Detekcija ulazaka i izlazaka vozila s parkirališta temeljena na računalnoj obradi slike

Varga, Luka

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Preddiplomski studij

DETEKCIJA ULAZAKA I IZLAZAKA VOZILA S PARKIRALIŠTA TEMELJENA NA RAČUNALNOJ OBRADI

Završni rad

Luka Varga

Osijek, 2020.

SADRŽAJ

1.UVOD

U današnje vrijeme, gradovi su sve više pod nadzorom video kamera, kako zbog sigurnosti tako i zbog pregleda prometa. Većina zgrada baš iz tog razloga ima instalirano spomenute. Te iste kamere često daju pogled i na ulaz parkirališta gdje to dolazi u korist ovog završnog rada.

U ovom završnom radu će se iskoristiti slika s nadzorne kamere kako bi se detektirao broj ulazaka i izlazaka vozila s parkinga odnosno na parking, putem nadzorne kamere. Računalna obrada slike omogućuje da se to napravi u realnom vremenu te da se u svakom trenutku zna broj automobila na parkiralištu, što može biti od koristi informirajući korisnike parkirališta o popunjenosti istoga. U početku rada, kratko su opisane tehnologije potrebne za izradu ovog završnog rada, kako i gdje se koriste, tko ih najviše koristi i slično.

Glavni dio rada se odnosi na primjenu spomenutih tehnologija, o njihovim raznim mogućnostima i bibliotekama korištenima uz iste. Govoreći o samom programu, podijeljen je na nekoliko dijelova, a najbitnije za navesti su biblioteke korištene tijekom rada te načini pomoću kojih se može doći do raznih video zapisa te sama njihova obrada.

1.1 Zadatak završnog rada

Zadatak je napraviti program u *C#* koji će biti u stanju detektirati i brojati automobile i ostala prijevozna sredstva, ovisno o tome da li dolaze ili odlaze s određenog parkirališta. Program će primati videozapis direktno iz video-nadzorne kamere te ga sekvencijalno obrađivati i sumirati rezultate.

2.KORIŠTENE TEHNOLOGIJE

Za potrebe pisanja ovog rada, korišteno je *Microsoftovo* razvojno okruženje *MS Visual Studio* [1], koje je studentima omogućeno preko *Microsoftovog DreamSpark* programa. *Visual Studio* omogućava brzo, lako i jednostavno pisanje i testiranje koda, pronalazaka grešaka te otklanjanje istih. Također sadrži i neke alate poput dizajnera oblika, dizajnera klasa i ostalih koji uvelike pomažu pri kreaciji raznih aplikacija. Uz sve već nabrojane pogodnosti, mogućnost korištenja raznih proširenja je jedna od najbitnijih stvari za navesti jer upravo zbog toga, mogu se koristiti mnogi dodatni alati i funkcionalnosti iz različitih biblioteka, a neki od njih su predstavljeni u nastavku rada.

2.1 C# i .NET

Programski jezik koji je odabran za izradu ovog rada je *C#*. *C#* omogućuje izgradnju raznovrsnih i moćnih aplikacija koje koriste i rade na *.NET* platformi (objašnjeno u sljedećem dijelu). On stoji kao programski jezik koji podržava objektno orijentiranu paradigmu programiranja, a do toga je došlo jer je upravo ovaj jezik nastao po uzoru na osnovnu inačicu *C* programskog jezika, uz brojne inovacije koje pospješuju razvoj aplikacija. Neke od značajki objektno orijentiranog programiranja su *enkapsulacija*, nasljeđivanje i *polimorfizam*. Upravo to omogućuje jednostavno svrstavanje objekata s kojima se radi, u velike i funkcionalne hijerarhije. *C#* koristi vrlo jednostavnu sintaksu, pa tako programerima koji su već upoznati s nekim od programskih jezika poput *C* ili *Jave*, neće predstavljati problem prilagoditi se. *C#* sintaksa predstavlja pojednostavljenje mnogih kompleksnosti *C++-a* te pruža neke korisne funkcionalnosti poput *null* vrijednosti, direktan pristup memoriji i ostalo [2].

.Net Framework, spomenut u odjeljku prije, predstavlja softversko okruženje za upravljanje izvršavanjem (engl. *run-time execution*) za *Windows,* koje pruža razne usluge svojim aplikacijama te omogućuje specifikaciju rezultata po pitanju namjene aplikacije (hoće li one biti namijenjene za *web, Windows, Windows Phone* itd.). Sastoji od dvije bitne komponente: virtualne mašine zvane *Common Language Runtime* , što je mehanizam za pokretanje koji rukuje aplikacijama u pogonu i *.NET Framework Class Library* koja omogućava knjižnicu ispitanih, višekratno iskoristivih kodova koje developeri mogu koristiti prilikom izrada svojih vlastitih aplikacija [2].

