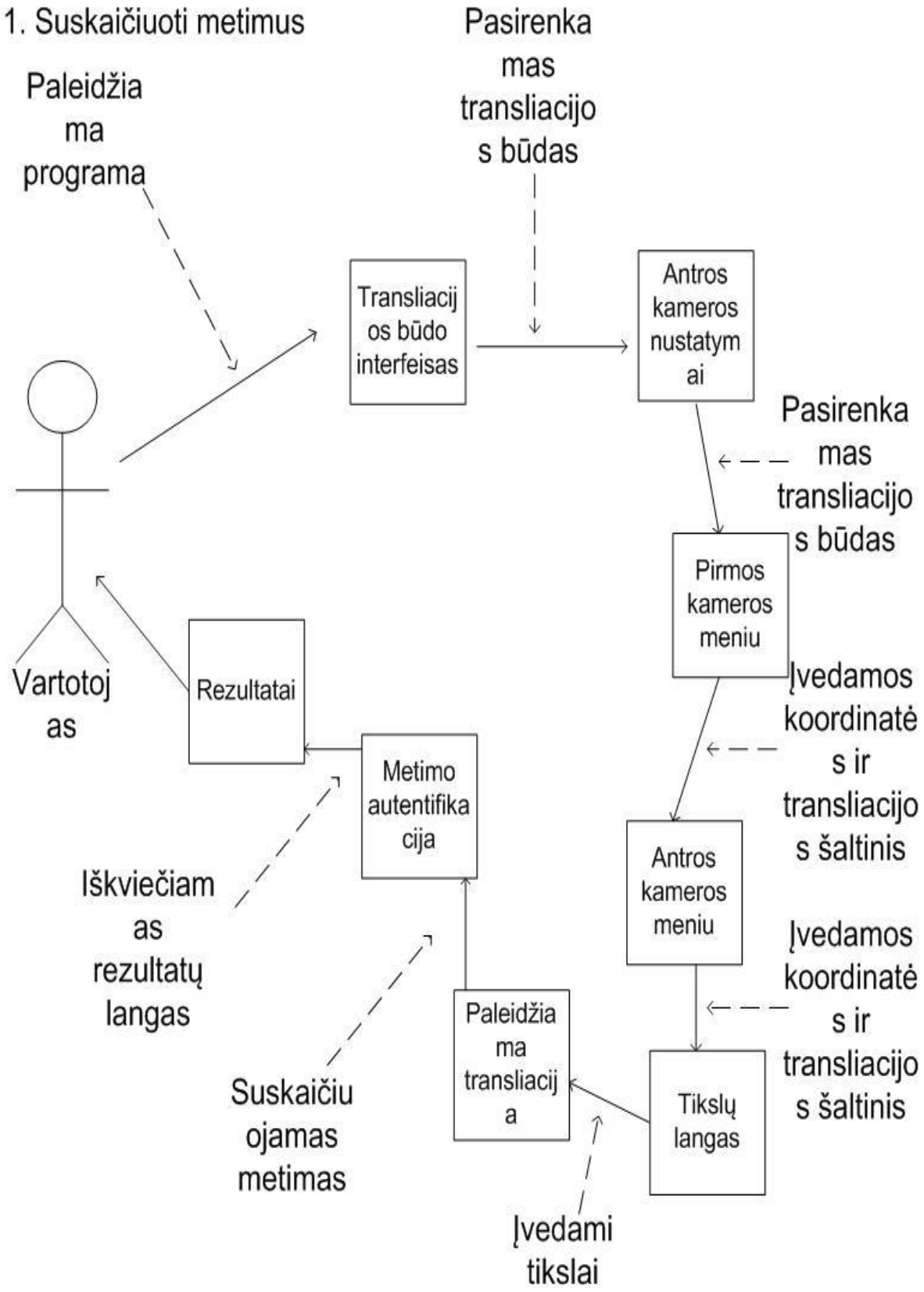
3.11.1 pav. Panaudos atvejo sąveikų diagrama

3.12 Sekų diagrama, sistemos panaudos atvejui



3.12.1 pav. Panaudos atvejo trasos diagrama

3.13 Būsenų diagrama, sistemos panaudos atvejui



3.13.1 pav. Panaudos atvejo būsenos diagrama

4. AUTOMATINIS MESTŲ IR PATAIKYTŲ METIMŲ ATPAŽINIMAS

4.1 Įžanga

Šiame skyriuje aprašyti žingsniai, kurie buvo atlikti, tam kad sistema skaičiuotų išmestus ir pataikytus metimus, kiekvienoje versijoje (žr. 1.2 sk.). Aprašytos kiekvieno varianto kritinės situacijos, bei jų išvengimo būdai. Stebimų zonų apibrėžimo galimybės, *bool* kintamųjų išdėstymo tvarka, bei apibrėžtų pozicijų pasirinkimas kiekvienoje kameroje.

