LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

VELOBRAUCĒJU KUSTĪBAS TRAJEKTORIJU ANALĪZE, IZMANTOJOT IOS VISION ATTĒLU ATPAZĪŠANU

BAKALAURA DARBS

Autors: Mārcis Nīmants
Studenta apliecības Nr.: mn11023
Darba vadītājs: M. dat. Oskars Ozols

RĪGA 2019

ANOTĀCIJA

Pārprojektējot krustojumus vai tos slēdzot remontdarbu laikā, ir svarīgi organizēt satiksmi, lai tā apmierinātu esošo dalībnieku pārvietošanās plūsmu un intensitāti. Velo braucēju trajektoriju atzīmēšana ielas attēlā ir viens no uzskatāmiem pamatojumiem, plānojot šādas izmaiņas. Darbā tiek aprakstīti iespējamie risinājumi velo braucēju atpazīšanā ar konvolūciju neironu tīkliem un izsekošanā ar datorredzi. Izvēlētie risinājumi tiek izmantoti mobilās lietotnes izstrādē uz iOS operētājsistēmas ierīcēm, kas reāllaikā spētu reģistrēt velo braucēju pārvietošanos un atzīmēt veiktās trajektorijas virs videomateriāla. Šāds rezultāts tika sasniegts, kas potenciāli spētu risināt šādu problēmu ātrāk nekā reāllaikā, ja tiek izmantots iepriekš uzņemts video materiāls, nevis ierīces kamera.

Atslēgvārdi: konvolūciju neironu tīkli, datorredze, objektu atpazīšana, objektu izsekošana, iOS operētājsistēma

ABSTRACT

CYCLIST TRAJECTORY ANALYSIS USING IOS VISION IMAGE RECOGNITION

When designing intersections or closing them during construction work it is crucial to satisfy current flow and intensity of traffic participants. Visualizing trajectories performed by cyclists is one of the ways to prove existing demand when making such changes. Paper shows possible solutions to detect cyclists using convolutional neural networks and track them with computer vision. Feasible solutions are used to develop real-time iOS application which can track cyclists and visualize performed trajectories above image. Developed solution has potential to perform this faster than real-time if previously captured video is processed instead of live capture from device's camera.

Keywords: convolutional neural networks, computer vision, object detection, object tracking, iOS operating system

SATURA RĀDĪTĀJS

APZĪMĒJUMU SARAKSTS

Core ML – ietvars mašīnmācīšanās modeļu pārveidei formātā, kas tiek atbalstīts iOS mobilajās ierīcēs;

iOS operētājsistēma – mobilā operētājsistēma, kas tiek izmantota iPhone un iPad viedierīcēs;

Konvolūciju neironu tīkli – dziļo neironu tīklu paveids, kas tiek izmantots attēlu atpazīšanā;

Lineāra atbalsta vektoru mašīna (SVM) – mašīnmācīšanās algoritms, kas tiek izmantots klasifikācijas un regresijas uzdevumos;

Pascal VOC datu kopa –standarts vizuālu objektu klasifikācijā un atpazīšanā pēc kā tiek vērtēti datorredzes un mašīnmācīšanās risinājumi;

Robežtaisnstūris – objektu ierobežojoša taisnstūrveida ģeometriska forma, kas apraksta tā izmēru un atrašanās vietu attēlā;

Xcode – integrētā izstrādes video iOS mobilo lietotņu izstrādē;

YOLO – mašīnmācīšanās modelis objektu atpazīšanai un lokalizēšanai, izmantojot konvolūciju neironu tīklus.

IEVADS

Projektējot satiksmes infrastruktūru krustojumos, viens no vērā ņemamiem aspektiem ir satiksmes dalībnieku pārvietošanās trajektorijas. Apvienība "Pilsēta cilvēkiem" veic regulāru velo satiksmes uzskaiti, kā arī ielu krustojumu attēlos atzīmē velobraucēju pārvietošanās trajektorijas. Trajektorijas tiek atzīmētas ar mērķi iegūt uzskatāmus datus, lai ielas posma slēgšanas vai pārprojektēšanas gadījumā tiktu apmierināta droša un efektīva velo satiksme. Līdz šim trajektoriju atzīmēšanai nepieciešami ievērojami apvienības darbinieku laika resursi. Lai iegūtu šos datus, pirmkārt, nepieciešams nofilmēt krustojumu stundas garumā. Otrkārt, ievākto videomateriālu pārskatīšanai nepieciešams tik pat daudz vai vairāk laiks, atzīmējot veiktās trajektorijas ielas attēlā.

Ņemot vērā autora iepriekšējo pieredzi iOS lietotņu izstrādē, kā arī iOS mobilās operētājsistēmas attīstību mašīnmācīšanās tehnoloģijās, tika izvēlēts risinājums veidot atpazīšanas programmatūru ierīcēm ar attiecīgo operētājsistēmu. Turklāt šāds risinājums ir izdevīgs apvienības darbiniekiem, jo tam nav nepieciešama centralizēta servera infrastruktūra, kas iekļautu papildus izstrādes un uzturēšanas izmaksas.

Darba mērķis ir automatizēt trajektoriju atzīmēšanu ielas attēlā. Tehniskais risinājums darbosies iOS operētājsistēmas mobilajās ierīcēs. Lai veiksmīgi realizētu izvirzīto mērķi, paveicamie uzdevumi ir:

1. Izvēlēties atbilstošāko mašīnmācīšanās modeli objektu atpazīšanai;
2. Veikt atpazīto objektu (velobraucēju) izsekošanu videomateriālā;
3. Atzīmēt izsekoto objektu veiktās trajektorijas.

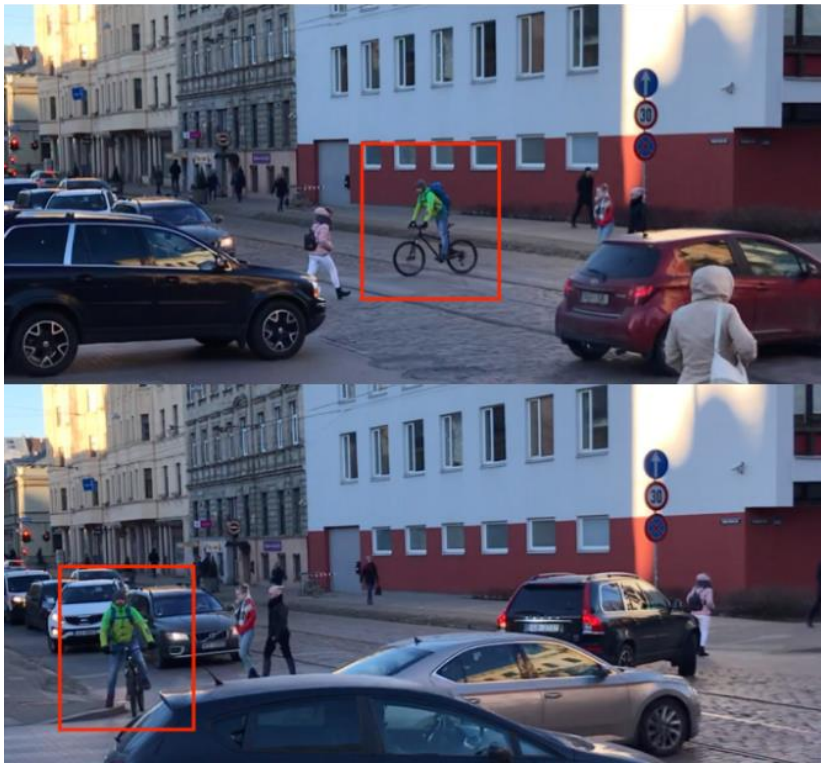
Tika izvirzīta hipotēze, ka objektu atpazīšana, kombinēta ar objektu izsekošanu, var strādāt reāllaikā uz mobilās ierīces.

Darba pirmajā nodaļā tiek aprakstīti iespējamie objektu atpazīšanas risinājumi. Otrajā nodaļā tiek aprakstīta objektu izsekošana. Darba trešajā nodaļā tiek aprakstīta mobilās lietotnes izstrāde, kur izmantoti tehniskie risinājumi, kas izvēlēti pirmajā un otrajā nodaļā.

1. OBJEKTU ATPAZĪŠANA ATTĒLĀ

Objektu atpazīšanu attēlā varētu iedalīt trīs kategorijās - klasifikācija, atrašanās vietas noteikšana un semantiskā segmentācija. Klasifikācijas gadījumā tiek iegūta informācija par to, kādā kategorijā ietilpst attēlā dominējošais objekts, bet netiek norādīta objekta atrašanās vieta. Meklējot objekta atrašanās vietu attēlā, rezultātā tiek iegūts objekta ierobežojošs robežtaisnstūris, kā arī meklējamā objekta klasifikācija, piemēram, vai atpazītais objekts ir automašīna vai velosipēds. Semantiskās segmentācijas gadījumā, kas apvienota ar klasifikāciju, tiek iegūta meklējamā objekta forma un tā atrašanās vieta attēlā.

Ņemot vērā problēmas specifiku - velosipēdisti, kas pārvietojas pilsētas satiksmē un var mainīt pārvietošanās virzienu, interesanta ir tieši objekta atrašanās vietas noteikšana. 1.1. attēlā redzams, kā velosipēdists mainījis braukšanas virzienu un dažādos kadros neizskatās vienādi. Tāpēc semantiskā segmentācija šajā gadījumā nedotu noderīgu informāciju, vai divos dažādos kadros velosipēdists ir viens un tas pats. Savukārt klasifikācijas rezultāts neatrisinātu tālākās izsekošanas problēmu, jo nebūtu zināms izsekojamā objekta robežtaisnstūris.



