C# aplikacija za nadgledanje parkirališta temeljena na računalnoj obradi slike

Češić, Nikolina

Master's thesis / Diplomski rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:487593

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-21**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Diplomski sveučilišni studij računarstva

C# APLIKACIJA ZA NADGLEDANJE PARKIRALIŠTA TEMELJENA NA RAČUNALNOJ OBRADI SLIKE

Diplomski rad

Nikolina Češić

Osijek, 2020.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 18.09.2020.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime studenta:	Nikolina Češić
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-953R, 27.09.2019.
OIB studenta:	50052826767
Mentor:	Izv. prof. dr. sc. Irena Galić
Sumentor:	Dr. sc. Krešimir Romić
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Doc.dr.sc. Mirko Köhler
Član Povjerenstva 1:	Izv. prof. dr. sc. Irena Galić
Član Povjerenstva 2:	Dr. sc. Hrvoje Leventić
Naslov diplomskog rada:	C# aplikacija za nadgledanje parkirališta temeljena na računalnoj obradi slike
Znanstvena grana rada:	**Obradba informacija (zn. polje računarstvo)**
Zadatak diplomskog rada:	Potrebno je razviti metodu za praćenje ulazaka i izlazaka, te praćenje broja slobodnih mjesta na parkiralištu. Metoda treba kao ulazni podatak uzimati sliku s nadzorne kamere i postupcima računalne obrade slike dobiti tražene informacije u stvarnom vremenu. Metodu je potrebno razviti u C# programskom jeziku uz pomoć EmguCV biblioteke. Sumentor je dr. sc. Krešimir Romić. Tema je rezervirana za Nikolinu Češić.
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 2 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 2 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	18.09.2020.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 01.10.2020.

Ime i prezime studenta:	Nikolina Češić
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-953R, 27.09.2019.
Turnitin podudaranje [%]:	8

Ovom izjavom izjavljujem da je rad pod nazivom**: C# aplikacija za nadgledanje parkirališta temeljena na računalnoj obradi slike**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Irena Galić

i sumentora Dr. sc. Krešimir Romić

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD 1
1.1. Pregled područja 1
2. KORIŠTENE TEHNOLOGIJE 2
2.1. C# programski jezik 2
2.2. Emgu CV 4
2.3. .NET Framework 5
2.4. Common Language Runtime 6
2.5. .NET Framework biblioteka klasa 7
2.6. Microsoft Visual Studio 8
2.6.1. Visual Studio 2017. 8
3. IMPLEMENTACIJA METODE 9
3.1. Ideja za rješavanje problema 9
3.2. Učitavanje video isječka 9
3.3. Optical-flow 12
3.4. Detekcija ulaska/izlaska 14
4. TESTIRANJE 21
4.1. Preciznost 21
4.2. Brzina obrade 26
5. ZAKLJUČAK 30
LITERATURA 31
SAŽETAK 33
ABSTRACT 34
ŽIVOTOPIS 35

1. UVOD

U današnje vrijeme, tehnologiju je moguće primijeniti u gotovo svakom dijelu života pa tako i za rješavanje naizgled jednostavnih problema kao što je nadzor slobodnih, odnosno zauzetih parkirališnih mjesta na bilo kojem parkiralištu uz pomoć statične nadzorne video kamere.

Krajnji cilj diplomskog rada bio je prikazati podatke prikupljene video kamerom u stvarnom vremenu te na osnovu njih odrediti koliko parkirališnih mjesta je slobodnih, a koliko zauzetih na nadgledanom parkiralištu. Dobivene informacije bilo je potrebno prikazati na samoj aplikaciji kako bi korisnik imao uvid u stanje slobodnih i zauzetih parkirališnih mjesta.

Za obradu slike dobivene kamerom, korištena je EmguCV tehnologija koja omogućuje manipuliranje video materijalom kako bi se olakšalo dobivanje informacija koje će biti prikazane na aplikaciji. Sama aplikacija izrađena je uz pomoć C# programskog jezika te Visual Studio razvojnog okruženja.