4.2 Pirma MPAAS versija

Pirmoje MPAAS versijoje transliacija buvo vykdoma iš video failo, skaičiuojami išmesti ir pataikyti metimai, kamera pastatyta iš šono (žr. 1.2.1 pav.). Buvo sekamas tik krepšinio kamuolys, asmenys metė iš statiškos pozicijos, be to sistema negalėjo atskirti, kuris žmogus atliko arba pataikė metimą. Spalvų reikšmės programų sistemai buvo perduodamos slaiderių *(angl. sliders)* pagalba, o stebimos dvi konkrečios zonos apibrėžiamos keturiais pikselių taškais - viršutinė kadro dalis, ir apatinė pozicija po krepšio tinkleliu. Taip pat reikalingos ne tik konkretaus objekto koordinatės esamame kadre, bet ir prieš tai buvusio paveikslo to paties daikto arba asmens pozicija. Tada galima sudaryti tokį salygos sakinį, kad esama objekto koordinatė turi patekti į nurodytą intervalą, o prieš tai buvusi ne. Tokiu būdu yra išvengiama nereikalingų priskaičiavimų. Jeigu nurodoma tik tiesė, pavyzdžiui, x = 600, y = 315, o ne intervalas, tai metimai skaičiuojami nebus, kadangi tokiu atveju sekamas objektas turi būti matomas kiekviename pikselyje ir kiekviename kadre, tačiau tam reikalingas milžiniškų resursų kompiuteris. Naudojant 640x480 pikselių rezoliuciją, reikėtų naudoti bent 15 pikselių ilgį arba plotį, priklausomai nuo to ar keturkampis yra gulksčias ar stačias. 4.2.1 pvz. nurodytas visai tai iliustruojantis sąlygos sakinys.

```
int iLastX = -1; // prieš tai buvusi X koordinatė
int iLastY = -1; // prieš tai buvusi Y koordinatė
int count = 0; // atlikti metimai
while (true)
{
int posX = dM10 / dArea;
int posY = dM01 / dArea;
if (posX > 580 && posX < 600 && posY > 295 && posY > 315 && iLastX < 580 && iLastX > 600
                                        && iLastY < 295 && iLastY > 315 )
  {
                count++;
  }
iLastX = posX;
iLastY = posY;
}
```

4.2.1

Tačiau toks būdas yra reikalaujantis didelio atidumo, kadangi reikia nurodyti daug operatorių. Naudojant vieną kamerą tai dar yra pakankamai efektyvu, tačiau pridėdant antra, trečią ir t.t. metodas pasidaro labai komplikuotas.

Kitas būdas, panaudoti papildomą loginį kintamąjį. Dydis įgyja *true* reikšmę, kada kamuolys patenka į nurodytą zoną, o visais kitais atvejais jis yra *false*. Tada sąlygos sakinyje nurodoma, kad jeigu sviedinio koordinatės patenka į nurodytą intervalą ir *bool* kintamojo reikšmė yra *false*, vadinasi yra išmestas arba pataikytas metimas (žr. 4.2.2 pvz.).

```
bool ar_mete = false;
int count = 0; // atlikti metimai
while (true)
{
int posX = dM10 / dArea;
int posY = dM01 / dArea;
  if (ar_mete = false && posX > 580 && posX < 600 && posY > 295 && posY < 315)
  {
                count++;
                ar_mete = true;
  }
  if (posX < 580 || posX > 600)
  {
                ar_mete = false;
  }
iLastX = posX;
iLastY = posY;
}
```

4.2.2

Šioje programų sistemos versijoje zonos buvo nustatomos naudojant sąlygos sakinius (žr. 4.2.1 pvz.), tačiau taip pat galima naudoti specifines *OpenCV* bibliotekos funkcijas, pavyzdžiui, *Rect2d()* [48] funkciją, kuriai reikia nurodyti kairįjį viršutinį keturkampio tašką (X ir Y koordinates), taip pat ilgį, bei plotį. Viskas yra matuojama pikseliais.

Kritinės situacijos naudojant tokį variantą yra trys: kai kamuolys atšoka nuo žemės į viršutinę kadro zoną, tada yra užfiksuojamas atliktas metimas. Antra klaida, kai metimas neliečia lanko, tokiu atveju yra fiksuojamas pataikymas. Trečia klaida – sistema negali atpažinti metimų iš šono, kadangi sviedinys iškart kerta tiek mesto, tiek pataikyto metimo zonas. Daugiau sistemos klaidų išplaukia dėl netinkamų HSV paletės spalvų parinkimo.

Rezultatai buvo vertinami darant prielaidą, kad HSV paletės reikšmės yra parinktos tinkamos. Versijos bandymams buvo atliktas 100 metimų, bei padarytos 4 klaidos. Kai kamuolys atšoko aukštai nuo žemės. Atvejai kada metimas nepastebimas užfiksuoti nebuvo. 4.2.1 lent. pateikiama išmestų metimų klaidų chronologinė seka.

4.2.1 lentelė. Pirmos versijos mestų metimų skaičiavimo rezultatai

Metimų kiekis	0	10	20	30	40	50	60	70	80	90	100
Klaidų kiekis	0	0	1	1	2	3	3	3	4	4	4

Iš 100 pataikytų metimų buvo užfiksuotos 8 klaidos. Teoriškai įmanoma, kad būtų ir 100 netikslumų, kadangi yra kritinė situacija, kai metimas nesiekia lanko. Atmetus šiuos atvejus, gaunasi dvi klaidas, kurios įvyko dėl nepakankamai gero kamuolio atpažinimo. Dėl atsimušimų į grindis yra sukeliami garso trikdžiai. Kadangi sistemos kodas buvo parašytas taip, kad paskutinė koordinatė turi nepriklausyti nurodytai zonai, o esama priklausyti, tai gaunasi, kad objektas patenka į nurodytą intervalą tada dėl garso esama koordinatė fiksuojama kur nors kitur kadre (nenurodytoje zonoje) ir iš to seka, kad paskutinė koordinatė nebuvo intervale, o esama yra, dėl to priskaičiuojami du metimai, nors turėtų būti tik vienas, kadangi realiame vaizde kamuolys dar nepaliko nurodytos zonos.