1.1. att. **Video kadrējumi ar velobraucēju, kurš maina virzienu**

Nodaļā tiek apskatīti iespējamie varianti objektu atpazīšanai, izmantojot dažādus konvolūciju neironu tīklus un tiek vērtēta to atbilstība izvirzīto problēmu risināšanai.

1.1. Konvolūciju neironu tīkli (CNN)

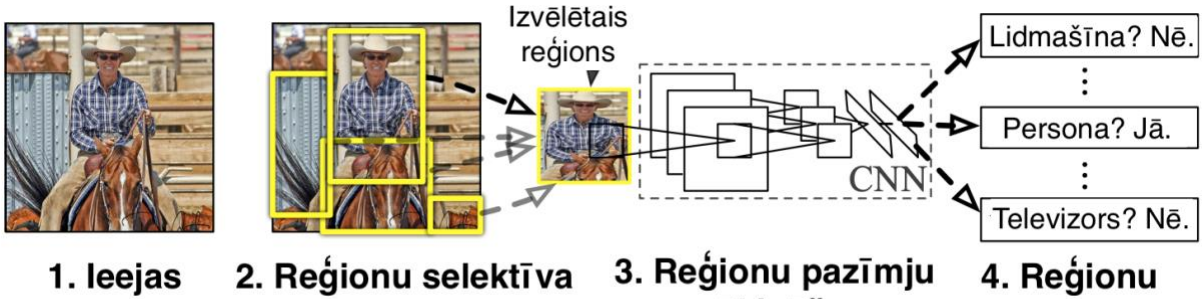
Attēlu klasifikācijas problēmu ir iespējams atrisināt ar konvolūciju neironu tīklu, taču tas nenosaka objekta atrašanās vietu attēlā. Tomēr, sadalot attēlu vairākos reģionos un iegūstot jaunus attēlus, iespējams klasificēt šos reģionus un aptuveni noteikt objekta atrašanās vietu sākotnējā attēlā. Attēlā 1.2. redzams, kā attēls sadalīts 4 vienāda izmēra reģionos, kur tiks veikta klasifikācija. Taču izmantojot šādu metodi, iegūtie reģioni var precīzi neatbilst meklējamam objektam, kā arī šai metodei nepieciešami lieli skaitļošanas resursi, ja klasifikācija tiek veikta daudz vairāk sadalītos reģionos. [1]

1.2. att. **Ieejas attēls sadalīts vairākos vienāda izmēra reģionos** [1]

1.2. Reģionos bāzēti konvolūciju neironu tīkli (*R-CNN*)

Reģionos bāzētos konvolūciju neironu tīklos netiek veikta pilnīga reģionu pārlase un klasifikācija, bet reģioni, pirms to klasifikācijas, tiek atrasti selektīvi [2].

Attēlā 1.3. attēlota *R-CNN* darbība:



1.3. att. **Reģionos bāzēta konvolūciju tīkla darbība** [2]

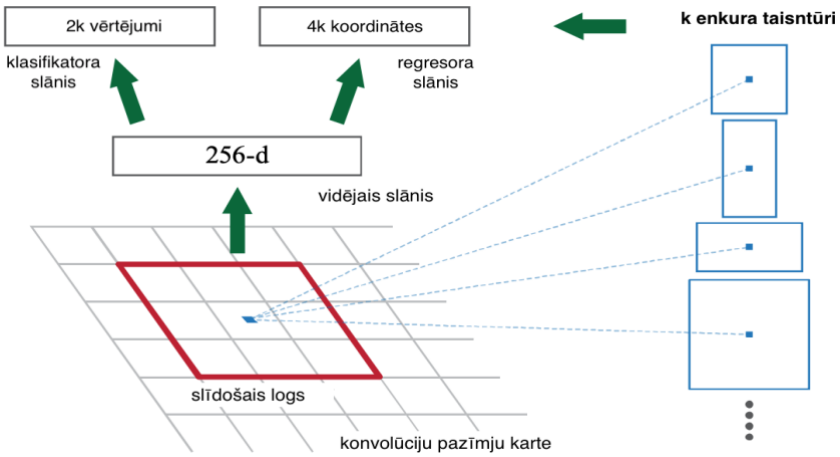
1. solī tiek atlasīts ievaddatu attēls un pārveidots nepieciešamajā izmērā;
2. solī tiek veikta reģionu ierosināšana ar kādu no metodēm, kas aprakstītas 1.1.2. apakšnodaļā;
3. solī tiek aprēķinātas izvēlēto reģionu pazīmes, pārveidojot attēlu 227x227px RGB formātā un pielietojot 5 konvolūciju un 2 pilnsaistes slāņus;

4. solī reģioni tiek klasificēti, izmantojot lineāru atbalsta vektoru mašīnu (*SVM*).

1.2.1 Reģionu ierosināšana

Dažkārt saukti par reģionu ierosinātāju tīkliem (*RPN - Region Proposal Network*) [3], šie konvolūciju neironu tīkli tiek izmantoti kā daļa no reģionos bāzētiem konvolūciju neironu tīkliem.

Attēlā 1.4. parādīta tīkla darbība. Lai izveidotu reģionu ierosinājumus, pāri konvolūciju pazīmju kartei tiek virzīts slīdošais logs. Katrā slīdošajā logā tiek noteikti *k* enkura taisnstūri. *k* apzīmē maksimālo daudzumu ierosinājumu katrā slīdošajā logā. Šie *k* taisnstūri tiek padoti uz klasifikatora un regresora slāņiem. Klasifikatora slānī tiek aprēķināta robežtaisnstūra iespējamība, ka tas satur meklējamo objektu, kā arī nesatur meklējamo objektu, tāpēc rezultāts ir *2k* vērtējumi. Regresora slānī tiek precizētas ierosinātā reģiona koordinātes. Regresora slāņa rezultāts katra *k* taisnstūra koordinātes (*x*, *y*, platums, augstums), kas kopā saskaitot veido *4k*. Klasifikatora slāni saturošie konvolūciju slāņi ir kopīgi ar reģionos bāzēto konvolūciju neirona tīkla slāņiem, jo abos tīklos tiek meklētas līdzīgas attēla pazīmes. [3]



1.4. att. **Reģionu ierosinātāja tīkls** [3]

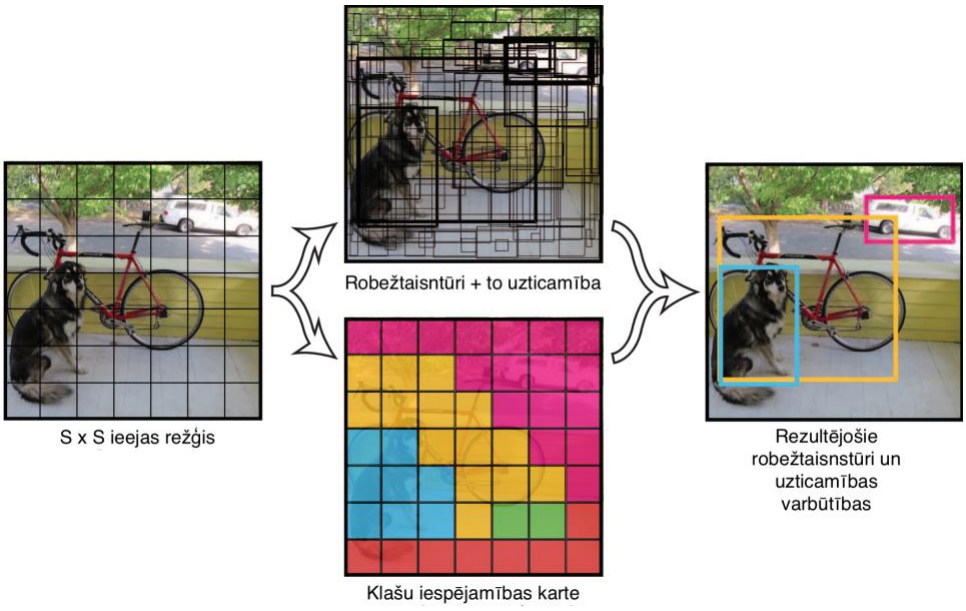
1.3. YOLO [4]

YOLO darbība virspusīgi tiek parādīta attēlā 1.5.

1. solī ieejas bildei tiek mainīts lielums uz 448x448px;
2. solī tiek darbināts viens konvolūciju neironu tīkls, kas nosaka aptuvenos objektu robežtaisnstūrus;
3. solī tiek noteiktas klašu varbūtības šiem robežtaisnstūriem.

1.5. att. ***YOLO*** **modeļa darbība** [4]

Attēlā 1.6. redzams, ka ieejas bilde tiek sadalīta S x S režģī, kur katra šūna ir atbildīga par kāda objekta noteikšanu, ja šī objekta centrs atrodas konkrētajā šūnā. Katrā šūna tiek izveidoti B robežtaisnstūri un to uzticamības varbūtības. Šīs uzticamības varbūtības raksturo, cik modelis ir pārliecināts, ka robežtaisnstūris satur objektu un cik pārliecināts ir par robežtaisnstūra izmēriem.



1.6. att. ***YOLO*** **objektu klasificēšana un lokalizēšana [4]**

Katrs robežtaisnstūris sastāv no 5 prognozētiem lielumiem - *x*, *y*, *w*, *h* un *p*. (*x*, *y*) koordinātes atspoguļo robežtaisntūra centru režģa šūnu koordinātu sistēmā. (*w*, *h*) raksturo robežtaisntūra platumu un augstumu. (*p*) atspoguļo uzticēšanās varbūtību šim robežtaisnstūrim.