1.1. Pregled područja

U svrhu proučavanja teme diplomskog rada, otkrivena su različita moguća rješenja problema navedenog u uvodu. Dok su u istraženim rješenjima problematiku rješavali označavanjem rubnih dijelova parkirališnih mjesta ili detektiranjem rubova koji se pojavljuju, problem diplomskog rada riješen je povlačenjem linije koja se dijeli na četiri jednaka dijela čime se dobije pet točaka na kojima se detektira ulazak i izlazak automobila. Neke od istraženih metoda u nastavku će biti ukratko objašnjene, a metoda rješavanja problema diplomskog rada detaljnije će biti objašnjenja u nadolazećim poglavljima.

Metoda zasnovana na značajkama slike, prema [1], je metoda koja podrazumijeva obradu videa s fiksne kamere korištenjem računalnog vida te kao takva ima brojne prednosti kao što su niži troškovi početne implementacije te mogućnost snimanja u svrhu nadzora ili u sigurnosne svrhe. Koristi se algoritmom za adaptivno oduzimanje pozadine kako bi se otkrili objekti u prvom planu. Detektirani objekti se tada prate koristeći Kalman filter s ciljem praćenja njihove putanje i otkrivanja ulazaka i izlazaka s parkirališnog mjesta. Kalman filter ima široku primjenu u tehnologiji kao što su vođenje, navigacija i kontrola vozila, a posebice zrakoplova, svemirskih letjelica te dinamički smještenih brodova kako je predstavljeno u[2] .

Metoda detekcije rubova, kako je predstavljeno u [3], koristi *Canny edge* algoritam kojim se slika pretvara u crno-bijelu s naglaskom na piksele za koje se pretpostavlja da su rubovi. Ukoliko je visok omjer piksela rubova prema ukupnom broju piksela slike, zaključuje se da je parkirališno mjesto zauzeto i obrnuto.

2. KORIŠTENE TEHNOLOGIJE

2.1. C# programski jezik

C# je moderan objektno-orijentiran programski jezik koji programerima omogućuje stvaranje različitih sigurnih i snažnih aplikacija koje se pokreću na .NET Frameworku. Može se koristiti za stvaranje *Windows Forms* aplikacija, XML internetskih usluga, klijent-server aplikacije, aplikacije za upravljanje bazom podataka i još mnoge. Pruža napredno uređivanje koda, pogodne izglede korisničkog sučelja, ugrađeno pronalaženje pogrešaka i mnoge druge alate koji omogućuju lakše razvijanje aplikacija utemeljenima na C# programskom jeziku i .NET Frameworku.

C# je izrazito izražajan jezik, ali vrlo jednostavan i lak za učenje. Programeri koji su upoznati s bilo kojim jezikom, C, C++ ili Java, mogu započeti rad s C# programskim jezikom u vrlo kratkom vremenu. C# sintaksa pojednostavljuje mnoge složenosti C++ programskog jezika i pruža snažne značajke kao što su tipovi podataka koji mogu imati vrijednost nula, lambda izraze i izravno pristupanje memoriji što nije moguće u Javi. Kao objektno-orijentiran jezik, podržava koncept enkapsulacije, nasljeđivanja i polimorfizma. Sve varijable i metode, uključujući *Main* metodu, polaznu točku aplikacije, su enkapsulirane unutar definicije klase. Klasa se može naslijediti izravno od jedne roditeljske klase, ali može implementirati više sučelja. C# proces izgradnje aplikacije je jednostavan u usporedbi s C i C++ te je fleksibilniji nego Java. Ne postoje odvojene zaglavne datoteke i ne zahtjeva se deklariranje metoda i tipova određenim redoslijedom te je moguće definirati neograničeni broj klasa, struktura, sučelja i događaja. [4]

Tab.2.1. *C# prostori imena korišteni u izradi projekta*

System	Sadrži osnovne klase koje definiraju često korištene vrijednosti i referentne vrste podataka, događaje, sučelja…
System.Collections.Generic	Sadrži sučelja i klase koje definiraju generičke kolekcije
System.ComponentModel	Pruža klase koje se koriste za implementiranje ponašanja vremena izvršavanja i dizajniranja komponenti i kontrola
System.Data	Pruža pristup klasama koje predstavljaju ADO.NET arhitekturu
System.Drawing	Pruža pristup GDI+ osnovnoj grafičkoj funkcionalnosti
System.Linq	Pruža klase i sučelja koji podržavaju upite koji koriste LINQ (*Lanuage-Integrated Query*)
System.Text	Sadrži klase koje predstavljaju ASCII i Unicode kodiranje znakova
System.Threading.Tasks	Pruža tipove koji pojednostavljuju rad pisanja istovremenog i asinkronog koda
System.Windows.Forms	Sadrži klase za stvaranje aplikacija za Windows OS
System.Diagnostics	Sadrži tipove koji omogućuju odnos sa sustavnim procesima, događajima i brojačima izvođenja