4.2.2 lentelė. Pirmos versijos pataikytų metimų skaičiavimo rezultatai

Metimų kiekis	0	10	20	30	40	50	60	70	80	90	100
Klaidų kiekis	0	1	1	2	3	3	5	5	5	7	8

Šis metodas yra pakankamai paprastas ir tikslus, jeigu nėra aktualu, kas atliko ir pataikė metimą. Tačiau pačio tyrimo tikslas buvo, kad metimo pozicija nebūtų statiška ir kad būtų galima sekti daugiau negu vieną žmogų, todėl sistema buvo tobulinama ir toliau.

4.3 Antra MPAAS versija

Antroje MPAAS versijoje transliacija taip pat buvo leidžiama iš video failo, tačiau šiame variante jau sekami ir asmenys, ir kamuolys, kurie buvo atpažįstami naudojant HSV spalvų paletę (žr. 2.3 sk.). Buvo išmėginti du atvejai, kai kamera yra pastatyta iš šono ir iš galo. Skaičiuojami pataikyti ir mesti metimai, kiekvienam žmogui atskirai.

4.3.1 Antra MPAAS versija, kai kamera pastatyta iš šono

Pirmiausia buvo išbandytas atvejis, kai kamera, kuri filmuoja transliacija, pastatyta iš šono (žr. 1.2.2 pav.). Kadangi buvo siekiama, kad sistema atpažintų ir asmenis, todėl zonos kurios fiksuoja atliktą metimą buvo padarytos dinamiškos, o jų padėtis priklauso nuo žmogaus judėjimo pozicijos. Vartotojas gali keisti tik jos ilgį ir plotį. Abu matai yra matuojami pikseliais. Pataikymo pozicija išliko statiška t.y. tokia pat kaip ir pirmoje versijoje (žr. 4.2 sk.).

Norint sukurti dinamišką judėjimo zoną yra stebimos sekamo asmens pikselių X ir Y koordinatės ir pridedamas atitinkamas kiekis pikselių. Tokiu būdu gaunami du taškai, kiti du gaunami atimant pasirinktą kiekį. Taip gaunama dinamiška zona, kadangi nuolat atraminė koordinatė yra asmens būvimo vieta. Pataikytas metimas yra fiksuojamas taip pat kaip ir pirmoje versijoje (žr. 4.2 sk.). Pikselių kiekį, kurį reikia pridėti ir atimti reguliuoja vartotojas. Kaip gauti žmonių būvimo koordinates žiūrėti 2.3.4 pvz..

Tam kad atskirti kuris asmuo pataikė metimą buvo naudojami papildomi du loginiai kintamieji, kurie įgyja *true* reikšmes, kai kamuolys kerta nurodytą zoną. Sąlygos tikrinimo salinys *if* yra panašus, kaip ir pirmoje versijoje, tačiau jame pridedamas stebėjimas, kurio asmens *bool* dydis įgyja

true reikšmę. Tokiu būdu galima nustatyti, kas konkrečiai pataikė metimą. Taip pat reikia pastebėti, kad kiekvieną kartą kada yra fiksuojamas pataikymas visi loginiai kintamieji turėtų įgyti *false* tuo atvejui, jeigu sistema nepastebi taiklaus metimo. Kadangi, jeigu sekantį metimą atlieka ir pataiko kitas žmogus, pataikymas skaičiuosis abiems. Be to papildomai galima nurodyti, kad jeigu yra fiksuojamas kurio nors asmens atliktas metimas, tai visų kitų asmenų loginiai kintamieji, kurie nusako atliktą metimą, turi įgyti *false* reikšmę.

Kai kamera buvo pastatyta iš šono, pasitaikė dvi kritinės situacijos. Pirma dažna kritinė situacija, kad kamuolys atšoka nuo žemės ir kerta nurodytą išmesto metimo zoną. Situacija panaši kaip ir pirmoje versijoje (žr. 4.2 sk.), tačiau gerokai dažnesnė, kadangi sviedinys nebūtinai turi atšokti aukštai. Kita kritinė situacija, tai metimai atlikti iš kampų. Kadangi atliktų metimų zonos yra dinamiškos, o pataikytų metimų ne, bei kamera filmuoja iš šono, tai žmogui atlikus metimą iš kampinės pozicijos kamuolys kerta tiek mesto, tiek ir pataikyto metimo sritis.