2.2 *Emgu CV* biblioteka

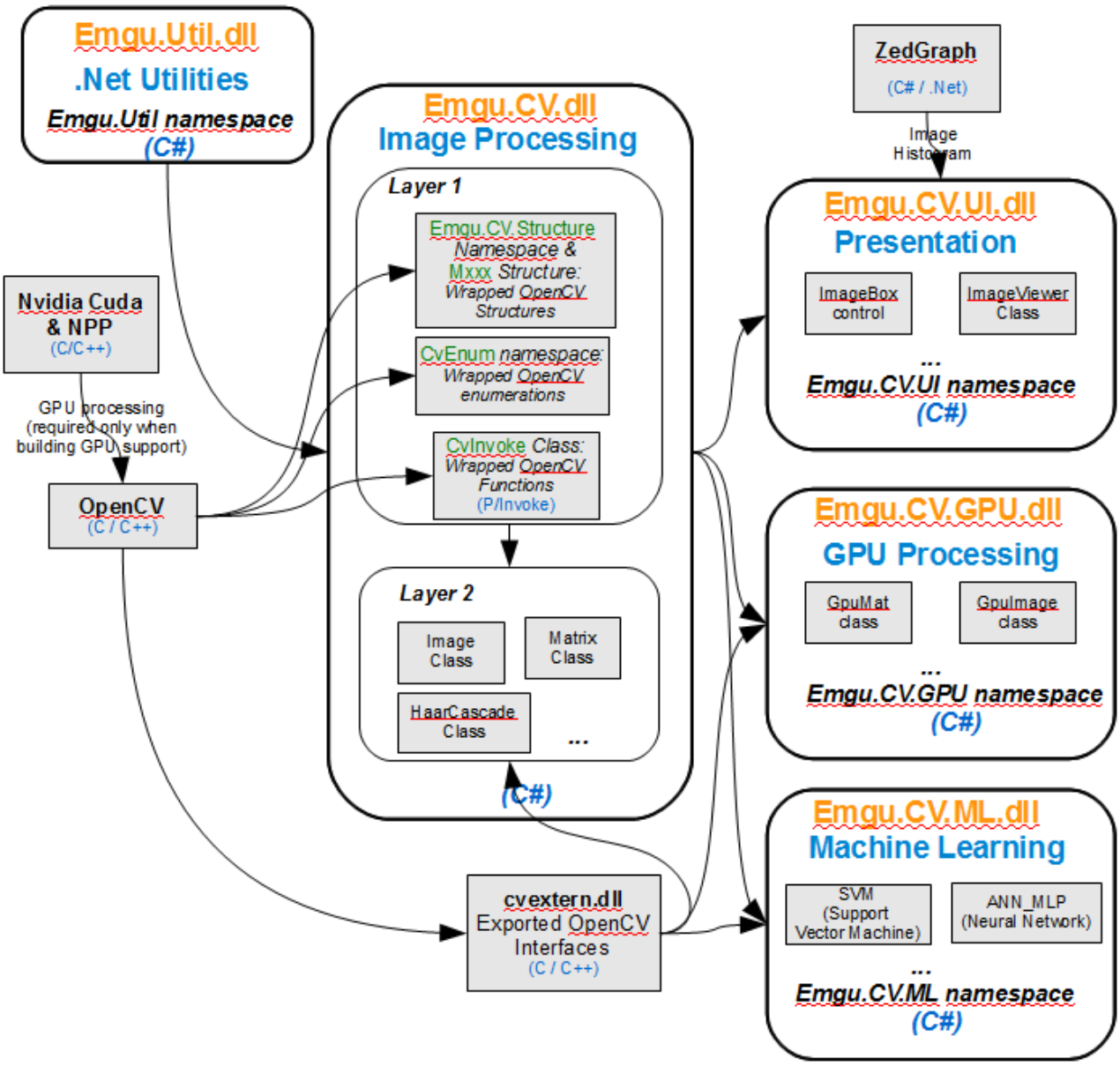
Emgu CV je omotač (engl. *wrapper*) koji služi za obradu slika. Usko je povezan s *OpenCV* (engl. *Open Source Computer Vision Library*) jer je *Emgu CV .NET* omot za *OpenCV*. Taj omotač omogućuje pozivanje funkcija *OpenCV* iz programskih jezika *.NET* C#, *VB IronPython i VC ++* a koji su samo neki od podržanih jezika. *Emgu CV* može se koristiti u *Mono*, a radi i pod *Linuxom*, *Windowsima, Mac Os x* i popularnim platformama poput *Androida, iPhonea, iPod Toucha* i *iPada*.

Za razliku od ostalih omotača poput *OpenCVDotNet*, *SharperCV* ili *Code Project* koji koriste nesigurni kod, *Emgu CV* je napisan u potpunosti u *C#*. Prednost tome je što se može sastaviti u *Mono* i zato je kompatibilan na bilo kojoj platformi koju podržava i *Mono*, poput *Linuxa, Solarisa* itd. Puno je napora uloženo u stvaranje čiste *C#* implementacije jer se zaglavlja moraju prenijeti, što bi se inače upravljanom *C++* implementacijom jednostavno uključilo, ali itekako vrijedi jer pruža udobnost i sigurnost znajući da je kod više-platformski.

Jedan od ciljeva *Emgu CV-a* je pružanje jednostavne infrastrukture za *.NET* programere i developere koja pomaže sastaviti poprilično sofisticirane vizualne aplikacije u kratkom vremenu. Biblioteka *Emgu CV* obuhvaća mnoga područja pogleda, uključujući tvornički pregled proizvoda, medicinsko snimanje, korisnička sučelja, kalibraciju kamere, stereo vid i robotiku. Budući da računalni vid i strojno učenje često idu jedno s drugim, *Emgu CV* također omotava cijelu biblioteku strojnog učenja opće namjene iz biblioteke za obradu slika *OpenCV*.

Još neke od dodatnih prednosti su automatsko prikupljanje smeća (engl.*garbage collection*), generičke operacije na pikselima slika, *XML* dokumentacija, *IntelliSense* podrška (automatsko popunjavanje koda, informacije o parametrima, popisi članova itd.) [3].