1.3.1 Fast YOLO un YOLO atšķirības

Fast YOLO jeb ātrākais *YOLO* modelis izmanto mazāk konvolūciju slāņus, nekā pilnais YOLO modelis, kā arī šajos slāņos tiek izmantoti mazāk filtri. Slāņu daudzums tiek samazināts no 24 uz 9 slāņiem, kas to padara neprecīzāku, taču pieprasa mazāk skaitļošanas jaudas. [4]

1.4. Pascal VOC datu kopa

Pascal VOC datu kopa ir standarts vizuālu objektu klasifikācijā un atpazīšanā pēc kā tiek vērtēti datorredzes un mašīnmācīšanās risinājumi [5]. Šī datu kopa tiek plaši izmantota mašīnmācīšanās modeļu trenēšanā un vērtēšanā. Jaunākā *Pascal VOC 2012* datu kopa klasifikācijas un atpazīšanas kategorijā iekļauj 20 dažādas datu klases, ieskaitot velosipēdus. *Pascal VOC 2012* un *2011* datu kopas neatšķiras klasifikācijas kategorijā [6], tāpēc, vērtējot iespējamos modeļus darba problēmas risināšanā, tiks izmantoti vērtējumi no abām datu kopām.

1.5. Modeļa izvēle velobraucēju atpazīšanai

Modelis velobraucēju atpazīšanai attēlā tika izvēlēts, balstoties uz trīs kritērijiem - ātrdarbība, precizitāte un iespēja mašīnmācīšanās modeli konvertēt uz *Core ML* formātu.

Modeļa ātrdarbība ir īpaši svarīga, jo mērķis ir velobraucēju atpazīšana reāllaikā, kā arī iOS operētājsistēmas darbināto mobilo ierīču veiktspēja nav salīdzināma ar šādam mērķim izmantojamu servera puses aparatūru.

No veiktspējas ierobežojumiem izriet arī precizitāte, lai atpazīšanai tiktu izmantoti iespējami nelieli skaitļošanas resursi. Vērtējot objektu atpazīšanas metodes, jāņem vērā, ka nepieciešama ne tikai objektu klasifikācija, bet arī objektu lokalizācija. Ņemot vērā augsto vides mainību (velobraucēji kustībā, kas maina pārvietošanās virzienu), svarīgāka ir objektu lokalizācija ar robežtaisntūriem, nevis pilnīga segmentācija un objektu formas iegūšana.

Core ML formāts ir Apple atbalstīts mašīnmācīšanās modeļu formāts, kas padara dažādu trešo pušu modeļus izmantojamus iOS operētājsistēmas darbinātā vidē. Mums interesējošais *Core ML* formāts atbalsta konvolūciju neironu tīklus, kas veidoti "Caffe" vai "Keras" (min. versija 1.2.2.) ietvaros [7].

Tabulā 1.1. tiek salīdzināti potenciāli izmantojami mašīnmācīšanās modeļi objektu klasificēšanai un lokalizēšanai. Visi salīdzinātie modeļi pieejami "Caffe" formātā, kas liecina par savietojamību ar *Core ML* formātu. Problēmas risinājumā tiks izmantots *Fast YOLO* modelis, jo tas uzrāda 155 kadru skaitu sekundē veiktspēju, darbinot to uz grafiskā procesora. Visticamāk, reālie rezultāti uz mobilās ierīces atšķirsies, taču *R-CNN* un *Faster R-CNN* modeļi

nevieš cerības tos izmantot reāllaikā, uz mobilās ierīcēs, to salīdzinoši lēnās darbības dēļ. Turklāt *Fast YOLO* precizitāte būtiski neatpaliek no *R-CNN* risinājumiem, uzrādot 67.2% precizitāte velosipēdu datu klasē.

1.1. Tabula

Objektu atpazīšanas modeļu salīdzinājums

Modeļa nosaukums	**Precizitāte Pascal VOC datu kopā**	**Precizitāte Pascal VOC datu kopā velosipēdu klasē**	**Savietojamība ar Core ML formātu**	**Komentāri**
R-CNN [2]	53.5% [2]	77.0% (VOC 2007 datu kopā) [2]	**Savietojams,** jo pieejams "Caffe" formātā [8]	Aptuvenais darbības ātrums, izmantojot grafisko procesoru – 13 sekundes uz attēlu jeb **0.07 kadri sekundē** [2]
Faster R-CNN [3]	65.7% [3]	74.7% [3]	**Savietojams**, jo pieejams "Caffe" formātā [9]	Aptuvenais darbības ātrums, izmantojot grafisko procesoru – **5 kadri sekundē REF** [3]
Fast YOLO [4]	57.9% [4]	67.2% [4]	**Savietojams**, jo pieejams "Caffe" formātā [10]	Aptuvenais darbības ātrums, izmantojot grafisko procesoru – **155 kadri sekundē** [4]

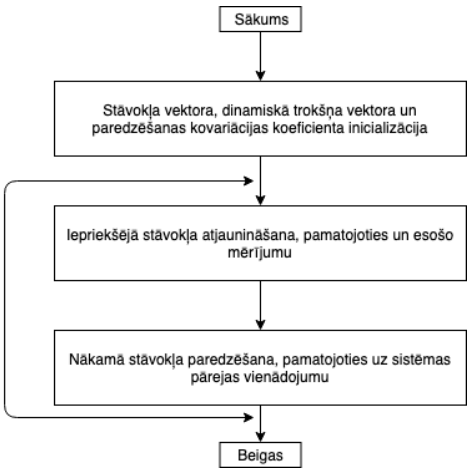
2. OBJEKTU IZSEKOŠANA

Objektu izsekošana videomateriālā ir sarežģīta problēma, jo objekti pārvietojoties maina formu un krāsu, kā arī objektu fons var izmainīties, ja izsekojamo objektu šķērso kāds cits objekts. Šī datorredzes nozare tiek plaši izmantota automātiskajā novērošanā, video materiālu indeksēšanā, žestu atpazīšanā, satiksmes uzraudzībā un transportlīdzekļu automātiskajā vadībā. Sekošanas algoritmu realizācijā bieži tiek pieņemts, ka objektu pārvietošanās ir plūstoša un bez straujām kustībām. [11] Svarīgs faktors objektu izsekošanā ir izsekojamo objektu iepriekšējā atpazīšana, kas aprakstīta darba pirmajā nodaļā.

Objekti var tikt izsekoti pēc dažādām to pazīmēm. Objekti, kas aizņem nelielu laukumu kadrā, var tikt izsekoti pēc punkta, kas atrodas objekta vidū vai vairākiem punktiem, kas piešķirti objekta svarīgām un atpazīstamām detaļām. Primitīvu un maz mainīgu objektu gadījumā, tie var tikt izsekoti pēc to ģeometriskās formas. Izsekošanu pēc objekta silueta vai formas var izmantot, ja objekts ir sarežģītāks un ar mainīgu formu. [11]

2.1. Izsekošana pēc punktu pazīmēm

Izsekošana pēc punktu pazīmēm tiek izmantota gadījumos, kad objekti var mainīt formu, rotāciju vai izmēru, pārvietojoties no viena kadra uz citu. Viena no metodēm, ko izmanto punktu izsekošanā, ir deterministiskā metode [12]. Tajā tiek izmantoti izsekojamā objekta tuvums, pārvietošanās ātrums un kustība kā parametru kopums punkta iespējamās atrašanās vietas aprēķināšanā. Otra izmantotā metode ir statistiskā, kur tiek izmantoti tādi objekta parametri kā atrašanās vieta, ātrums un izmērs.



2.1. att. **Kalman filtra darbība**

Abām metodēm tiek bieži pielietots Kalman filtrs, kas trokšņainā vidē palīdz atrast precīzāku objekta stāvokli, kā arī nodrošina precīzāku nākamā stāvokļa paredzēšanu [13]. Kalman filtra darbība parādīta attēlā 2.1 [14].

2.2. Kodolā bāzēta objektu izsekošana

Kodolā bāzētā objektu izsekošanā (*Kernel based tracking*) objekts tiek izsekots pēc tā ierobežojošās ģeometriskās formas, kas visbiežāk ir taisnstūris vai elipse. Forma, pārvietojoties no kadra uz kadru, var mainīties pēc izmēra, proporcijām, rotācijas un citām ģeometriskām īpašībām [13].

Objekta līdzības vērtību aprēķināšanā tiek izmantota histogramma un tās intensitātes vērtības. Lai uzlabotu precizitāti kadros ar sarežģītu fonu, tiek izmantota lineāra atbalsta vektoru mašīna (*SVM*) [12].

Ņemot vērā, ka ierobežojošā figūra var ietvert objekta fonu vai kāda no objekta daļām var atrasties ārpus figūrās, šajā sekošanā var rasties neprecizitātes, objektam pārvietojoties [15].

2.3. Izsekošana pēc objekta silueta

Izsekošana pēc objekta silueta tiek izmantota objektu ar sarežģītu formu izsekošanā. Viens no metodes paveidiem ir izsekošana pēc kontūras (*Contour tracking*). Izmantojot šo metodi, priekšnosacījums ir, lai divos blakus esošos kadros objekta reģioni pārklātos. Otra izmantotā metode ir objekta formu salīdzināšana (*Shape Matching*), kurā tiek noņemts objekta fons un forma tiek salīdzināta no kadra uz kadru [15].