Svi programski jezici imaju osnovne komponente koje formiraju veće jedinice za izgradnju korisnih aplikacija pa se tako i C# program sastoji od određenih elemenata. Ulazni elementi podrazumijevaju komentare, praznine, identifikatore, ključne riječi, operatore i sl. [5]

Komentari nisu dio koda koji se izvršava, oni pružaju informaciju korisniku, odnosno programeri ih koriste kako bi korisniku program bio pojašnjen. Dvije kose crte, //, ispred teksta označavaju isti komentarom. Ukoliko se komentar proteže kroz više redova, početak komentara označava se oznakom /* bez razmaka između, a završetak komentara oznakom */, također bez razmaka između.

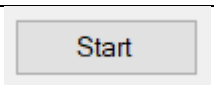
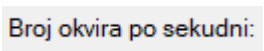
Praznine uključuju razmake i tabulatore koji se koriste za formatiranje teksta kako bi bio lakši i jednostavniji za čitanje i izmjenu.

Identifikatori su riječi programa, neke izabere sam programer, a neki su ključne riječi rezervirane za posebnu upotrebu kao što su, na primjer, *public*, *private*, *class*, *void* i sl.

Operatori su oznake koje simboliziraju operacije koje će se izvoditi kao što su zbrajanje ili množenje. C# sadrži mnoge grupe operatore kao što su aritmetički operatori, operatori usporedbe, logički operatori, operatori bitova i zamjene te operatori jednakosti.

U samoj izradi projekta C# programski jezik korišten je prvenstveno za kreiranje *Windows Forms* aplikacije. U *Windows Forms* izrazu, *form* označava vidljivu površinu na kojoj se prikazuju informacije korisniku. *Windows Forms* aplikacija se izrađuje dodavanjem kontrola spomenutoj formi i razvijanjem odgovora na korisniče zahtjeve, odnosno poduzete akcije kao što su klik računalnog miša ili pritisak tipke tipkovnice. Kada korisnik kreira nekakav zahtjev na formi ili na nekom od kontrola kreiranih u formi, taj zahtjev generira događaj na koji aplikacija reagira koristeći napisani kod. *Windows Forms* može sadržavati različite kontrole koje programer može dodavati po svojoj želji ili potrebi, a neke od kontrola su *text-box, button, drop-down box, radio button* i sl. U Visual Studiu, u kojem je projekt i izrađen, može se koristiti metoda *drag-and-drop* za kreiranje *Windows Forms* aplikacije, odnosno za dodavanje potrebnih kontrola na određeno mjesto u formi. [6]

Tab. 2.2. *Kontrole korištene u izradi Windows Forms aplikacije*

Button		Pokreće, zaustavlja ili prekida proces.
ImageBox		Kontrola koju omogućuje EmguCV, a zamjenjuje *PictureBox*. Umjesto bit mape prikazuje sliku kao objekt.
Label		Prikazuje tekst koji korisnik ne može izmijeniti.
ListBox		Prikazuje listu podataka.

2.2. Emgu CV

EmguCV je višeplatformska biblioteka za obradu slike. Usko je povezan s OpenCV bibliotekom jer je EmguCV .NET omotač (engl. *wrapper*) u OpenCV[1] biblioteci za obradu slike.

[1] OpenCV je biblioteka programskih funkcija koje je razvila tvrtka Intel. Koriste se za računalni vid u stvarnom vremenu. Napisano je u optimiziranom C i C++ programskom jeziku, a sadrže preko 500 funkcija koje pokrivaju različita područja računalne vizije.