Arhitektura *Emgu CV* biblioteke vidljiva je na slici 2.1



Slika 2.1 Arhitektura *Emgu CV* [3]

3.IMPLEMENTACIJA METODE

3.1 Učitavanje video isječaka i slike s kamere

Prilikom rada na ovome projektu, korišten je video isječak (na slici 3.1 se nalazi primjer tzv. „*parking_example_1.avi*“), snimljen pomoću kamere, kako bi se program mogao testirati. U nastavku je prikazan dio koda koji je poslužio da se dohvati datoteka koja se nalazi na računalnom sustavu.

```
private void btnStart_Click(object sender, EventArgs e)
{
    ibStream.Enabled = false;
    if (capture == null)
    {
        try
        {
            capture = new VideoCapture("parking_example_1.avi");

        }
        catch (NullReferenceException excpt)
        {
            MessageBox.Show(excpt.Message);
        }
    }
```

Slika 3.1 Dohvaćanje video isječka

Nakon što je video isječak uspješno dohvaćen, dio koda koji omogućuje da se isti pokrene nalazi se u nastavku (slika 3.2). Uz ponuđenu opciju da se pokrene učitani snimak, dodana je i opcija zaustavljanja istoga, koja je aktivna cijelo vrijeme tijekom izvođenja videa, kako bi bilo moguće zaustaviti u bilo kojem željenom trenutku.

```
            if (capture != null)
            {
                if (captureInProgress)
                {  //if camera is getting frames then stop the capture and set button Text
                    // "Start" for resuming capture
                    btnStart.Text = "Start"; //

                    Application.Idle -= ProcessFrame;
                }
                else
                {
                    //if camera is NOT getting frames then start the capture and set button
                    // Text to "Stop" for pausing capture
                    btnStart.Text = "Stop";

                    Application.Idle += ProcessFrame;
                }
                captureInProgress = !captureInProgress;
            }
        }

        }


}
```

Slika 3.2 Pokretanje video isječka

Kako bi program bio u stanju obraditi dohvaćeni video, tu u punom izdanju dolazi upotreba *Emgu CV* biblioteke, koja omogućava da se video obradi na vrlo jednostavan način putem već prije integriranih metoda i funkcija koje dolaze upakirane u biblioteci. Nakon što je omogućen pristup snimljenom materijalu, metode za pristup podacima vezanih za određene okvire videa ili nekim podacima koji opisuju lokaciju promatranih objekata, mogu se dohvatiti putem metode *GetCaptureProperty* i njoj sličnih. Koje su to i kako to sve izgleda, prikazano je u nastavku:

```
using ...

namespace Emgu.CV
{
    public class VideoCapture : UnmanagedObject, IDuplexCapture, ICapture
    {
        public VideoCapture();
        public VideoCapture(CaptureType captureType);
        public VideoCapture(int camIndex);
        public VideoCapture(string fileName);

        public bool FlipHorizontal { get; set; }
        public bool FlipVertical { get; set; }
        public CaptureModuleType CaptureSource { get; }
        public int Width { get; }
        public FlipType FlipType { get; set; }
        public int Height { get; }
        public bool IsOpened { get; }

        public event EventHandler ImageGrabbed;

        public virtual void DuplexQueryFrame();
        public virtual void DuplexQuerySmallFrame();
        public double GetCaptureProperty(CapProp index);
        public virtual bool Grab();
        public void Pause();
        public virtual Mat QueryFrame();
        public virtual Mat QuerySmallFrame();
        public void Read(Mat m);
        public virtual bool Retrieve(IOutputArray image, int channel = 0);
        public bool SetCaptureProperty(CapProp property, double value);
        public void Start(System.ServiceModel.Dispatcher.ExceptionHandler eh = null);
        public void Stop();
        protected override void DisposeObject();

        public enum CaptureModuleType
        {
            Camera = 0,
            Highgui = 1
        }
    }
```

Slika 3.3 Integrirane funkcije *Emgu CV* biblioteke [3]