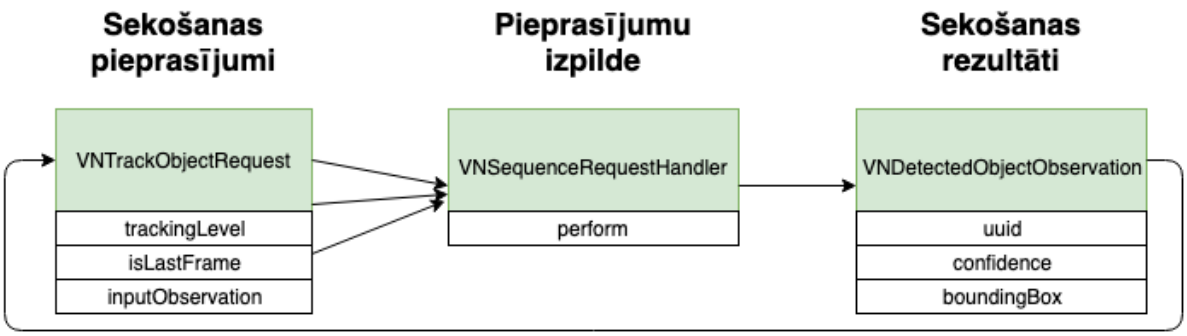
2.4. iOS Vision izsekošana

Attēlā 2.2. attēlots *iOS Vision* objektu izsekošanas dzīves cikls. Šis dzīves cikls var tikt pārtraukts, ja tiek izskatīts pēdējais video kadrs vai ja ir zināms, ka nākamajā kadrā neeksistē neviens izsekojams objekts. *iOS Vision* objektu izsekošana tiek veikta līdzīgi kā kodolā bāzētā objektu izsekošana (*Kernel based tracking*), jo par ieejas parametru tiek izmantots ierobežojošs taisnstūris.

Pirmajā solī tiek izveidots viens *VNTrackObjectRequest* sekošanas pieprasījums katram atrastajam objektam. Parametros iespējams norādīt izsekošanas precizitāti (*trackingLevel*), pazīmi, vai konkrētais objekts tiks izsekots arī nākamajā video kadrā, kā arī ieejas novērojuma rezultātu (*inputObservation*), kas iekļauj izsekojamā objekta atrašanās vietu kadrējuma attēlā. Ieejas novērojuma rezultāts var sastāvēt no atpazīta objekta, kurš iepriekšējā kadrā netika izsekots vai arī objekta, kurš tika izsekots iepriekšējā iterācijā un rezultējās veiksmīgā novērojumā. Izsekošanas dzīves cikla laikā šie pieprasījumi var tik papildināti ar jauniem vai izņemti tie pieprasījumi, kas vairs nav atrodami kadrējumā. [16]

Otrajā solī tiek izpildīti izveidotie pieprasījumi ar klases *VNSequenceRequestHandler* metodi, kas akceptē vairākus sekošanas pieprasījumus un kārtējo video kadrējuma attēlu. [16]

Trešajā solī tiek iegūti sekošanas rezultāti izveidotajiem pieprasījumiem, ja tie bija veiksmīgi. *VNDetectedObjectObservation* klases rezultāts katram pieprasījumam satur tā unikālo identifikatoru (uuid), lai objekti varētu tikt identificēti visā sekošanas laikā, pieprasījuma rezultāta uzticamības vērtējumu (*confidence*), kā arī izsekojamā objekta robežtaisntūri attēlā. [16]



2.2. att. **Vision objektu izsekošanas dzīves cikls**

2.4.1 VNTrackObjectRequest klases apraksts

Šīs klases objekts tiek inicializēts ar *VNDetectedObjectObservation* klases objektu, kas tiek iegūts no iepriekšējā sekošanas pieprasījuma rezultāta vai tiek izveidots, ja objekts netika izsekots, bet ir zināms tā robežtaisnstūris. [17]

VNTrackObjectRequest objektam nepieciešams norādīt izsekošanas precizitātes līmeni - ātru (*VNRequestTrackingLevel.fast*) vai precīzu (*VNRequestTrackingLevel.accurate*). Dokumentācijā nav minēts veiktspējas un precizitātes salīdzinājums šiem līmeņiem, tādēļ nepieciešams veikt mērījumus, lai atrastu optimālu risinājumu problēmai.

Ja parametrs *isLastFrame* tiek norādīts kā patiess, tad konkrētais objekts vairs netiks izsekots un tiks atbrīvoti resursi citiem izsekošanas pieprasījumiem. [18]

Norādot *VNTrackObjectRequest.usesCPUOnly* parametru kā patiesu, pieprasījums tiks izpildīts uz centrālā procesora bez iespējas to izpildīt uz grafiskā procesora. Tas varētu noderēt situācijās, kad grafiskā procesora resursi nepieciešami lietotāja saskarnei un izsekošanas veiktspēja ir pietiekama, izmantojot tikai centrālo procesoru. Tomēr, ņemot vērā, ka Vision pieprasījumi strādā ar neironu tīklu palīdzību, efektīvāka pieprasījumu apstrāde gaidāma, izmantojot grafisko procesoru. [19]

2.4.2 VNSequenceRequestHandler klases apraksts

Pieprasījumu izpildes objekts tiek inicializēts bez parametriem, bet tā *perform* metode akceptē masīvu ar *VNTrackObjectRequest* klases objektiem un kārtējo video kadru kādā no sekojošiem formātiem: *CVPixelBuffer, CGImage, CIImage, Data*. Metode *perform* atgriež patiesu vērtību, ja visi pieprasījumi tika ieplānoti un izpildīti vai tiek izsaukta izņēmumsituācija, ja pieprasījumus nebija iespējams apstrādāt sistēmas kļūdas gadījumā. [20]

Arī pieprasījumu izpildes klasi ir svarīgi neatbrīvot no atmiņas, jo tā satur informāciju par iepriekšējiem sekošanas pieprasījumiem un saglabā kešatmiņā kadru informāciju. [21]

2.4.3 VNDetectedObjectObservation klases apraksts

Veiksmīgi izpildīta pieprasījuma gadījumā iepriekš izmantotais *VNTrackObjectRequest* klases objekts satur vienu vai vairākus *VNDetectedObjectObservation* objektus, kurā atrodama novērojuma uzticamība, kā arī novērojuma robežtaisnstūris. Ja objekts kadrējumā tika atrasts, šo *VNDetectedObjectObservation* objektu var izmantot atkārtoti nākamā sekošanas pieprasījuma inicializēšanai, lai turpinātu objekta izsekošanu nākamajā kadrējumā. [22]

2.4.4 Izsekošanas ierobežojumi un potenciālās problēmas

Viens no ierobežojumiem, kas nav atrodams Apple dokumentācijā, bet tiek pieminēts 2018. gada Apple WWDC konferences laikā, ir maksimālais izsekojamo objektu skaits vienā pieprasījumā, kas ir 16 [21]. Ar šo problēmu varētu saskarties, ja novērojumi tiek veikti augstas satiksmes intensitātes ielas posmā. Izsekošanas pieprasījumus var atbrīvot, tiem norādot *lastFrame* parametru kā patiesu vai atbrīvojot tos no atmiņas.

Otrs potenciāls ierobežojums ir precizitāte izsekojot velobraucēju brīdī, kad tiek veikts pagrieziens un attiecībā pret kameru mainās velobraucēja forma un izskats. Kā potenciālais risinājums ir veikt objektu meklēšanu kadrējumā atkārtoti un sākt konkrētā velobraucēja izsekošanu no jauna, ja uzticamības vērtējums pieprasījuma apstrādē ir zems.

Treškārt, velobraucēja izsekošanu var pārtraukt dažādi objekti, kas parādās starp kameru un izsekojamo velobraucēju, piemēram, citi satiksmes dalībnieki. Šo problēmu potenciāli varētu risināt ar iepriekšējā punktā minēto atkārtotu meklēšanu kadrējumā. Turklāt video jāaptver viss apskatāmais reģions, citādi izsekojamie velobraucēji pazudīs no redzamības un atkal parādīsies.

Atkarībā no objekta atrašanas un lokalizēšanas metožu precizitātes, var mainīties izsekošanas rezultāti, ja sākotnējais robežtaisnstūris nav pietiekami precīzs vai arī iekļauj pārāk daudz fona objektu.

3. MOBILĀS LIETOTNES DARBĪBA

Objektu atpazīšanai un lokalizēšanai video materiāla kadros tika izvēlēts YOLO modelis, kas aprakstīts darba pirmajā nodaļā. Izsekošanai tika izvēlēts iOS iebūvēts risinājums – *Vision* ietvars, kas plašāk aprakstīts darba otrajā nodaļā.

Nodaļā tiks aprakstītas prasības attiecībā uz sagaidāmo rezultātu iOS lietotnes darbībā, kā arī tiks aprakstīti veiktspējas un precizitātes mērījumi, kombinējot izvēlētos tehniskos risinājumus.

3.1. Trajektoriju sagatavošanas prasības

Attēlā 3.1. redzams līdzšinējais rezultāts trajektoriju atzīmēšanā ielas attēlā, ko apvienības "Pilsēta Cilvēkiem" dalībnieki veic manuāli. Izvirzītās prasības pret lietotnes atzīmētajām trajektorijām ir līdzīgas – tās ir aptuvenas, tomēr norāda galvenās velobraucēju pārvietošanās trajektorijas. Šajā konkrētajā gadījumā par fonu tiek izmantots ekrānšāviņš no *Google* ielas attēla.