Vertinant sistemos darbo rezultatus buvo padaryta prielaida, kad spalvų reikšmės pasirinktos tinkamai. Buvo atlikta 100 metimų ir padarytos 22 klaidos. 19 klaidų pasitaikė dėl kamuolio atšokimo ir 3 klaidos pasirodė dėl netobulo atpažinimo, t.y. garso faktoriaus, kuris plačiau aprašytas prieš tai buvusiame skyriuje (žr. 4.2 sk.). Kadangi pataikymų skaičiavimo specifika niekuo nesiskyrė lyginant su pirmu atveju, todėl papildomas tyrimas nebuvo atliktas. 4.3.1.1 lent. pateikiama chronologinė seka, kai kamera pastatyta iš šono.

4.3.1.1 lentelė. Antros versijos mestų metimų skaičiavimo rezultatai, kai kamera pastatyta iš šono

Metimų kiekis	0	10	20	30	40	50	60	70	80	90	100
Klaidų kiekis	0	1	3	6	7	9	12	17	17	19	22

4.3.2 Antra MPAAS versija, kai kamera pastatyta iš galo

Kitas išbandytas atvejis atvejis, kai kamera buvo pastatyta iš galo (žr. 1.2.3 pav.. Nors paveikslėlis yra iš trečios versijos, tačiau tai toks pat variantas, kaip ir antros) Visa asmenų atpažinimo ir zonų nustatymo specifika yra analogiška (žr. 4.3.1 sk.).

Tuo atveju kai kamera yra pastatyta iš galo rezultatai buvo sėkmingesni, tačiau kritinių situacijų taip pat pasitaikė. Išmetant metimus objektai nuo kito turi laikytis tam tikrą atstumą. Pataikymų atveju papildomi skaičiuojama, kai kertama nurodyta zona, kadangi filmuojant iš galo sviedinys tą sritį gali kirsti ir atšokus. Todėl čia klaidos tikimybė išryškėja. Kadangi vieno pikselio taškas yra labai mažas, todėl nepakanka sistemos kode nurodyti, kad jeigu esama Y koordinatė yra didesnė negu prieš tai buvusi tai reiškia, kad kamuolys krenta žemyn. Pasitaikys atveju, kada nors ir nežymiai bet koordinačių reikšmių seka judės į priešingą pusę. Tai dar viena priežastis, kodėl reikalingos bent dvi kameros siekiant 100% tikslumo rezultatų.

Filmuojant iš šitos pozicijos klaidos pasitaiko tose metimuose, kurie atšoka toli nuo lanko. Taip pat galimos klaidos ir tais atvejais, kada metimai nesiekia lanko, tačiau šį kartą tokių metimų nepasitaikė.

Vertinant rezultatus vėl buvo daroma prielaida, kad atpažinimui reikalingas spalvų reikšmes parinktos tinkamai. Iš 100 mestų metimų buvo padaryta 17 klaidų (žr. 4.3.2.1 lent.). Visos jos buvo dėl blogo kamuolio atšokimo. Šį kartą klaidų dėl garso faktoriaus nepasitaikė (žr. 4.2 sk.).

4.3.2.1 lentelė. Antros versijos mestų metimų skaičiavimo rezultatai, kai kamera pastatyta iš galo

Metimų kiekis	0	10	20	30	40	50	60	70	80	90	100
Klaidų kiekis	0	1	1	2	5	9	10	11	11	14	17

Iš 100 pataikytų metimų buvo padaryta 18 klaidų (žr. 4.3.2.2 lent.), iš jų 16 dėl netinkamo kamuolio atšokimo, 1 klaida pasirodė dėl garso faktoriaus, o kita klaida atsirado dėl galimai prasto atpažinimo, tačiau tai labiau išimtis ir daugiau tokių atvejų nebuvo pasitaikę.

4.3.2.2 lentelė. Antros versijos pataikytų metimų skaičiavimo rezultatai, kai kamera pastatyta iš galo

Metimų kiekis	0	10	20	30	40	50	60	70	80	90	100
Klaidų kiekis	0	2	3	5	8	9	12	13	14	17	18

4.4 Trečia MPAAS versija

Trečioje sistemos versijoje (žr. 1.2.3 ir 1.2.4 pav.) transliacija taip pat buvo paleidžiama iš video failo, be to buvo pradėtas stebėti programinė įrangos veikimas realiu laiku, taip pat pridėta antra kamera. Viena pastatyta iš šono, o kita iš galo. Abiejuose kamerose buvo sekami žmonės, bei krepšinio kamuolys. Objektų iškyrimas iš aplinkos toks pat, kaip ir prieš tai buvusiuose variantuose (žr. 2.3 ir 2.5 sk.). Pataikyto ir atlikto metimo zonos apibrėžiamos kaip ir antroje programos versijoje (žr. 4.3 sk.), o jų skaičiavimas įvykdomas, kai abiejų kamerų loginiai kintamieji įgauna *true* reikšmę. Tam kad atskirti, kas atliko arba pataikė metimą, naudojamas tas pats metodas, kaip ir antroje versijoje (žr. 4.3 sk.). Skirtumas tik tas, kad reikalingas papildomas sąlygos sakinys, kuris tikrina ar abi kameros pastebėjo atliktą metimą. Analogiškas tikrinimas vykdomas ir pataikymo stebėjimui.