3.2 Optički tok (engl. *Optic flow*)

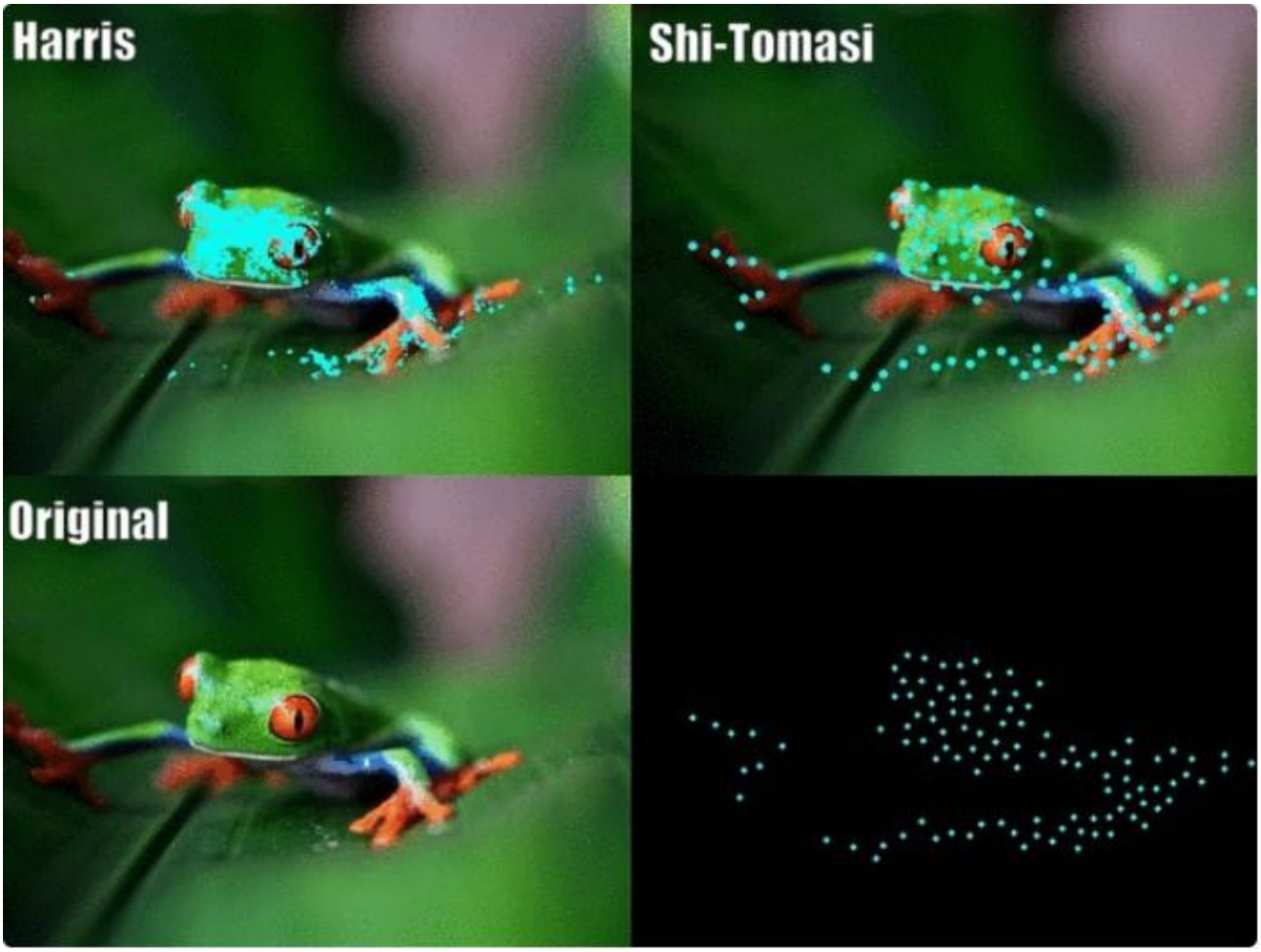
Pojam optički tok odnosi se na vizualne pojave koje se događaju svakodnevno. U osnovi, optički tok je prividni vizualni pokret koji se doživljava tijekom kretanja. Uz pretpostavku da promatrač sjedi u automobilu ili vlaku i gleda kroz prozor, on vidi da se drveće, zemlja, zgrade itd. pomiču. To kretanje je optički tok. Ovaj pomak također može reći koliko je promatrač blizu različitim predmetima koje vidi. Udaljeni predmeti poput oblaka i planina kreću se toliko sporo da izgledaju kao

da miruju. Predmeti koji su bliži, poput građevina i drveća, čini se da se kreću unazad, pri čemu se bliži objekti kreću brže od udaljenih objekata. Vrlo bliski predmeti, poput trave ili malih znakova uz cestu, kreću se tako brzo da se dobije osjećaj da prolete.

Postoje jasni matematički odnosi između veličine optičkog toka i mjesta na kojem se objekt nalazi u odnosu na promatrača. Ako udvostručite brzinu kojom promatrač putuje, optički protok koji vidi također će se udvostručiti. Ako se neki predmet dovede dvostruko bliže, optički protok će se opet udvostručiti. Također će se optički protok mijenjati ovisno o kutu između promatračevog smjera vožnje i smjera objekta koji gleda. Pretpostavimo da putuje naprijed. Optički protok je najbrži kada je objekt na promatračevoj strani za 90 stupnjeva ili neposredno iznad ili ispod njega. Ako se objekt približi smjeru naprijed ili nazad, optički protok će biti manji. Objekt neposredno ispred promatrača neće imati optički tok i činiti će se da miruje [4].

Nedavna istraživanja u računalnim tehnologijama omogućuju upravo identičnu mogućnost da računala opažaju svoj okolni svijet tehnikama poput ovih, zvane „*Object detection*“ koji pripadaju određenoj klasi i semantičkoj segmentaciji za pikselnu klasifikaciju.

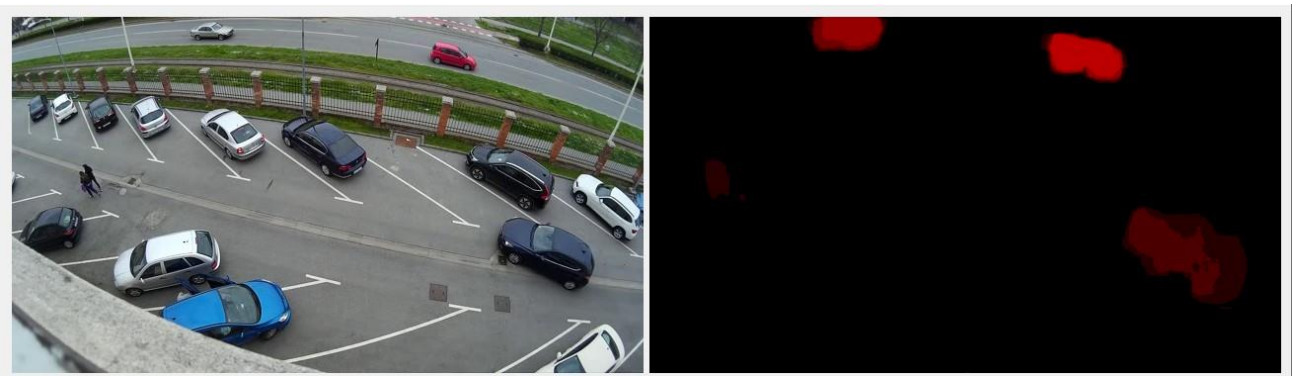
Međutim, za obradu video isječaka u stvarnom vremenu, implementacija ovih tehnika odnosi se samo na odnose objekata unutar istog okvira (x, y), a koji ignoriraju utjecaj varijable vremena (t). Drugim riječima, svaki okvir ocjenjuju neovisno, kao da su potpuno nepovezane slike, za svaku vožnju [5].