3.1. att. **Ielas attēlā atzīmētas velo braucēju pārvietošanās trajektorijas**

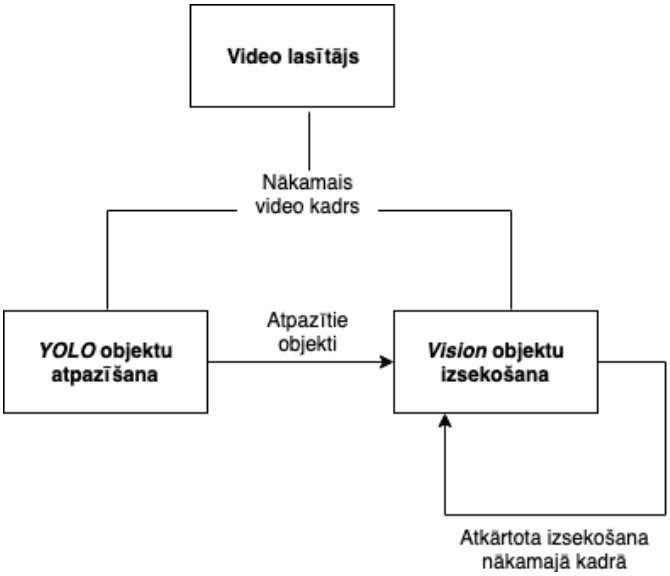
3.2. Lietotnes darbības apraksts

Izstrādātās lietotnes darbība parādīta 3.2. attēlā, kas sastāv no trīs galvenajiem moduļiem – video lasītāja, *YOLO* objektu atpazīšanas un *Vision* objektu izsekošanas.

Video lasītāja modulis tiek izmantots, lai iegūtu katru nākamo video kadrējumu, kas tiek izmantoti gan objektu atpazīšanā, gan izsekošanā. Šī moduļa darbībā netiek veikta video

atskaņošana reāllaikā, kas nozīmē, ka nākamais kadrs tiek pieprasīts tikai tad, kad iepriekšējais ir apstrādāts.

Lai iegūtu izsekojamos robežtaisnstūrus, tiek izmantots *YOLO* objektu atpazīšanas modulis. Iegūtie robežtaisnstūri tālāk tiek izsekoti *Vision* objektu izsekošanas modulī.



3.2. att. **Objektu atrašana un izsekošana iOS lietotnē**

Lietotnes darbības laikā tika novērota ievērojama aizture kadra attēlošanā un trajektoriju zīmēšanā, jo *Vision* un *YOLO* darbība aizņem nozīmīgu laiku. Tika izdarīts secinājums, ka atpazīšana un izsekošana jāveic citos pavedienos kā lietotnes galvenajā pavedienā, tādēļ šo darbību izpildei tika izveidota atsevišķi *DispatchQueue* pavedieni, kuru izpilde nebloķē galveno pavedienu un tā vizuālo atjauninājumu izpildi.

3.2.1 Ieejas datu apstrāde

Gan *Vision* izsekošana, gan *YOLO* ieejā akceptē *CVPixelBuffer* klases instanci, kas ir *Core Video* bibliotēkas pikseļu buferis, kas glabā šo buferi galvenajā atmiņā [23]. Tomēr, kā iepriekš minēts, 1.3. apakšnodaļā, *YOLO* pieprasa pikseļu bufera pārveidi uz kvadrātveida izmēru. Tas tiek panākts ar *CIContext* klases *render* metodi, kas pārveido attēlu vajadzīgajā 448x448px formātā.

3.2.2 *YOLO* izejas datu apstrādē

YOLO klases *prediction* metode atgriež *Core ML* bibliotēkas *MLMultiArray* daudzdimensionāla masīva objektu, kas raksturo mašīnmācīšanās modeļu ieejas vai izejas pazīmes [24]. *YOLO* izmantošanas gadījumā, tas ir 125x13x13 tenzors, kur 13x13 ir režģa izmērs un 125 ir katras režģa šūnas pazīmes. Pazīmēs sastāv no maksimums 5 robežtaisntūriem

un katru robežtaisntūri veido 25 vērtības, kas sastāv no 20 atpazīstamām klasēm (mums interesē tikai klase ar indeksu 1, kas ir velosipēds), kā arī x, y, platuma, augstuma koordinātes attiecībā pret režģa šūnas robežām, kā arī pārliecinātības varbūtība, ka robežtaisnstūris satur meklējamo objektu.

x un y koordinātes var tikt aprēķinātas pēc formulas

$$S * \left(\frac{1}{1+ e^{-t}}\right) * \frac{448}{13},$$

kur *S* ir režģa šūnas x vai y kārtas numurs, *t* ir x vai y robežtaisnstūra režģa šūnas kārtas numurus (0-12), bet 448/13 apzīmē režģa šūnas platumu vai augstumu ieejas attiecībā pret ieejas attēlu (šajā gadījumā katras režģa šūnas platums un augstums ir vienādi). Loģistikas funkcija [25] $\left(\frac{1}{1+ e^{-t}}\right)$ tiek izmantota, lai iegūtu koordinātes 0-1 diapazonā.

Robežtaisnstūru īstā platuma un augstuma aprēķināšanai tika izmantotas *YOLO* 2. versijas konfigurācijas enkuru vērtības [26]. Enkuru vērtības ir *YOLO* definētas vērtībās, kas apzīmē augstuma un platuma attiecību 10 dažādiem robežtaisnstūriem. Augstuma un platuma aprēķināšana ieejas attēla koordinātu sistēmā tiek veikta pēc formulas

$$e^{q} * anchor(b) * \frac{448}{13},$$

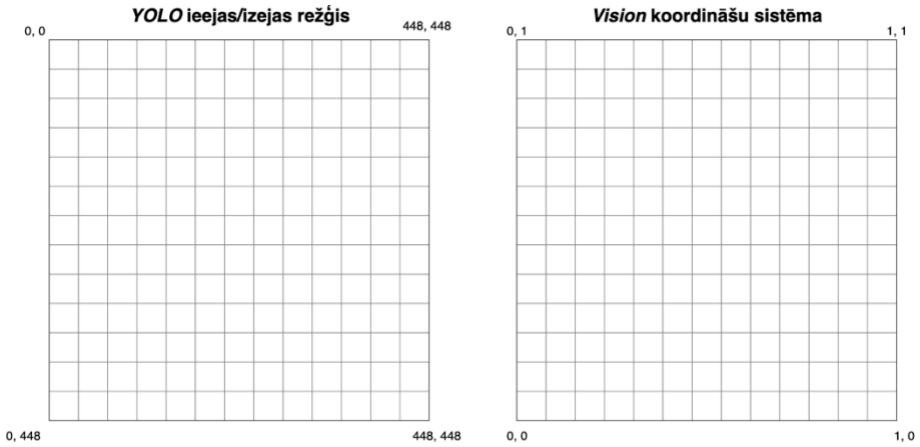
kur q ir robežtaisnstūra augstums vai platums relatīvi pret kādu no enkura robežtaisnstūriem. Funkcija *anchor* atgriež enkura vērtību *b* robežtaisnstūrim, bet 448/13 apzīmē jau iepriekšminēto režģa šūnas platumu vai augstumu.

Pārliecinātības varbūtība tiek aprēķināta procentuālā vērtībā pēc iepriekš minētās loģistikas funkcijas.

Rezultātā tiek iegūti robežtaisnstūri, kas, iespējams, daļēji pārklājās. Lai izslēgtu šos gadījumus, šeit var pielietot *Tensorflow* metodes *NonMaxSuppression* implementāciju, kas īsumā notiek šādos soļos:

I. Robežtaisnstūri tiek sakārtoti no lielākās pārliecinātības varbūtības uz mazāko
II. Tiek izvēlēts kārtējais robežtaisnstūris ar nākamo vislielāko pārliecinātības varbūtību
    a. Visi pārējie robežtaisntūri, ar kuriem šķēlums pār apvienojumu (*IOU – intersection over union*) ir lielāks par mūsu vēlamo slieksni, tiek izņemti no sakārtotā saraksta
    b. Atkārto soli II līdz tiek sasniegts sakārtotā saraksta pēdējais elements

Iegūtie robežtaisnstūri ir definēti iepriekš minētajās 448x448px koordinātēs, tāpēc tos pirms izsekošanas nepieciešams pārveidot *Vision* pieprasītajā koordinātu sistēmā, kas parādīta 3.3. attēlā.



3.3. att. ***YOLO* ieejas/izejas režģis un *Vision* koordinātu sistēma**

Ja robežtaisnstūris krusto kādu no tajā brīdī izsekojamajiem objektiem, iegūtais robežtaisnstūris netiek ņemts vērā. Šāda rodas, ja divi potenciāli izsekojamie objekti atrodas pārāk tuvu vai attēlā tiek atrasts objekts, kas jau tiek izsekots.

3.2.3 *Vision* izejas datu apstrāde

Tāpat kā ieejas pikseļu buferī, arī robežtaisnstūra izejas datu koordinātes ir normalizētā formā 0-1, ar koordinātu sākuma punktu kreisajā apakšējā stūrī. Otrs mums svarīgais parametrs ir pārliecinātības varbūtība normalizētā formā (0-1). Ja varbūtība, ka objekts joprojām atrodas iegūtajā robežtaisnstūrī, ir mazākā par 0.3, *VNDetectedObjectObservation* rezultāta instance netiks uzskatīta par ticamu izsekošanai nākamajā video kadrā, un ir nepieciešams veikt atkārtotu objekta atpazīšanu.

3.2.4 Trajektoriju atzīmēšana ielas attēlā

Ņemot vērā prasības, kas aprakstītas 3.1. apakšnodaļā, fons trajektoriju atzīmēšanai tiek iegūts no video materiāla, kas tiek izmantots trajektoriju zīmēšanai.