OpenCV funkcije mogu biti pozvane .NET kompatibilnim jezicima kao što su C#, VB, VC++, IronPython, itd. Omotač (engl. *the wrapper*) može prevesti Visual Studio, Xamarin Studio i Unity, a može biti pokrenut na Windowsu, Linuxu, Mac OS X-u, iOS-u, Androidu i Windows Phoneu. EmguCV je u potpunosti napisan u C# programskom jeziku. Prednost je da može biti preveden u Monu što omogućuje pokretanje na bilo kojoj platformi koju podržava Mono[2], uključujući iOS, Android, Windows Phone, Mac OS X i Linux. Jedan od ciljeva EmguCV-a je omogućiti infrastrukturu računalnog vida koja je .NET programerima jednostavna za korištenje te im olakšava kreiranje aplikacija. EmguCV biblioteka se rasprostranjuje na mnoga područja računalnog vida uključujući tvornički pregled proizvoda, medicinsko snimanje, korisnička sučelja, kalibracija kamere, robotika i sl., kako je predstavljeno u [7]. Posljednja izdana stabilna verzija EmguCV-a je Emgu.CV-4.3.0, a izdana je u lipnju 2020. godine.

Osim što je u potpunosti napisan u C# što omogućuje pokretanje na mnogim platformama i što se može pokrenuti u okruženjima mnogih programskih jezika, EmguCV ima još neke dodatne prednosti:

- *Image* klasa s generičkim parametrima *Color* i *Depth*
- Automatsko upravljanje memorijom
- Podržano i korištenje *Image* klase i direktno pozivanje funkcija iz OpenCV-a
- Generičke operacije na pikselima slike [7]

Tab. 2.3. *EmguCV prostori imena korišteni u izradi projekta*

Emgu.CV	Omotač OpenCV funkcija za obradu slike, prema [8]
Emgu.CV.GPU	Računalni vid koristi GPU,prema [9]
Emgu.CV.UI	Korisničko sučelje (ImageBox) za prikazivanje objekta slike, prema [10]
Emgu.Util	Skup funkcionalnosti korišten u Emgu projektima, prema [11]

2.3. .NET Framework

Microsoft je razvio C# programski jezik zajedno s .NET razvojnim okruženjem, novom računalnom platformom koja pojednostavljuje razvoj aplikacija. Dizajnirali su .NET okruženje kako bi ispunili sljedeće ciljeve:

[2] Mono je projekt besplatnog i otvorenog koda (engl. *open-source*) predvođen Xamarinom, podružnicom Microsofta s odgovarajućim setom alata .NET okruženja između ostalog uključujući C# prevoditelj (engl. *compiler*) i CLR (Common Language Runtime).

- za omogućavanje dosljednog objektno-orijentiranog okruženja bilo da je kod pohranjen i izvršen lokalno, izvršen lokalno, ali podijeljen na internetu ili izvršen udaljeno
- za omogućavanje okruženja za izvršavanje koda koje minimizira razvoj softvera i sukob verzija
- za omogućavanje okruženja za izvršavanje koda koje osigurava sigurno izvršavanje koda uključujući kod kojeg je kreirala nepoznata ili nepouzdana treća strana
- za stvaranje iskustva programera dosljednog širokom rasponu različitih tipova aplikacija kao što su aplikacije za Windows operacijski sustav i internet aplikacije
- za izgradnju svih komunikacija industrijskog standarda kako bi se osiguralo da se kod baziran na .NET okruženju može integrirati s bilo kojim drugim kodom

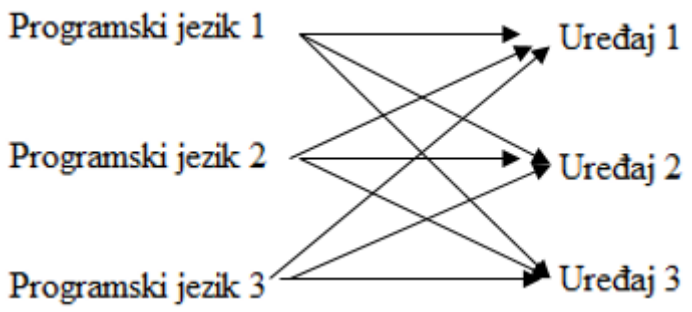
Svrha .NET okruženja ispunjava zahtjeve profesionalnog programera, a dva glavna dijela tog okruženja su CLR *(*engl. *Common Language Runtime)* i .NET Framework biblioteka klasa. [5]

2.4. Common Language Runtime

CLR pokreće kod i omogućuje usluge koje olakšavaju razvojni proces. „*Runtime*“ označava da se kod pokreće, odnosno da se kod izvršava. „*Common Language*“ znači da se pokrenuti može kod pisan u različitim programskim jezicima koji dijele istog pružatelja usluga.