Slika 3.4 Prepoznavanje piksela [5]

Do sad se uglavnom pričalo o optičkom toku za rijetki skup piksela, ali kako bi zadatak bio odrađen što bolje i uspješnije, ovdje će se koristiti gusti optički tokovi (engl. *dense optical flow)*. Iako postoje razne izvedbe gustog optičkog toka, u ovom radu će se koristiti metoda „*Farneback*“, a koju je osmislio sam Gunnar Farneback, uz upotrebu *OpenCV-a*, biblioteke već prije spomenute, za implementaciju. Kod gustog optičkog toka, bitno je naglasiti da se ovdje pokušava izračunati vektor optičkog toka za svaki piksel svakog okvira, što zbog kompleksnosti računa, daje točniji i gušći rezultat, ali malo sporije, a koji je pogodan za aplikacije koje rade s pokretom i učenjem struktura te video segmentacije [6].

Na sljedećem primjeru (slika 3.4) je pokazano kako izgleda optički tok izgleda.



Slika 3.5 Primjer slike obrađene optičkim tokom [6]

Ovi rezultati, gdje su uspješno odvojeni dijelovi slike, tj. objekti koji su aktivni od onih koji nisu, dobiveni su koristeći integriranu metodu zvanu *CalcOpticalFlowFarneback* koja radi na način da uspoređuje dva susjedna okvira (engl. *frame*) tj. kretanje predmeta između svaka dva uzastopna okvira niza i uspoređuje promjene piksela uzorkovane kretanjem objekta koji se snima ili kamerom koja ga snima koje su se dogodile te ih izdvaja. Kako bi spomenuta metoda bila učinkovito iskorištena, predaje joj se nekoliko parametara objekta koji nam je u fokusu. Kako to izgleda, vidljivo je na slici 3.5.

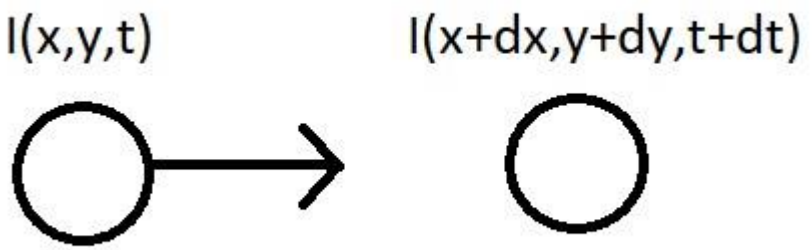
```
CvInvoke.CalcOpticalFlowFarneback(prevGrayImage, grayImage, FlowX, FlowY,
    0.5, 1, 20, 2, 5, 1.1, Emgu.CV.CvEnum.OpticalflowFarnebackFlag.UseInitialFlow);
```

Slika 3.6 Primjer *Farneback* metode

U prikazanom, redom predani parametri govore sljedeće: *prevGrayImage* označava prvi promatrani okvir ili sliku, dok *grayImage* predstavlja drugu tj. slijedeću sliku s kojom se prva uspoređuje. Parametri *FlowX* i *FlowY* označavaju izračunati protok slike, broj 0.5 određuje razmjer slike, broj 1 opisuje broj slojeva obrade slike, broj 20 prosječnu veličinu prozora. Broj 2 odredi koliko će se puta operacija algoritma odraditi na jednom od slojeva. Broj 5 je malo kompliciraniji jer on

određuje veličinu susjedstva piksela korištenog za pronalaženje polinomskog širenja u svakom pikselu. Veće vrijednosti znače da će se slika aproksimirati s glatkijim površinama što daje čvršći algoritam i više zamućenih polja. Zadnji broj među parametrima određuje standardnu Gaussovu devijaciju koja se koristi za izravnavanje derivata korištenih kao osnova za polinomsko širenje. Zadnji parametar govori metodi kako koristi ulazni protok slike kao početnu aproksimaciju cijelog protoka.

Sve gore navedeno od parametara, služi kako bi program što detaljnije mogao usporediti razlike između susjednih okvira i njihovih međusobnih razlika. A kako ta metoda izgleda matematički, to je prikazano formulom gdje se uspoređuje objekt intenziteta I (x, y, t) nakon vremena dt, kada se pomakne za dx i dy, gdje je sada novi intenzitet I (x + dx, y +dy, t + dt) [7]. Slikovito prikazano u nastavku.



Slika 3.7 Promjena intenziteta promatranog objekta [7]