Trajektorijas tiek zīmētas slānī virs kadra attēla skata, izmantojot *CAShapeLayer* klasi trajektorijas attēlošanai un *UIBezierPath* klasi trajektorijas izveidošanai, savienojot punktus, kas tiek iegūti no izsekošanas laikā iegūto objekta robežtaisnstūru apakšējās kreisās virsotnes. Jāņem vērā, ka *CAShapeLayer* attēlošana skatā ir jāveic galvenajā sistēmas pavedienā (*Main thread*).

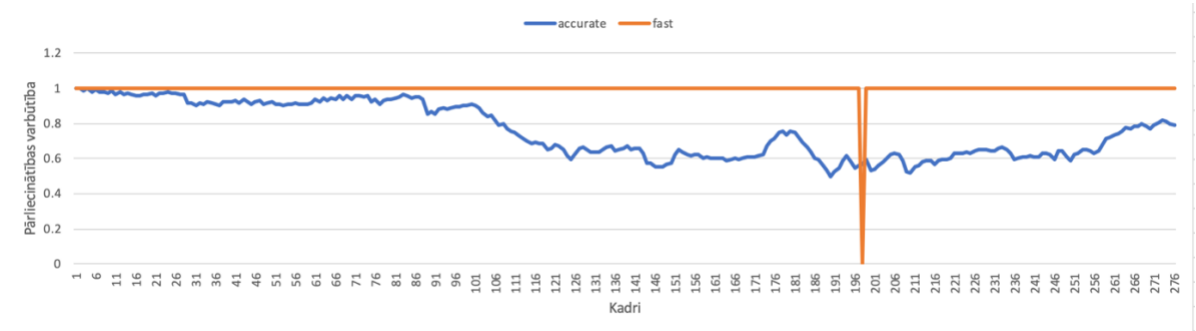
3.3. Darbības novērtējums

Veiktspējas mērījumos tika izmainīti dažādi parametri, kas būtiski ietekmē lietotnes darbību. Visi mērījumi tika veikti, izmantojot iPhone X mobilo ierīci, uz kuras uzstādīta iOS 12.1.4 operētājsistēma. Kā testpiemērs tika ņemts 276 kadru garš videomateriāls, kas redzams attēlā 3.4., kur velobraucējs ir labi redzams visos kadros, bet pakāpeniski virzās uz kreiso attēla pusi. Video garums ir 9,2s, kadru skaits sekundē – 30, kopējais kadru skaits – 275.

3.4. att. **Mērījumos izmantotā video materiāla kadri**

3.3.1 Izsekošanas precizitāte, mainot izsekošanas līmeni

Attēlā 3.5. tiek salīdzināta precīzā (*accurate*) un ātrā (*fast*) izsekošanas metode pēc prognozētajām pārliecinātības varbūtībām. Abas metodes paredzēja robežtaisnstūra koordinātes vienlīdz precīzi, taču pārsteidzoši, ka ātrais izsekošanas līmenis nedod precīzas pārliecinātības varbūtības robežtaisnstūriem, bet parāda tikai divas diskrētas vērtības – 0.0 vai 1.0.



3.5. att. **Ātrās un precīzās sekošanas pārliecinātības varbūtību salīdzinājums**

3.3.2 Atpazīšanas un izsekošanas precizitātes vizuāls salīdzinājums

Tika veikts eksperiments, kā rezultātā tika iegūtas divas trajektorijas (skatīt 3.6. attēlu) – sarkanā trajektorija apzīmē no *YOLO* atpazīšanas iegūtos rezultātus, bet zaļā trajektorija apzīmē no Vision izsekošanas iegūtos rezultātus.

Redzams, ka izsekošana notiek daudz plūstošāk par atpazīšanu. Turklāt, skatoties kreisajā attēla pusē, atpazīšanas modulis nespēj atpazīt velobraucēju, jo tas ir attālinājies, kamēr izsekošana spēja izsekot objektu līdz pat brīdim, kad velosipēdists vairs nav redzams kadrā. Darba 1.5. apakšnodaļā tika aprakstīta *YOLO* vidējā precizitāte, kas apstiprina praktiskajā rezultātā iegūto neprecizitāti un trajektorijas atšķirības. Jāpiemin, ka izsekošanas daļā, pirmajā kadrā ieejas robežtaisnstūris tika iegūts no atpazīšanas moduļa.

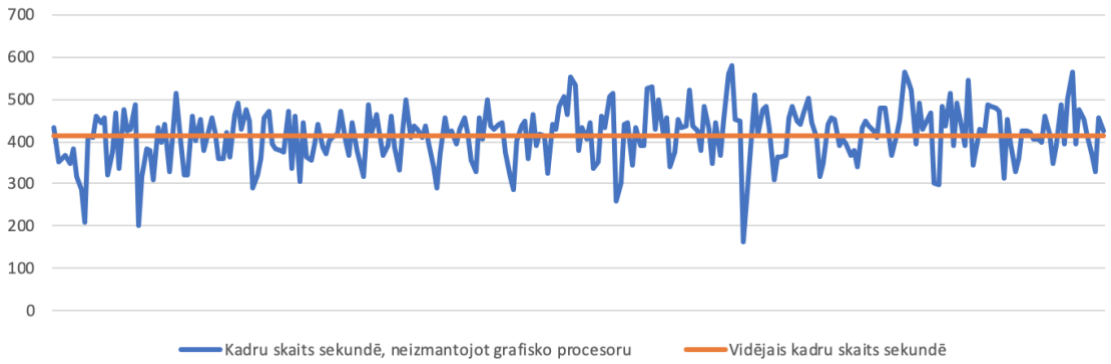
3.6. attēlā redzama arī ievērojamais attālums no velo braucēja līdz zaļajai trajektorijai, kas liecina par to, ka robežtaisntūris nav pietiekami precīzs (iekļauj daudz fona, kas ir pieminēts darba otrās nodaļas apakšnodaļā 2.2.).

3.6. att. ***Yolo* precizitāte (sarkanā trajektorijas) un *Vision* precizitāte (zaļā trajektorija)**

3.3.3 Izsekošanas veiktspēja

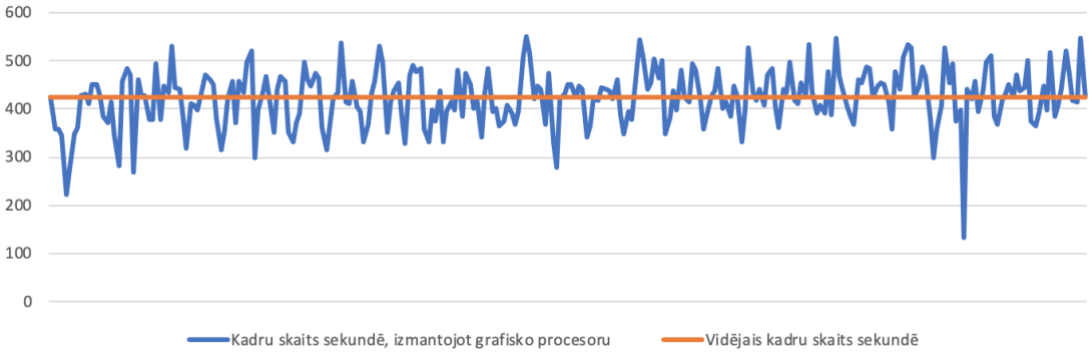
Lai veiktu izsekošanas veiktspējas mērījumus, ar *YOLO* tika atrasts objekts pirmajā video kadrā, bet turpmākos kadrus objektu meklēšana netika veikta. Mērījumu vērtības tika saglabātas pēc katra apstrādātā kadra.

Izmantojot tikai centrālo procesoru, *Vision* izsekošana uzrādīja vidējo rezultātu 413 kadri sekundē, minimālo rezultātu 162 kadri sekundē, augstāko rezultātu 578 kadri sekundē. Grafiks redzams attēlā 3.7.



3.7. att. ***Vision*** **izsekošanas kadru skaits sekundē, neizmantojot grafisko procesoru**

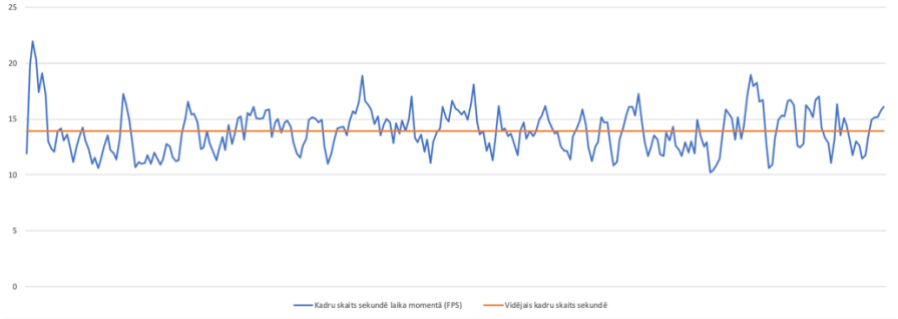
Izmantojot parametru *usesCPUOnly* ar vērtību nepatiess, kas norāda, ka sistēma var izmantot arī grafisko procesoru izsekošanas pieprasījumu veikšanai, vidējais rezultāts bija 422 kadri sekundē, kas ir nedaudz augstāks rādītājs, kas tika sasniegts, izmantojot tikai centrālo procesoru, taču tika iegūts zemāks maksimālais rezultāts – 548 kadri sekundē, kā arī zemāks minimālais rezultāts – 131 kadrs sekundē. Grafiks redzams attēlā 3.8.



3.8. att. ***Vision*** **izsekošanas kadru skaits sekundē, izmantojot grafisko procesoru**

3.3.4 *YOLO* atpazīšanas veiktspēja

YOLO atpazīšanas veiktspēja tika mērīta līdzīgi kā izsekošanas veiktspēja – mērījumi tika veikti pēc katra apstrādātā kadra. Vidējais atpazīšanas ātrums bija 13 kadri sekundē, minimālais – 10 kadri sekundē, maksimālais – 21 kadrs sekundē. Atpazīšanas laikā netika veikti izsekošanas pieprasījumi. Atpazīšanas veiktspējas grafiks redzams 3.9. attēlā.