Atlikus šiuos pakeitimus kritinės situacijos eliminuojamas. Abi kameros padengia viena kitos netikslumus. Sistemos klaidos gali atsirasti tik dėl nevisiškai tinkamo HSV spalvų reikšmių parinkimo arba garso sukeliamų trikdžių (žr. 4.2 sk.). Tačiau buvo prarastas sistemos veikimo greitis. Todėl programos darbo rezultatai buvo nepatenkinami, kai transliacija filmuojama realiu laiku (žr. 4.4.2 lent.), tačiau transliuojant iš video failo programos veikimas buvo priimtinas. 4.4.1 lent. pateikiami trečios versijos mestų metimų atpažinimo rezultatai, kai video įrašas yra paleidžiamas iš video failo.

Visos prielaidos buvo padarytos tos pačios, kaip ir ankstesniais atvejais (žr. 4.2 ir 4.3 sk.). Iš 100 metimų buvo padarytos 4 klaidos. Priešingai, nei prieš tai buvusiai atvejais, šį kartą visos keturios klaidos pasirodė dėl neatpažinto metimo, kadangi kompiuterio veikimo greitis buvo nepakankamas.

4.4.1 lentelė. Trečios versijos mestų metimų skaičiavimo rezultatai, transliuojant iš video failo

Metimų kiekis	0	10	20	30	40	50	60	70	80	90	100
Klaidų kiekis	0	0	0	1	1	1	3	3	4	4	4

Palyginimui 4.4.2 lent. pateikiami atliktų metimų atpažinimo rezultatai, kai transliacija buvo vykdoma realiu laiku. Iš 100 metimų buvo padarytos 77 klaidos.

4.4.2 lentelė. Trečios versijos mestų metimų skaičiavimo rezultatai, transliuojant realiu laiku

Metimų kiekis	0	10	20	30	40	50	60	70	80	90	100
Klaidų kiekis	0	7	18	26	34	40	47	56	61	68	77

4.4.3 lent. pateikti pataikytų metimų rezultatai, kai sistemos transliacija vykdoma iš video failo. Kadangi eksperimento rezultatai buvo fiksuojami vienu metu ir sistema stebi ar yra taiklus metimas tik tada, kai pastebimas atliktas metimas, tai viso gaunasi 96 metimai, kadangi 4 metimai buvo neatpažinti. Dėl to, kad atliktų metimų rezultatai realiu laiku buvo nepatenkinami todėl pataikymai atskirai nebuvo tikrinami.

4.4.3 lentelė. Trečios versijos pataikytų metimų skaičiavimo rezultatai, transliuojant iš video failo

Metimų kiekis	0	10	20	30	40	50	60	70	80	90	96
Klaidų kiekis	0	0	0	0	0	1	1	1	2	2	3

4.4.4 lent. pateikiami trečios versijos greičio rezultatai.

4.4.4 lentelė. Trečios versijos veikimo greičio palyginimas su realiu laiku

Įrašo trukmė	115s	128s	179s	142s	122s
Įrašo trukmė sistemoje	164s	181s	252s	201s	172s

4.5 Ketvirta MPAAS versija

Ketvirtoje ir kol kas paskutinėje sistemos versijoje (žr. 1.2.5 ir 1.2.6 pav.) pakeitimai buvo atlikti šoninėje kameroje. Kadangi trečiame sistemos variante greitis nebuvo patenkinamas (žr. 4.4.4 lent.) antroje kameroje nuspręsta naudoti statiškas metimų zonas ir sekti tik kamuolį, analogiškai, kaip ir pirmoje versijoje (žr. 4.2 sk.). Galinė kameroje viskas išliko taip pat (žr. 4.4 sk.). Tokiu būdu

buvo padidintas sistemos veikimo greitis (žr. 4.5.5 lent.), neprarandant tikslumo (žr. 4.5.1, 4.5.2, 4.5.3, 4.5.4 lent.). Visa objektų išskyrimo procedūra ir spalvų reikšmių parinkimas yra analogiškas prieš tai buvusiems programos variantams.

Reikia pastebėti, kad šį kartą sąlyga ne tik tikrina ar kamuolys patenka į nurodytą zoną, bet ir ar pirma kamera pastebėjo atliktą metimą. Jeigu galinė kamera nepastebėjo metimo, tai metimas niekada nebus užskaitomas. Taip yra todėl, kad programų sistemoje transliacija abiejuose kamerose yra laikoma, kaip begalinė kadrų seka, todėl yra pradedamas begalinis ciklas. Remiantis tokiu principu visada kamuolys pirmiau bus pastebimas pirmoje kameroje, kadangi sritis stebima yra kylančio sviedinio trajektorijoje, o antroje kameroje stebimas plotas yra ten, kur žaidimo įrankis yra netoli aukščiausio taško arba pačiame aukščiausiame taške.

Tada yra atliekamas dar vienas tikrinamas, ar abu *boolean* tipo kintamieji yra *true*. Jei sąlyga yra tenkinama, užskaitomas atliktas metimas, bei dar vienas *bool* dydis, skirtas atskirtis kas metė, įgauna *true* reikšmę. Jis bus reikalingas vėliau, nustatant, kas pataikė. Kadangi galinėje kameroje mestų metimų zonos *bool* kintamieji yra atskiri kiekvienam asmeniui, taip yra atskiriamas žmogus, kuris atliko metimą. Šioje vietoje visi kintamieji, kurie buvo naudoti identifikuojant atliktą metimą įgauna *false* reikšmę, tik vienintelis indikatorius apie tai, kas atliko metimą yra *true*. Taip yra išvengiama klaidų, bei nereikalingų priskaičiavimų.