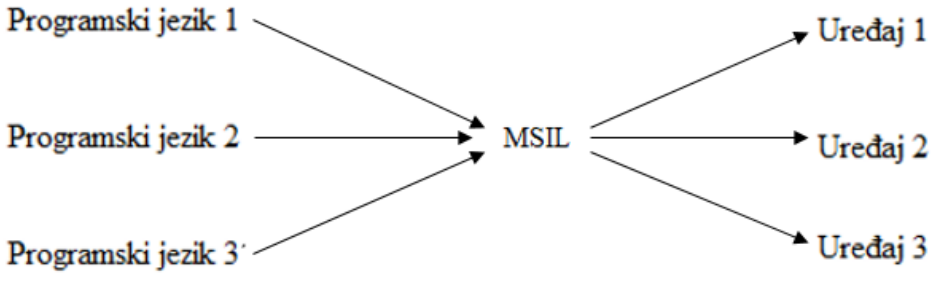
Microsoft je razvio C# kako bi iskoristio prednosti CLR-a. Njegove značajke rade iznimno dobro s CLR-om. Popularni Visual Basic programski jezik razvio se u Visual Basic .NET koji je objektno orijentirani jezik koji koristi prednosti CLR-a. Visual Basic programeri moraju naučiti mnoge nove značajke kako bi iskoristili CLR koristeći Visual Basic .NET. C++, kao i njegov prethodnik, C, ima mnoge mogućnosti koje ne odgovaraju novom pristupu. Verzija C++ jezika, zvana prilagođeni C++, prilagođava C++ radu s CLR-om kako bi C++ programeri mogli integrirati kod s drugim CLR korisnicima.

Veliki problem s kojim se susreću programeri je veliki broj različitih tipova procesora koji pokreću kod. Windows, Macintosh i Unix strojevi koriste široku raznolikost hardvera kao i osobni digitalni asistenti (PDA), mobiteli, velika računala i druge platforme. Jedan način koji može omogućiti rad programa na nekom uređaju je prevođenje programa u izvorni skup naredbi (Sl. 2.1.) za taj uređaj. Koristeći ovaj pristup, u najgorem slučaju će biti potrebno sto prevoditelja za prijevod deset različitih programa u izvorni skup naredbi za svaki od deset uređaja, odnosno bit će potrebno onoliko prevoditelja koliki je umnožak korištenih programskih jezika i uređaja.



Sl. 2.1. *Prevođenje tri jezika u izvorni kod za tri uređaja*

Drugi pristup je koristeći CLR, odnosno pružanje posrednog jezika (Sl. 2.2.) koji ima funkciju kao i izvorni jezik uređaja. Taj jezik naziva se MSIL (*Microsoft Intermediate Language*). Prilikom pokretanja, CLR koristi JIT (*Just In Time*) prevoditelj za prijevod MSIL koda u izvorni kod za korišteni uređaj. To zahtjeva jedan JIT prevoditelj za svaki uređaj koji će prevoditi MSIL kod u izvorni kod tog uređaja. Ovakav proces prijevoda nije zahtjevan kao prijevod višeprogramskih jezika u izvorni kod jer je MSIL kod vrlo sličan izvornom kodu. [5]



Sl. 2.2. *Prevođenje koristeći posredni jezik*

2.5. .NET Framework biblioteka klasa

.NET Framework biblioteka klasa pruža veliki i vrlo koristan set tipova koji ubrzavaju razvojni proces. Biblioteka grupira tipove u prostore imena kombinirajući povezane tipove. Sadrži 100 prostora imena od kojih koristimo samo nekoliko, a neki od njih su:

- System – sadrži osnovne tipove
- System.Collections – definira različite kolekcije objekata
- System.Data – upravlja podacima iz različitih izvora uključujući i baze podataka
- System.Drawing – omogućuje grafike
- System.IO – dopušta čitanje i pisanje
- System.Net – pruža sučelje koje računala koriste za komunikaciju putem mreža
- System.Runtime.Remoting – podržava distribuirane aplikacije