3.9. att. ***YOLO*** **atpazīšanas kadru skaits sekundē**

3.3.5 Novērotās problēmas izsekošanas laikā

Viena no visbiežāk sastopamajām problēmām bija sekošanas pārtraukumi un nesekmīga sekošanas turpināšanās, kad izsekojamo objektu aizsedza citi satiksmes dalībnieki, piemēram, gājēji vai automašīnās. Attēlā 3.10. redzams kā velosipēdu vidējā kadrā aizsedz gājēji un tiek būtiski zaudēta pārliecinātības varbūtība (apzīmēta ar raustītu robežtaisnstūri), bet tā nav 0, kā rezultātā gājēji tiek atpazīti kā velo braucējs. Dažkārt šāda situācijā gadās, ja aizsedzošais objekts ir kustīgs – izsekošana tiek zaudēta oriģinālajam izsekotajam objektam, bet tā tiek turpināta citam objektam, kas ne vienmēr ir velosipēds.

3.10. att. **Velo braucēja izsekošana, kad tas tiek aizsegts**

3.4. Iespējamie lietotnes uzlabojumi

- Salīdzinājumā ar objektu izsekošanu, objektu atrašana kadrā ir lēnāka un nespēj strādāt reāllaikā. Viens no risinājumiem, kas patērētu vairāk sistēmas resursus, taču potenciāli izslēgtu sekošanas gaidīšanu uz iepriekšēju objektu atrašanu, būtu veikt objektu atrašanu paralēli un vairākus kadrus priekšā sekošanas pieprasījumiem;
- Šobrīd objektu meklēšana tiek veikta katrus ik pēc 10 kadriem kadrus. Izsekošanas precizitāte varētu tikt uzlabota, ja objektu meklēšana tiktu veikta nekavējoties, kad izsekojamā objekta robežtaisnstūris zaudē pārliecinātības varbūtību;
- Pret trajektoriju iezīmēšanu krustojuma attēlā netika noteiktas stingras prasības, tādēļ tika izstrādāts vienkārši īstenojams risinājums, kurā netiek ņemts vērā attēla telpiskais dziļums. Tomēr, ņemot vērā rezultātā redzamās atpazīšanas un izsekošanas neprecizitātes, trajektorija dažkārt atrodas tālu no velo braucēja. Šādus telpiskus datus par attēlu varētu iegūt, izmantojot iPhone ierīces ar divām kamerām un *AVDepthData* klasi [27]. Jāņem vērā, ka šāds risinājums nebūtu pieejams iepriekš ierakstītam videomateriālam, kas nav uzņemts ar atbilstošu ierīci;
- Lietotnes darbības laikā tās lietotājam tiek parādīti visi izsekotie objekti ar to ierobežojošajiem robežtaisnstūriem. Iespējams, tiktu iegūti papildus skaitļošanas resursi objektu atpazīšanai, ja šādus datus neattēlotu reāllaikā, bet lietotājam tiktu parādītas tikai rezultējošās trajektorijas;

- Būtu nepieciešams pārliecināties par video statisku uzņemšanu, jo trajektoriju zīmēšanas daļa paļaujas uz fona nemainību, kuru nodrošināt šobrīd ir ierīces lietotāja uzdevums.

REZULTĀTI

Darbā veiksmīgi tika izstrādāta mobilā lietotne velo braucēju atpazīšanai un izsekošanai. Hipotēze, ka šāda sistēma spēj strādāt reāllaikā uz mobilās ierīcēs, kombinējot objektu atpazīšanu un izsekošanu, apstiprinājās.

Uzrādītie izsekošanas veiktspējas rezultāti uzrāda, ka izsekošanu ir iespējams veikt ne tikai reāllaikā, bet arī daudz īsākā laikā pie noteikuma, ka tiek apstrādāts iepriekš uzņemts video materiāls, nevis kadru iegūšana tiek veikta no ierīces kameras. Ņemot vērā iegūto vidējo 422 kadru sekundē izsekošanas veiktspēju, video materiālu ar 30 kadru sekundē ātrumu varētu apstrādāt 14 reižu lielākā paātrinājumā, salīdzinot ar reāllaiku.

Objektu atpazīšanas rezultāti uzrādīja daudz zemāku veiktspēju, nekā izsekošana. Salīdzinot labāko iegūto rezultātu – 21 kadrs sekundē, veiktspēja ir ievērojami zemāka par minētajiem 155 kadri sekundē uz *Titan X* grafiskā procesora [4]. Tomēr šajā pētījumā iegūtais rezultāts ir augstāks par iegūtajiem 11 kadriem sekundē, izmantojot iPhone 7 ierīci ar operētājsistēmu iOS 10 līdzīgā pētījumā [28]. Veicot objektu atpazīšanu pēc nepieciešamības un izlaižot noteiktu kadru skaitu starp katru atpazīšanu, iespējams panākt veiktspēju, kas ir daudz tuvāka reāllaikam.

Veiktās trajektorijas tika veiksmīgi iezīmētas ielas attēlā, bet tās netika normalizētas un reducētas, kas būtu nepieciešams pie augstas velo braucēju satiksmes intensitātes un ilgāka video uzņemšanas laika. Turklāt, trajektoriju precizitāte ir ļoti atkarīga no robežtaisnstūriem, kas iegūti atpazīšanas un izsekošanas laikā, kas ne vienmēr deva precīzus rezultātus.

SECINĀJUMI

Ņemot vērā iegūto rezultātu, var secināt, ka šādai sistēmai ir potenciāls darboties reālos apstākļos un reāllaikā, kas būtiski samazinātu cilvēka ieguldīto laiku trajektoriju pārzīmēšanā manuāli. Turklāt šāda sistēma varētu būt īpaši pievilcīga bezpeļņas organizācijām, jo objektu atpazīšana un izsekošana notiek tikai uz mobilās ierīcēs. Šādam risinājumam nav nepieciešams interneta savienojums un servera puses darbība, kas arī nodrošina principā neierobežotu mērogojamību, neieguldot papildus resursus.

Objektu atpazīšanā arī tika novērotas kļūdas, tāpēc risinājumā izmantoto konvolūciju neironu tīklu varētu trenēt, izmantojot jau ievāktos videomateriālus, kur redzami velo braucēji.

Trūkums iOS Vision izsekošanai ir maksimālais izsekojamo objektu skaits laika momentā, kas ir 16 objekti. Ļoti intensīvas velo satiksmes apstākļos, šis ir vērā ņemams ierobežojums. Potenciālais risinājums būtu veikt izsekošanu atkārtoti tajos kadros, kur atpazīto objektu daudzums pārsniedz 16 objektus. Šāds risinājums, iespējams, varētu strādāt reāllaikā, jo izsekošana uzrādīja augstu veiktspējas rezultātu.

Viens no turpmākajiem pētījuma virzieniem ir attēla dziļuma noteikšana un trajektoriju iezīmēšana telpiskā attēlā. Turklāt trajektorijas realizētajā risinājumā pārklājas un mēdz būt neprecīzas, tāpēc būtu jāīsteno algoritms, kas reducē pārklājošās trajektorijas un izslēdz to anomālijas neprecīzi veiktas izsekošanas rezultātā.

PATEICĪBAS

Darba autors izsaka pateicību darba vadītājam Oskaram Ozolam par ieguldījumu darba vadīšanā un regulāru palīdzību darba attīstībā.

Pateicību vēlos izteikt arī apvienībai “Pilsēta Cilvēkiem” par problēmas aktualizēšanu un praktiskiem ieteikumiem pētījuma attīstībā. Īpaši vēlos pateikties apvienības dalībniekam Andrejam Oliņam par video materiālu sagatavošanu Rīgas pilsētas krustojumos, kas bija ļoti vērtīgs materiāls, analizējot mobilās lietotnes darbību reālos apstākļos.

IZMANTOTĀ LITERATŪRA

[1] P. Sharma, «A Step-by-Step Introduction to the Basic Object Detection Algorithms (Part 1),» 2018. [Tiešsaiste]. Pieejams: https://www.analyticsvidhya.com/blog/2018/10/a-step-by-step-introduction-to-the-basic-object-detection-algorithms-part-1/. [Piekļūts 04.03.2019].

[2] J. D. T. D. J. M. Ross Girshick, «Rich feature hierarchies for accurate object detection and semantic segmentation,» 2014. [Tiešsaiste]. Pieejams: https://arxiv.org/pdf/1311.2524.pdf. [Piekļūts 04.03.2019].

[3] K. H. R. G. J. S. Shaoqing Ren, «Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,» 2016. [Tiešsaiste]. Pieejams: https://arxiv.org/pdf/1506.01497.pdf. [Piekļūts 07.03.2019].

[4] S. D. R. G. A. F. Joseph Redmon, «You Only Look Once: Unified, Real-Time Object Detection,» 2016. [Tiešsaiste]. Pieejams: https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Redmon_You_Only_Look_CVPR_2016_paper.pdf. [Piekļūts 13.04.2019].

[5] L. V. G. C. K. I. W. J. W. A. Z. Mark Everingham, «The PASCAL Visual Object Classes (VOC) Challenge,» 2009. [Tiešsaiste]. Pieejams: https://link.springer.com/content/pdf/10.1007%2Fs11263-009-0275-4.pdf. [Piekļūts 13.04.2019].