Toliau seka pataikyto metimo tikrinimas. Viskas atliekama identiškai, tik abiejų kamerų zonos stebimos toje pačioje vietoje. Todėl teoriškai pataikytas metimas pastebimas vienu metu. Pirmiausia vykdomas tikrinimas ar viena ir kita kamera užfiksavo metimą. Taip pat salygos sakinyje reikia nurodyti kintamąjį, kuris nusako, kas atliko metimą. Toliau seka tikrinimas, jeigu abi kameros pamatė pataikytą metimą jis yra užskaitomas konkrečiam asmeniui. Tikrinimo pabaigoje visi *bool* kintamieji, kurie skirti nustatyti pataikytą metimą prilyginami *false* reikšmei. Taip pat svarbu begalinio ciklo pabaigoje esamas koordinates prilyginti buvusioms, tai padeda lengviau apibrėžti reikalingas sritis. Pagal poreikį tokių dydžių gali būti ir daugiau.

Tam kad būtų galima transliuoti kadrų seką paprasčiausias būtas naudoti *imshow()* [49] funkciją. Jeigu norima rodyti kelias video transiacijas reikia kurti vieną didelį langą, perkuopijuojant visus norimus rėmus. Tam tikslui galima naudoti *copyTo()* [50] funkciją. Tačiau svarbu, kad kiekvienos kadrų sekos spalvų kanalų kiekis būtų vienodas, o tai galima padaryti naudojant *cvtColor()* funkciją.

Ketvirtos MPAAS veikimo tikslumas nepakito. 4.5.1 lent. pateikiami išmestų metimų skaičiavimo rezultatai, kai transliacijos paleidžiamos iš video failą.

4.5.1 lentelė. Ketvirtos versijos mestų metimų skaičiavimo rezultatai, transliuojant iš video failo

Metimų kiekis	0	10	20	30	40	50	60	70	80	90	100
Klaidų kiekis	0	1	0	1	2	2	2	3	4	5	5

Visos prielaidos vertinant rezultatus buvo tos pačios. Iš 100 metimų buvo padarytos 5 klaidos. Lyginant su trečios versijos rezultatais gauname, kad sistemos veikimas beveik nepakito, o klaidos atsirado dėl nepakankamo kompiuterio greičio. Palyginimui, 4.5.2 lent. pateikiamas metimų skaičiavimas realiu laiku.

4.5.2 lentelė. Ketvirtos versijos mestų metimų skaičiavimo rezultatai, transliuojant realiu laiku

Metimų kiekis	0	10	20	30	40	50	60	70	80	90	100
Klaidų kiekis	0	1	2	3	3	4	5	7	7	8	9

Kameros su programų sistema buvo sujungtos naudojant *wi-fi* tinklą ir *IP Webcam* (žr. 7.1 sk.) mobiliąją programėlę. Iš 100 metimų buvo padarytos 9 klaidos. Iš jų 6 dėl nepakankamo sistemos veikimo greičio, o trys dėl netinkamai veikiančio duomenų perdavimo tinklo. Tai tokie atvejai, kai video transliacija stabtelėdavo ir buvo praleidžiami keli kadrai, kuriose buvo atliekami metimai. Tačiau sistemos veikimas buvo tinkamas. 4.5.3 lent. pateikiamas pataikytų metimų skaičiavimas, kai transliacija vykdoma iš failo.

4.5.3 lentelė. Ketvirtos versijos pataikytų metimų skaičiavimo rezultatai, transliuojant iš video failo

Metimų kiekis	0	10	20	30	40	50	60	70	80	90	95
Klaidų kiekis	0	0	1	1	2	3	3	3	4	4	4

Iš 95 pataikytų metimų (metimų kurie buvo pastebėti, žr. 4.5.1 lent.) buvo padarytos 4 klaidos. Palyginimui, 4.5.4 lent. pateikiami pataikytų metimų skaičiavimo rezultatai, kai transliacija buvo vykdoma realiu laiku. Lyginant su trečiąja sistemos versija sistemos veikimo rezultatai realiu laiku ženkliai pagerėjo.

4.5.4 lentelė. Ketvirtos versijos pataikytų metimų skaičiavimo rezultatai, transliuojant realiu laiku

Metimų kiekis	0	10	20	30	40	50	60	70	80	90	91
Klaidų kiekis	0	1	2	4	5	6	7	7	8	9	11

4.5.5 lentelė. Ketvirtos versijos veikimo greičio palyginimas su realiu laiku

Įrašo trukmė	115s	128s	179s	142s	122s
Įrašo trukmė sistemoje	114s	127s	176s	140s	121s

5. IŠVADOS

1. Remiantis tyrimo rezultatais (žr. 2.6 sk.) naudojant spalvų atpažinimo technologija kartu su judėjimo atpažinimu, naudingiau yra pirmiausia objektus atpažinti pagal spalvas, o pagal judėjimą juos patikslinti.
2. Remiantis tyrimo rezultatais (žr. 4.2, 4.3, 4.4 ir 4.5 sk.), reikalingos bent dvi kameros, tam kad būtų galima pasiekti 100% tikslumą skaičiuojant atliktus ir pataikytus metimus.
3. Remiantis tyrimo rezultatais (žr. 4.4, 4.5 sk.), naudojant kompiuterį, kurio procesorius yra i3 2GHz, 8 GB RAM dvi kameros maksimalus kiekis, tam kad sistema veiktų tolygiai realiu laiku.
4. Remiantis tyrimo rezultatais (žr. 2.4 sk.), kol kas nei vienas objektų sekimo metodas iš OpenCV papildomų bibliotekų, nėra tinkamas naudoti, kuomet reikia sekti krepšinio kamuolį arba asmenis.