- System.Text – upravlja kodiranjem znakova
- System.Threading – omogućava višenitno programiranje
- System.Web – omogućava preglednik-server komunikaciju
- System.Web.Services – omogućava izgradnju i korištenje web usluga
- System.Windows.Forms – za korisnička sučelja u aplikacijama za Windows OS
- System.Xml – pruža podršku za obradu XML-a [5]

2.6. Microsoft Visual Studio

Microsoft Visual Studio je integrirano razvojno okruženje (*IDE – Integrated Development Environment*) Microsofta. Koristi se za razvoj računalnih programa kao što su internetske stranice, internetske aplikacije, internetske usluge te mobilne aplikacije. Visual Studio za razvoj softvera koristi Microsoftove razvojne platforme kao što su *Windows API, Windows Forms, Windows Presentation Foundation, Windows Store i Microsoft Silverlight*. Može kreirati i izvorni kod i prilagođeni kod.

Visual Studio uključuje uređivač koda koji podržava komponente završavanja koda (engl. *IntelliSense*) kao i obnavljanja koda. Osim ugrađenog ispravljača pogrešaka (engl. *debugger*), postoje i drugi ugrađeni alati kao što su dizajner formi za izradu GUI aplikacija, internetski dizajner, dizajner klasa i dizajner izgleda baza podataka. Prihvaća dodatke koji proširuju funkcionalnost na gotovo svakoj razini uključujući dodavanje podrške za sustav kontrole izvora kao što je na primjer Git, dodavanje novog seta alata kao što su uređivači i dizajneri izgleda za jezike određenog područja ili neke druge setove alata za druga stajališta softverskog razvoja informacijskog sustava.

Visual Studio podržava 36 različitih programskih jezika te omogućuje uređivaču koda i ispravljaču pogrešaka da podržavaju gotovo sve programske jezike. Ugrađeni programski jezici su C, C++, C++/CLI, Visual Basic .NET, C#, F#, JavaScript, TypeScript, XML, XSLT, HTML i CSS. Podrška ostalih jezika kao što je Python, Ruby, Node.js i M dostupna je ukoliko se instaliraju posebni dodaci. [12]

2.6.1. Visual Studio 2017.

Projekt, odnosno *Windows Forms* aplikacija, izrađena je u Visual Studio 2017. razvojnom okruženju. Objavljen je 7. travnja 2017. godine, odnosno tada je postao opće dostupan. Neke od promjena su da sadrži XAML uređivač, poboljšani *IntelliSens* i ispravljač pogrešaka, bolja produktivnost te na kraju, to je posljednja verzija Visual Studia koja podržava održavanje projekata za *Windows 10 Mobile*. [12]

3. IMPLEMENTACIJA METODE

3.1. Ideja za rješavanje problema

Rješenje problema diplomskog rada zamišljeno je korištenjem C# programskog jezika i EmguCV tehnologije s ciljem kreiranja programa koji ima mogućnost obrade slike na kojoj će biti moguća detekcija ulaska i izlaska automobila s parkirališnog mjesta. Sliku, odnosno video isječak, koji je snimljen u svrhu ovog projekta, potrebno je binarizirati radi lakše obrade, što je uspješno ostvareno korištenjem *Optical Flow* metode. *Optical Flow* metoda omogućuje detektiranje promjene između dva uzastopna okvira što je koristilo pri detektiranju ulaska i izlaska automobila. Za navedeno detektiranje omogućen je gumb *SetUp* koji pruža mogućnost postavljanja točaka koje će koristiti pri detekciji. Točke detekcije dobivene su povlačenjem linije na svakom parkirališnom mjestu koje želimo promatrati. Početna vrijednost postavljenih točaka za svako parkirališno mjesto je *free*, odnosno označava da je mjesto slobodno, ali se ta vrijednost, koja je ispisana u *ListBox*-u, može promijeniti u *occupied* klikom na tu vrijednosti ukoliko su točke postavljene na već zauzeto mjesto. Postavljena linija se dijeli na četiri jednaka dijela čime se dobije pet jednako razmaknutih točaka detekcije koje odgovarajućim redoslijednom, ovisno radi li se o ulasku ili izlasku, bilježe promjene između uzastopnih okvira te tako detektiraju promjenu zauzetosti promatranog parkirališnog mjesta. Smjer postavljanja linije je bitan i potrebno ju je postaviti od ulaska na parkirališno mjesto prema kraju.