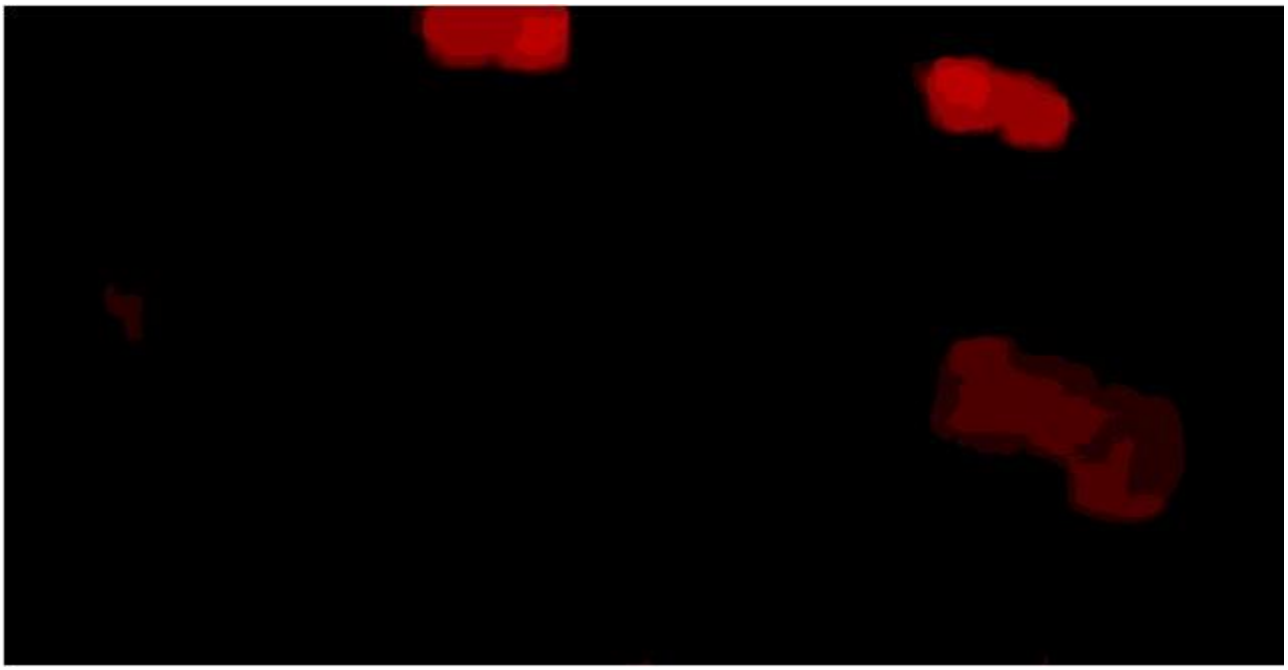
Kasnije, nakon usporedbe položaja, vizualizira kut (smjer) protoka prema nijansi i udaljenost (veličinu) protoka prema vrijednosti prikaza boja *HSV-a* (H engl. *hue* tj. nijansa, S engl. *saturation* tj. zasićenje, količina sive boje te V engl. *value* tj. veličina svjetline).

Sada na red dolazi još jedan korak obrade slike kako bi se istaknuli promatrani predmeti ili objekti da ga ljudsko oko lakše i efikasnije primijeti, a to je prebacivanje iz slike pune crvenih točkica i neodređenih geometrijskih likova u malo jasniju i vidljiviju verziju. To se postiže prebacivanjem u binarnu sliku gdje ne postoji ogroman broj nijansi neke boje već se sve prikazuje u dvije boje (crno i bijelo).

```
resultImage = redFlowResult.Convert<Gray, byte>()
    .ThresholdBinary(new Gray(10), new Gray(255));
```

Slika 3.8 Prijelaz u sivi prikaz [6]

Kako prethodno spomenuta metoda funkcionira možemo vidjeti ispod:



Slika 3.9 Prikaz optičkog toka u nijansama crvene boje [6]

Slika 3.10 Prikaz binarne slike optičkog toka [6]

3.3 Detekcija ulaska / izlaska

Najvažniji korak je, nakon svih jednostavnih koraka ranije, one prepoznate objekte s video kamere, nekako označiti ili uraditi nešto s njima. U aplikacijama računalnog vida često je zadatak otkrivanje i lokalizacija objekata. Pristupi otkrivanju objekata mogu se podijeliti u tri skupine: ručno izrađene metode koje se sastoje od nekih unaprijed definiranih pravila, pristupi bazirani na strojnom učenju gdje se podaci o objektima kodiraju i treći pristup koji je nešto između dvije predstavljene. Kako je cilj ovog programa detekcija ulaza/izlaza objekata, u ovom slučaju vozila na parking, fokus je na taj problem. S obzirom da nažalost nema već ranije osmišljene i/ili integrirane metode koja će pomoći u rješavanju spomenutog problema, rješenje je osmišljeno korak po korak. Odmah na početku su postavljene „mjerne točke“ ili točke koje će poslužiti pri detekciji vozila. Evo kako to izgleda:

```
//setting helping marks on road
colImage[250, 620] = new Bgr(0, 0, 255);
colImage[250, 600] = new Bgr(0, 0, 255);
```

Slika 3.11 Primjer koda za postavljanje točaka

Logika korištena pri rješavanju zadanog problema je da program promatra binarni optički tok objekata na snimci. U trenutku prijelaza objekta tj. vozila (auto, autobus, kombi …) preko mjesta predefiniranih točaka, na tom mjestu pojavila bi se bijela boja koja bi pokrenula signal koji nam govori da je vozilo prešlo određenu, zadanu granicu na slici. Spomenute točke su vrlo sitne, jedva primjetne, ali postavljene na dobroj lokaciji , to je u ovom primjeru na sredini puta kojim auti moraju proći nakon što im se podigne ulazna rampa, ni ne moraju biti ništa veće.