6. LITERATŪROS SĄRAŠAS

[1] Emad Monier, Per Wilhem, Ulrich Ru¨ckert, A Computer Vision Based Tracking System for indoor Team Sports, Introduction(1pp.), Germany, System of circuit Technology, Heinz Nixford Institute, University of Paderborn.

[2] žiūrėta [2018-05-05] https://www.goalcontrol.de/en/

[3] žiūrėta [2018-05-05] https://www.fusionsport.com/about/

[4] Andrii Maksai, Xinchao Wang, Pascal Fua, What Players do with the Ball: A Phisically Constrained Interaction Interaction Modeling, (1-10 psl), Computer Vision Laboratory, Ecole Polytechnique F´ed´erale de Lausanne (EPFL)

[5] Paresh R. Kamble, Avinash G. Keskar, Kishor M. Bhurcandi, Ball tracking in sports: a survey, Springer Netherlands, Online ISSN 1573-7462

[6] žiūrėta [2018-04-14] https://www.hawkeyeinnovations.com/products/ball-tracking

[7] žiūrėta [2018-03-06] https://docs.opencv.org/3.1.0/d0/d0a/classcv_1_1Tracker.html

[8] žiūrėta [2018-04-30] https://alloyui.com/examples/color-picker/hsv

[9] žiūrėta [2018-04-17] https://docs.opencv.org/2.4/modules/highgui/doc/user_interface.html?highlight=setmousecallback

[10] žiūrėta [2018-04-17] http://opencvexamples.blogspot.com/2014/01/detect-mouse-clicks-and-moves-on-image.html

[11] žiūrėta [2018-05-19] https://docs.opencv.org/3.1.0/db/d4e/classcv_1_1Point__.html

[12] žiūrėta [2018-04-17] https://www.tutorialcup.com/cplusplus/nested-loop.htm

[13] žiūrėta [2018-04-17] https://docs.opencv.org/2.4.13.2/doc/user_guide/ug_mat.html

[14] žiūrėta [2018-04-17] http://www.cplusplus.com/reference/algorithm/min_element/

[15] žiūrėta [2018-04-17] http://www.cplusplus.com/reference/algorithm/max_element/

[16] žiūrėta [2018-04-30] https://docs.opencv.org/3.1.0/d7/d1b/group__imgproc__misc.html#ga397ae87e1288a81d2363b61574eb8cab

[17] žiūrėta [2018-03-05] https://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html

[18] žiūrėta [2018-04-30]

https://docs.opencv.org/trunk/d2/de8/group__core__array.html#ga48af0ab51e36436c5d04340e036ce981

[19] žiūrėta [2018-03-05] https://docs.opencv.org/2.4/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html

[20] žiūrėta [2018-03-05] https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=erode#erode

[21] žiūrėta [2018-04-17] https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=moments#moments

[22] D.S.G. Pollock, CIRCULANT MATRICES AND TIME-SERIES ANALYSIS, Circulant Matrices and Polynomials (2pp.), England, Queen Mary, University of London

[23] žiūrėta [2018-03-07] https://photographyconcentrate.com/10-reasons-why-you-should-be-shooting-raw/

[24] Keyu Qi, Zhengjia Heng, Yangyang Zi, Cosine window-based boundary processing method for EMD and its application in rubbing fault diagnosis, Cosine window-based boundary processing method, China, School of Mechanical Engineering, Xi'an Jiaotong University.

[25] žiūrėta [2018-03-07] https://en.wikipedia.org/wiki/Discrete_Fourier_transform#cite_note-4

[26] žiūrėta [2018-03-06] https://docs.opencv.org/3.2.0/d2/dff/classcv_1_1TrackerKCF.html

[27] Andres Solıs Montero, Jochen Lang, Robert Lagani`ere, Scalable Kernel Correlation Filter with Sparse Feature Integration, KCF Algorithm (25-26pp.), Canada, School of Electrical Engineering and Computer Science, University of Ottawa

[28] Jilin Hu, Shiqiang Hu, Zhuojin Sun, A Real Time Dual-Camera Surveillance System Based On Tracking-Learning-Detection Algorithm, Tracking-Learning-Detection (888pp.), China, School of Aeronautics and Astronautics, Shanghai Jiao Tong University

[29] Vijay K. Sharma, K.K. Mahapatra, MIL based visual object tracking with kernel and scale adaptation, Abstract (1pp.), Positive and negative example selection (53pp.), Feature extraction and appearance model (53pp.), India, Department of Electronics and Communication Engineering, National Institute of Technology