[6] «Visual Object Classes Challenge 2012 (VOC2012),» 2012. [Tiešsaiste]. Pieejams: http://host.robots.ox.ac.uk/pascal/VOC/voc2012/index.html. [Piekļūts 13.04.2019].

[7] «Converting Trained Models to Core ML,» 2019. [Tiešsaiste]. Pieejams: https://developer.apple.com/documentation/coreml/converting_trained_models_to_core_ml. [Piekļūts 25.04.2019].

[8] R. Girshick, «R-CNN: Regions with Convolutional Neural Network Features,» 2017. [Tiešsaiste]. Pieejams: https://github.com/rbgirshick/rcnn. [Piekļūts 27.04.2019].

[9] R. Girshick, «Faster R-CNN (Python implementation),» 2018. [Tiešsaiste]. Pieejams: https://github.com/rbgirshick/py-faster-rcnn. [Piekļūts 27.04.2019].

[10] «ImageNet Classification,» [Tiešsaiste]. Pieejams: https://pjreddie.com/darknet/imagenet/#reference. [Piekļūts 27.04.2019].

[11] O. J. M. S. Alper Yilmaz, «Object Tracking: A Survey» 2006. [Tiešsaiste]. Pieejams: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.112.8588&rep=rep1&type=pdf. [Piekļūts 14.04.2019].

[12] H. S. C. K. M. Y. J. Y. C. S. G. K. In Su Kim, «Intelligent Visual Surveillance - A Survey,» 2010. [Tiešsaiste]. Pieejams: https://www.researchgate.net/publication/225179218_Intelligent_Visual_Surveillance_-_A_Survey. [Piekļūts 15.05.2019].

[13] A. M. Sandeep Kumar Patel, «Moving Object Tracking Techniques: A Critical Review,» 2013. [Tiešsaiste]. Pieejams: https://pdfs.semanticscholar.org/be93/0db4fb72970254ecb74e7a13aafaca912802.pdf. [Piekļūts 15.05.2019].

[14] K. P. Salil P. Banerjee, «Multiperson Tracking Using Kalman Filter,» 2008. [Tiešsaiste]. Pieejams: http://cecas.clemson.edu/~stb/ece847/projects/Multiperson_Track_KF.pdf. [Piekļūts 13.05.2019].

[15] D. G. T. U. K. J. Himani S. Parekh, «A Survey on Object Detection and Tracking Methods,» 2014. [Tiešsaiste]. Pieejams: https://pdfs.semanticscholar.org/25a6/c5dff9a7019475daa81cd5a7f1f2dcdb5cf1.pdf. [Piekļūts 03.04.2019].

[16] «Vision,» [Tiešsaiste]. Pieejams: https://developer.apple.com/documentation/vision. [Piekļūts 17.03.2019].

[17] «VNDetectedObjectObservation,» [Tiešsaiste]. Pieejams: https://developer.apple.com/documentation/vision/vndetectedobjectobservation. [Piekļūts 17.04.2019].

[18] «isLastFrame - VNTrackingRequest,» [Tiešsaiste]. Pieejams: https://developer.apple.com/documentation/vision/vntrackingrequest/2887356-islastframe. [Piekļūts 17.04.2019].

[19] «usesCPUOnly - VNRequest,» [Tiešsaiste]. Pieejams: https://developer.apple.com/documentation/vision/vnrequest/2923480-usescpuonly. [Piekļūts 17.04.2019].

[20] «perform - VNSequenceRequestHandler,» [Tiešsaiste]. Pieejams: https://developer.apple.com/documentation/vision/vnsequencerequesthandler/2880300-perform. [Piekļūts 17.04.2019].

[21] «Object Tracking in Vision,» [Tiešsaiste]. Pieejams: https://developer.apple.com/videos/play/wwdc2018/716 . [Piekļūts 23.04.2019].

[22] «VNDetectedObjectObservation,» [Tiešsaiste]. Pieejams: https://developer.apple.com/documentation/vision/vndetectedobjectobservation. [Piekļūts 24.04.2019].

[23] «CVPixelBuffer - Core Video,» [Tiešsaiste]. Pieejams: https://developer.apple.com/documentation/corevideo/cvpixelbuffer-q2e. [Piekļūts 14.05.2019].

[24] «MLMultiArray - MLFeatureValue,» [Tiešsaiste]. Pieejams: https://developer.apple.com/documentation/coreml/mlmultiarray. [Piekļūts 12.05.2019].

[25] K. Podnieks, «Neironu tīkli,» [Tiešsaiste]. Pieejams: http://podnieks.id.lv/slides/mining/metodes4neironutikli.pdf. [Piekļūts 02.05.2019].

[26] P. J. Redmon, «Convolutional Neural Networks,» [Tiešsaiste]. Pieejams: https://github.com/pjreddie/darknet/blob/master/cfg/yolov2-tiny-voc.cfg#L122. [Piekļūts 07.05.2019].

[27] «AVDepthData - Cameras and Media Capture,» [Tiešsaiste]. Pieejams: https://developer.apple.com/documentation/avfoundation/avdepthdata. [Piekļūts 23.05.2019].

[28] S. M. P. S. Maneesh Apte, «YOLO Net on iOS,» [Tiešsaiste]. Pieejams: http://cs231n.stanford.edu/reports/2017/pdfs/135.pdf. [Piekļūts 24.05.2019].

PIELIKUMI

1. Pielikums. Objektu izsekošanas metode, izmantojot iOS Vision

```
while true {
  var requests = [VNRequest]()
  for inputObservation in inputObservations {
    guard let rectObservation = inputObservation.value as? VNRectangleObservation else
{ continue }
    let request = VNTrackRectangleRequest(rectangleObservation: rectObservation)
    request.trackingLevel = trackingLevel
    requests.append(request)
  }
  measureFPS()
  do {
    try requestHandler.perform(requests, on: frame, orientation: videoReader.orientation)
  } catch {
    print(error)
  }

  for request in requests {
    guard let results = request.results as? [VNObservation] else {
      continue
    }
    guard let observation = results.first as? VNDetectedObjectObservation else {
      continue
    }

    confidences.append(observation.confidence)

    let confidence = observation.confidence
    outputObservations.append(observation)
    inputObservations[observation.uuid] = observation
  }
}
```

2. Pielikums. Ieejas datu sagatavošana un rezultāta apstrāde YOLO modulī

```
private func predict(pixelBuffer: CVPixelBuffer, inflightIndex: Int) {
    guard let pixelBufferResized = resizedPixelBuffers[inflightIndex] else { return }
    let ciImage = CIImage(cvPixelBuffer: pixelBufferResized)
    let x_scale = CGFloat(448.0) / CGFloat(CVPixelBufferGetWidth(pixelBuffer))
    let y_scale = CGFloat(448.0) / CGFloat(CVPixelBufferGetHeight(pixelBuffer))
    let transformed = CGAffineTransform(scaleX: x_scale, y: y_scale)
    let scaled = ciImage.transformed(by: transformed)
    ciContext.render(scaled, to: pixelBufferResized)
    if let result = try? yolo.predict(image: pixelBufferResized) {
        self.setObjectsToTrack(cgRects: result.regions)
    } else {
       print("error")
    }

    self.semaphore.signal()
}
```

3. Pielikums. Trajektoriju iezīmēšana ielas attēlā

```
enum PathType {
  case yolo, vision
}

private let yoloPath = UIBezierPath()
private let visionPath = UIBezierPath()
private let yoloLayer = CAShapeLayer()
private let visionLayer = CAShapeLayer()
private var previousTouchPoint = CGPoint.zero
func drawPath(at point: CGPoint, pathType: PathType) {
  let path: UIBezierPath
  let shapeLayer: CAShapeLayer
  switch pathType {
  case .yolo:
    path = yoloPath
    shapeLayer = yoloLayer
  case .vision:
    path = visionPath
    shapeLayer = visionLayer
  }
  path.move(to: point)
  if self.previousTouchPoint !=  CGPoint.zero {
    path.addLine(to: self.previousTouchPoint)
  }
  self.previousTouchPoint = point
  shapeLayer.path = path.cgPath
  switch pathType {
  case .yolo:
    shapeLayer.strokeColor = UIColor.red.cgColor
  case .vision:
    shapeLayer.strokeColor = UIColor.green.cgColor
  }
}
```

4. Pielikums. Veiktspējas mērījumu metode

```
var lastFrame: CFAbsoluteTime?
var timeArray: [Double] = []

func measureFPS() {
    let timeInterval = CFAbsoluteTimeGetCurrent()
    if let lastFrame = lastFrame {
        let deltaTimeInSeconds = timeInterval - lastFrame
        let fps = deltaTimeInSeconds == 0 ? 0 : 1 / deltaTimeInSeconds
        timeArray.append(fps)
    }
    self.lastFrame = timeInterval
}
```

DOKUMENTĀRĀ LAPA

Bakalaura darbs "Velobraucēju kustības trajektoriju analīze, izmantojot iOS Vision attēlu atpazīšanu" izstrādāts LU Datorikas fakultātē.

Ar savu parakstu apliecinu, ka pētījums veikts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai.

Autors: _____ Mārcis Nīmants

Rekomendēju/nerekomendēju darbu aizstāvēšanai

Vadītājs: M. dat. Oskars Ozols _____ __.05.2019.

Recenzents: zinātniskais asistents, Mg.sc.comp, Ilvars Mizniks

Darbs iesniegts Datorikas fakultātē __.05.2019.

Dekāna pilnvarotā persona: vecākā metodiķe Ārija Sproģe _____

Darbs aizstāvēts bakalaura gala pārbaudījuma komisijas sēdē

____.06.2019. prot. Nr. _____

Komisijas sekretārs(-e): _____