Slika 3.12 Prikaz postavljenih točaka

Nakon što vozilo prijeđe preko prve točke, i pošalje signal za to, program također pokreće i štopericu. Štoperica je uvedena kako bi se smanjile pogreške prilikom detekcije tj. registriranja automobila, jer preko promatranog parkinga prelaze, ne samo automobili, već i slučajni prolaznici, poput pješaka. Njezina uloga je tada da signal, ako traje manje od određenog zadanog intervala, ne ubroji kao prolazak vozila jer se očito radi o nekoj smetnji ili „šumu“. Tim postupkom su usput eliminirane i neke druge moguće pojave poput padanja listova, kiše ili bilo čega što se još može pojaviti prilikom snimanja.

```
//primjer provjere prolaska kroz neku točku
if (resultImage[250, 620].Intensity == 255)
{
    flag = 1;
    mpf2.Start();
}
if (flag == 1 && resultImage[250, 600].Intensity == 255 && mpf2.ElapsedMilliseconds > 7000)
{
    {
        mpf2.Reset();
        flag = 0;
        ct++;
    }

}

lbMessage.Text = "Broj ulazaka : " + ct.ToString();
```

Slika 3.13 Postavljanje uvjeta za slanje signala

Slika 3.14 Prepoznavanje novog ulaska

Nakon što su uspješno uklonjene smetnje i slučajni prolaznici, nastavlja se promatranje ponašanja programa. Signal prelaska preko prve točke je poslan, tijekom trajanja intervala kojeg štoperica odbrojava, vozilo prelazi i preko druge postavljene testne točke. Svi uvjeti koji su zatraženi od programa da se ispoštuju su zadovoljeni i program registrira ulazak vozila na parking.

4. TESTIRANJE

Prilikom obavljanja testova, nisu korištene snimke uživo već je korišten video koji je snimljen s prozora zgrade fakulteta. Tijekom trajanja videa, na parking dođe nekoliko objekata, a uz parking je vidljiva i cesta na kojoj prolazi nekoliko desetina automobila te tramvaj. Na snimci je dovoljno aktivnosti kako bi se očekivani testovi mogli provesti uspješno i u potpunosti. Kako bi bilo moguće testirati program i samu logiku izvođenja istoga, mjerne točke su prvo bile postavljene na cesti pored parkinga, jer je puno veća aktivnost objekata koji prolaze tim dijelom ekrana, pa su nakon provođenja spomenutog, prebačene na promatrani parking, jer je naposljetku taj dio cilj ovog rada.

4.1 Točnost detekcija

Iako se pokazalo da su pristupi temeljeni na strojnom učenju uglavnom najbolji, u ovom slučaju se oni nisu mogli primijeniti. Jedan od glavnih razloga je bio taj što postupci strojnog učenja vrlo često zahtijevaju mnogo uzoraka i primjera, a prikupljanje istog uglavnom nije jednostavan zadatak, jer prije svega zahtjeva mnogo vremena. Uz navedeni problem, sam postupak treniranja programa obično potraje dugo i nije u mogućnosti prilagoditi se na licu mjesta. Postoji još nekoliko predložaka za rješenje danog problema, koji su jednostavniji i ne zahtijevaju opsežnu obuku, koji spadaju u neke druge biblioteke i mogu se implementirati, ali biblioteka odabrana za korištenje je već prije navedene *Emgu CV*. Točnost detekcija koje se odnose na primjer video isječka opisane su u nastavku. Na samom početku, na parkingu je već parkirano 13 automobila. To se može vidjeti na sljedećoj slici:

Slika 4.1 Početno stanje broja vozila na parkingu

Slika 4.2 Povećanje broja parkiranih vozila dolaskom novog vozila

Kako bi se vidjelo da je početak video isječka, ispod broja ulazaka je prikazan i broj obrađenih okvira snimke, na slici 4.1 označeno brojem 7. U nastavku, kada na parking prilazi prvo novo vozilo, program će uspjeti registrirati ulaz, te će se broj ulazaka povećati (slika 4.2).

Nakon nekoliko stotina obrađenih okvira videa, na parking pristiže i drugo vozilo te program još jednom uspješno registrira objekt te povećava ukupan broj ulazaka na parking:

Slika 4.3 Dodatno povećanje broja vozila na parkingu

Tijekom ostatka snimke, na parking više ne pristigne ni jedno novo vozilo, no oba pristigla vozila se udalje s mjesta parkinga te isti napuste. Bitno je naglasiti da uz prolaske vozila, preko mjernih točaka prijeđu i dva prolaznika tj. šetača i program to ipak ne shvati kao odlazak vozila već uspješno registrira samo kao smetnju.

Slika 4.4 Prvo vozilo napušta parking

Slika 4.5 Drugo vozilo napušta parking

Iz svega priloženoga, može se zaključiti da je program izvršio svoj zadatak te pravilno prikazao tj. detektirao prikazane promjene.

4.2 Brzina obrade

Ovisno o vrsti video datoteke odnosno ekstenzije preko koje je pohranjena na uređaj, kodecima (engl. *codecs*) koji su instalirani na računalu, fizički hardver uređaja, i možda najbitnije, rezolucije u kojoj je snimani video, velik dio toka za obradu videozapisa zapravo se može potrošiti čitanjem i dekodiranjem sljedećeg okvira u video datoteci. Poboljšavanjem odabira prioriteta, stavljajući određene pozadinske programe na mirovanje, ili jednostavno unaprjeđenje tehnologije i fizičkih uređaja pomoću kojih računalo funkcionira, poput procesora i grafičke kartice, sve utječe na brzinu obrade video datoteke.

Od nekih slabijih konfiguracija, poput ove na kojoj je obavljeno testiranje programa, omogućuju programu da procesira od 7,8 pa do 11 ili 12 okvira videa u sekundi. Jača i modernija računala tu brojku mogu povećati i do nekoliko puta pa možemo vidjeti i brojke koje se kreću od 25 pa i 30 okvira u sekundi.

Slika 4.6 Broj procesiranih okvira u sekundi

5. ZAKLJUČAK

Kako je velika većina problema svakodnevice riješena pomoću raznih programa i aplikacija, upravo to je bila prekretnica za ovaj rad. U svrhu detektiranja vozila na parkiralištima, pokrivenih nadzornim kamerama, ovaj program je tu da to i ostvari. Program je osmišljen tako da, postavljena kamera koja nadzire parkirni prostor, snimku šalje programu, koji istu obrađuje i vraća povratnu informaciju o tome koliko vozila je ušlo odnosno izašlo s parkirališta te ih obavještava koliko je ukupno vozila na parkiralištu odnosno koliko slobodnih mjesta još ima.