[30] žiūrėta [2018-03-06] https://lt.wikipedia.org/wiki/Mediana

[31] žiūrėta [2018-03-09]
http://www.lidata.eu/index.php?file=files/mokymai/sda/sda.html&course_file=sda_I_1.3.html

[32] žiūrėta [2018-03-09] https://lt.wikipedia.org/wiki/Skirstinys

[33] žiūrėta [2018-03-09] https://lt.wiktionary.org/wiki/variacin%C4%97_eilut%C4%97

[34] Zdenek Kalal, Krystian Mikolajczyk, Jiri Matas, Forward-Backward Error: Automatic Detection of Tracking Failures, Object Tracking by median flow (2758pp.), 2010 International Conference on Pattern Recognition

[35] žiūrėta [2018-03-09] https://docs.opencv.org/3.1.0/d7/d86/classcv_1_1TrackerMedianFlow.html

[36] Freund Y, Schapire R. Experiments With a New Boosting Algorithm. In: Proceedings of the Thirteenth International Conference on Machine Learning Theory. San Francisco, CA: San Francisco: Morgan Kaufmann Publishers Inc.; 1996. p. 148{156.

[37] žiūrėta [2018-03-11] https://docs.opencv.org/3.1.0/d1/d1a/classcv_1_1TrackerBoosting.html

[38] Andreas Mayr, Harald Binder, Olaf Gefeller, Matthias Schmid, The Evolution of Boosting Algorithms From Machine Learning to Statistical Modelling, Abstract(1pp.), Germany, Institut für Medizininformatik, Biometrie und Epidemiologie, Friedrich-Alexander-Universitāt Erlangen-Ne ürnberg, Institut für Medizinische Biometrie, Epidemiologie und Informatik, Johannes Gutenberg-Universitāt Mainz, Institut für Medizinische Biometrie, Informatik und Epidemiologie, Rheinische Friedrich-Wilhelms-Universitāt Bonn

[39] žiūrėta [2018-03-12] https://docs.opencv.org/2.4/modules/core/doc/operations_on_arrays.html

[40] žiūrėta [2018-03-12] https://docs.opencv.org/2.4/doc/tutorials/imgproc/threshold/threshold.html

[41] žiūrėta [2018-03-12] https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=blur#blur

[42] žiūrėta [2018-03-12]
https://docs.opencv.org/2.4/doc/tutorials/imgproc/gausian_median_blur_bilateral_filter/gausian_median_blur_bilateral_filter.html

[43] žiūrėta [2018-04-13] https://www.tutorialspoint.com/cpp_standard_library/vector.htm

[44] žiūrėta [2018-04-13]
https://docs.opencv.org/3.1.0/d3/dc0/group__imgproc__shape.html#ga17ed9f5d79ae97bd4c7cf18403e1689a

[45] žiūrėta [2018-04-13]
https://docs.opencv.org/3.1.0/d3/dc0/group__imgproc__shape.html#ga819779b9857cc2f8601e6526a3a5bc71

[46] žiūrėta [2018-04-13]
https://docs.opencv.org/3.1.0/d3/dc0/group__imgproc__shape.html#ga4303f45752694956374734a03c54d5ff

[47] žiūrėta [2018-04-13] http://www.cplusplus.com/reference/vector/vector/push_back/

[48] žiūrėta [2018-04-03]
https://docs.opencv.org/trunk/dc/d84/group__core__basic.html#ga894fe0940d40d4d65f008a2ca4e616bd

[49] žiūrėta [2018-04-05]
https://docs.opencv.org/3.1.0/d7/dfc/group__highgui.html#ga453d42fe4cb60e5723281a89973ee563
[50] žiūrėta [2018-04-05] https://docs.opencv.org/3.1.0/d3/d63/classcv_1_1Mat.html
[51] žiūrėta [2018-03-13] https://play.google.com/store/apps/details?id=com.pas.webcam&hl=lt

7. PRIEDAI

7.1 IP WEBCAM

Tai papildoma programa [51], kuri leidžia perduoti telefono kameros filmuojamą vaizdą kompiuteriui, naudojant wi-fi ryšį. Tokia programa buvo pasirinkta dėl patogumo, kadangi ją pakanka įsirašyti tik telefone, o kompiuteryje ji yra valdoma naudojant bet kurią interneto naršyklę. Paleidus programėlę reikia paspausti *(angl. Start server).* Tačiau svarbu, kad visi įrenginiai būtų prisijungę prie to paties *wi-fi* tinklo. Kad pasiekti transliaciją per C++ programą, pakanka nurodyti transliacijos šaltinį *videocapture* kintamąjame.

```
VideoCapture cap(„http://xxx.xxx.x.xxx:xxxx/video?x.mpjeg“);
```

7.1.1.

Filmuojant eksperimentą tekdavo susidurti su tuo, kad sporto salėje nebūdavo interneto ryšio. Kaip *wi-fi* šaltinį galima naudoti vieną iš telefonų su mobiliuoju internetu, sukuriant interneto šaltinį *(angl. Mobile Hotspot)*, bei prie jo prijungiant norimus įrenginius. Kiekvienas telefonas atskirai nurodo, kiek įrenginių galima prijungti prie tinklo. (Eksperimento metu mažiausiai 4 įrenginiai buvo leidžiami)