3.2. Učitavanje video isječka

U svrhu ovog projekta kamerom je snimljen video isječak koji će se koristiti za nadzor slobodnih i zauzetih mjesta na parkiralištu pa u kodu za učitavanje navodimo samo ime video isječka, a prikaz tog koda može se vidjeti na slici 3.1.

```
capture = new VideoCapture("parking_example_1.avi");
```

Sl. 3.1. *Prikaz koda za učitavanje video isječka*

Ukoliko umjesto postojećeg video isječka, želimo prikaz s video kamere, koristi se kod prikazan na slici 3.2. Umjesto naziva video isječka, kako se piše ukoliko se želi prikazati već snimljeni video isječak, piše se redni broj kamere s koje se želi snimati, odnosno prikazati video okviri (engl. *video frames*).

```
capture = new VideoCapture("0");
```

Sl. 3.2. *Prikaz koda za učitavanje slike s video kamere*

Klikom na *Start* gumb, pokreće se video u dva *ImageBox*-a. U lijevom *ImageBox*-u pokreće se video u svojem izvornom obliku, a u desnom *ImageBox*-u pokreće se video u binarnom obliku, odnosno samo u dvije boje, crnoj i bijeloj. Prikaz koda za pokretanje video isječka u dva različita *ImageBoxa*-a prikazan je na slici 3.3. Bijela boja u videu predstavlja svaku kretnju koja se događa, dok je crna na onim pikselima kod kojih nema promjene. Za pretvorbu video isječka u binarni oblik korištena je *Dense Optical Flow* (*Gunnar Farneback)* metoda koja promatra sve točke i detektira sve piksele na kojima se događa promjena između dva uzastopna okvira čime se ostvaruje popunjenost objekta koji se kreće, odnosno ne detektira se samo na rubnim dijelovima te to olakšava detektiranje kretnje na postavljenim točkama.

```
ibStream.Image = colImage;
ibOptFlow.Image = binFlowResult;
```

Sl. 3.3. *Prikaz koda za prikazivanje video isječaka*

Video isječak se u svakom trenutku može pauzirati pritiskom istog gumba kao za pokretanje, te isto tako i pokrenuti od tog trenutka gdje se video isječak zaustavio, odnosno pauzirao. Prilikom pokretanja video isječka naziv gumba se mijenja iz *Start* u *Stop* i obratno, tj. prilikom zaustavljanja naziv gumba se mijenja iz *Stop* u *Start*. Kod na slici 3.4. prikazuje omogućeno pauziranje te ponovno pokretanje video isječka te u *if* petlji provjerava postoji li već nešto učitano što će se prikazivati te u tom slučaju samo omogući zaustavljanje video isječka i korištenje *SetUp* gumba, a ukoliko ne postoji, započinje učitavanje s video kamere te sami prikaz traje sve dok video okviri (engl. *video frames*) postoje.

```
if (capture != null)
{
    if (captureInProgress)
    {
        btnStart.Text = "Start";
        btnSetup.Enabled = true;
        lbParkData.Enabled = true;
        Application.Idle -= ProcessFrame;
     }
     else
     {
        btnStart.Text = "Stop";
        btnSetup.Enabled = false;
        lbParkData.Enabled = false;
        Application.Idle += ProcessFrame;
     }
     captureInProgress = !captureInProgress;
}
```

Sl. 3.4. *Prikaz koda za mogućnost pauziranja i ponovnog pokretanja video isječaka*

Nakon prvi put pokrenutog video isječka dobiva se mogućnost postavljanja određenih točaka koje će biti promatrane u svrhu detektiranja ulaska i izlaska automobila s parkirnog mjesta, ali video isječak se nakon prvog pokretanja mora zaustaviti kako bi se omogućilo korištenje gumba *SetUp* koji daje mogućnost postavljanja točaka. Točke se dobivaju povlačenjem linije koja se dijeli na četiri jednake dužine čime se dobije pet jednako razmaknutih točaka. Na slici 3.5. vidi se kod za kreiranje navedene linije potrebne za dobivanje točaka detekcije.

```
for (int i = 0; i < 5; i++)
{
    linePoints[i].X = ((line.P1.X - line.P2.X) / 4) * (4-i) + line.P2.X;
    linePoints[i].Y = ((line.P1.Y - line.P2.Y) / 4) * (4-i) + line.P2.Y;
    setUpImage.Draw(i.ToString(), linePoints[i], FontFace.HersheyComplex, 0.4, new Bgr(25,0,0));
}
```