Program je rađen u *C#* programskom jeziku koji je odabran jer najbolje odgovara rješavanju ponuđenog problema, uz dosadašnje prikupljeno znanje. Pored jednostavnosti koju osigurava, daje i laku mogućnost dodavanja i korištenja raznih biblioteka i proširenja koje u sebi sadrže mnoge dodatne materijale koji isto tako olakšavaju cijeli postupak. Za pristup problemu, kao i metodama i logici rješavanja istog, jako je dobro došlo znanje stečeno na kolegijima Programiranje I i II, te kolegijima Objektno-orijentirano programiranje i Razvoj programske podrške objektno orijentiranim načelima.

Kako se radi o problemu koji se može javiti u mnogim sličnim situacijama, vrlo je korisno bilo raditi na njemu jer se ovakav pristup može koristiti na velikom području. Što se samog programa tiče, postoji mogućnost ponekih grešaka jer smetnje poput nasumičnog prolaza šetača mogu poremetiti broj prikazanih parkiranih vozila, gdje se to, možda nekim drugačijim pristupom umjesto vremenskog intervala prolaza, može izbjeći. Ali postoji i prostor i način za poboljšanje, tako što bi se koristila bolja oprema prilikom snimanja prostora, gdje bi kvaliteta i razlučivost slike bila bolja i veća, te postavljanje senzora na možda nekim, strateški bolje određenim lokacijama.

LITERATURA

[1] Microsoft Visual Studio, https://visualstudio.microsoft.com/vs/ srpanj 2020.

[2] C# Basics for Beginners: Learn C# Fundamentals by Coding , https://www.udemy.com/course/csharptutorial-for-beginners/ , srpanj 2020.

[3] Emgu CV, About Emgu, https://stackoverflow.com/tags/emgucv/info , srpanj 2020.

[4] Introduction to Optical Flow, https://www.centeye.com/technology/optical-flow/ , srpanj 2020.

[5] Introduction to Motion Estimation with Optical Flow, https://nanonets.com/blog/optical-flow/ , srpanj 2020.

[6] OpenCV- The Gunnar-Farneback optical flow, https://www.geeksforgeeks.org/opencv-the-gunnar-farneback-optical-flow/ , srpanj 2020.

[7] Rapid Object Detection in C#, https://www.codeproject.com/Articles/826377/Rapid-Object-Detection-in-Csharp , srpanj 2020.

SAŽETAK

Naslov: Detekcija ulazaka i izlazaka vozila s parkirališta temeljena na računalnoj obradi slike

Cilj ovog završnog rada bio je napraviti program u *C#* koja će pomoći vozačima, posebno vozačima osobnih automobila, da lakše dođu do slobodnih parkirnih mjesta. Na početku samoga rada, kratko su opisane tehnologije potrebne za izradu ovog završnog rada, kao i glavni problem o kojemu se radi, tj. detekcija prolaska vozila putem optičkog toka. Programski jezik korišten za ostvarenje cilja je *C#*. Uz njega korišteni su *.NET* platforma, *Emgu CV* biblioteka te *Microsoft Visual Studio 2019*. Ponajprije je bilo potrebno omogućiti programu da dohvati video zapis kojega kamera snima. Nakon toga, program snimku obrađuje i putem optičkog toka, prepoznaje objekte koji se na njoj nalaze. Prepoznate objekte prebrojava prilikom prolaska preko određenih koordinata te javlja je li vozilo tek prišlo parkingu ili s njega izlazi. Program ispisuje dobiveni rezultat te pruža dodatnu informaciju o ukupnome broju parkiranih vozila i brzini obrade ukupnog broja okvira videa. Na ovaj način, prikazom broja parkiranih vozila, dolazi se do informacije je li parkiralište popunjeno ili ne.

Ključne riječi: video, detekcija objekata, *C#*, *Emgu CV*, optički tok, ulazak na parkiralište

ABSTRACT

Title: Detection of vehicle entrances and exits from the parking lot based on computer image processing

The goal of this project was to develop a program in *C#* which would help the drivers, especially car drivers, to get more easily to free parking spaces. At the beginning of the work, the technologies required to make this program are briefly described, as well as the main problem encountered, i.e. the detection of vehicle passage using optic flow. The programming language used to achieve the goal is *C#*. In addition, the *.NET* platform, the *Emgu CV* library and *Microsoft Visual Studio 2019* were used. First of all, it was necessary to enable the program to retrieve the video that the camera records. After that, the program processes the image and with the use of optic flow, recognizes the objects that are on it. It counts the recognized objects when passing over certain coordinates and reports whether the vehicle has just approached the parking lot or is exiting it. The program prints the obtained result and provides additional information on the total number of parked vehicles and the processing speed of the total number of video frames. In this way, by showing the number of parked vehicles, one comes to the information if the parking lot is full or not.

Key words: video, object detection, *C#*, *Emgu CV*, optic flow, parking lot entrance

ŽIVOTOPIS

Luka Varga rođen je 16. kolovoza 1996. godine u Osijeku. 2003. godine započinje obrazovanje u OŠ Ivana Kukuljevića u Belišću te nakon toga upisuje smjer opće gimnazije u Srednjoj Školi Valpovo. Tijekom školovanja postiže vrlo dobar uspjeh uz sudjelovanja na brojnim natjecanjima. 2014./2015. upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku, preddiplomski smjer – Računarstvo.