Sl. 3.5. *Prikaz koda za dijeljenje linije na pet točaka za detekciju*

Smjer povlačenja linije je bitan i treba se povući u smjeru ulaska automobila na parkirališno mjesto. Linija se kreira pritiskom lijeve tipke miša pri čemu se zabilježi početak linije te se lijeva tipka miša ne pušta sve do točke na kojoj želimo zabilježiti zadnju točku, odnosno kraj linije. Nakon puštanja lijeve tipke miša, odnosno nakon kreiranja linije, u *ListBox*-u se dobije ispis stanja tog parkirnog mjesta koje je podešeno na slobodno, a klikom na to stanje može se promijeniti na zauzeto ukoliko se želi promatrati detekcija već zauzetog parkirališnog mjesta. Na slici 3.6. prikazan je kod koji omogućuje klikom na stanje u *ListBox*-u promjenu tog stanja.

```
private void lbParkData_Click(object sender, EventArgs e)
{
    if (parking[lbParkData.SelectedIndex].State == false)
    {
        parking[lbParkData.SelectedIndex].State = true;
        parking[lbParkData.SelectedIndex].SetChkVals(1);
    }
    else
    {
        parking[lbParkData.SelectedIndex].State = false;
        parking[lbParkData.SelectedIndex].SetChkVals(0);
    }

    fillList();
}
```

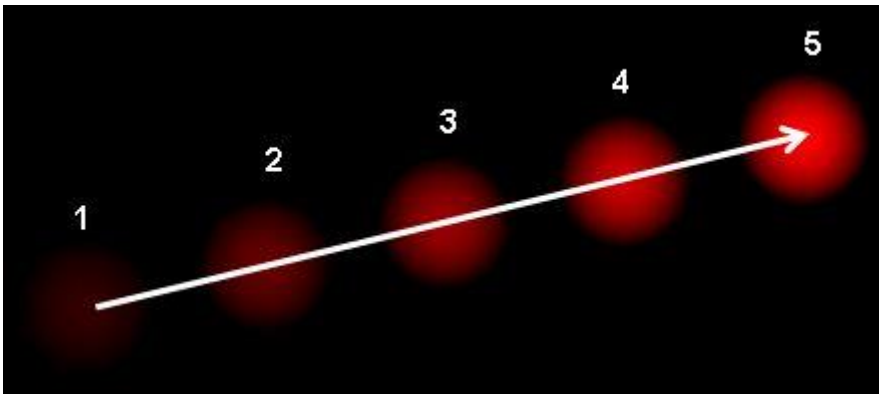
Sl. 3.6. *Prikaz koda za promjenu stanja parkirnog mjesta*

Postavljanje točaka detekcije može se obaviti samo na zaustavljenom video isječku pa je video potrebno zaustaviti kako bi se omogućilo korištenje gumba *SetUp* koji omogućuje postavljanje

navedenih točaka. Nakon postavljenih točaka zaustavljeni video je potrebno ponovno pokrenuti kako bi se započelo testiranje detekcije na njima.

3.3. Optical-flow

Za lakšu detekciju kretnji objekata u video isječcima, video je binariziran, odnosno prikazan u crno-bijeloj boji. Crna boja u video isječku predstavlja svaki piksel na kojemu se ne događa nikakva kretnja, to jest promjena okvira (engl. *frame*) u odnosu na prethodno prikazani. Suprotno crnoj, bijela boja prikazuje postojanu promjenu okvira u odnosu na prethodni, odnosno *detektira* kretnje u video isječku. Za binarizaciju video isječka u ovom projektu korištena je *Optical flow* metoda. *Optical flow* je kretnja objekata između dva uzastopna okvira video isječka uzrokovana kretnjama između objekta i kamere. To je 2D vektorsko polje gdje je svaki vektor vektor kretnje koji prikazuje kretnju točaka od prvog do drugog okvira. Slika 3.7. prikazuje kretanje lopte kroz pet uzastopnih okvira, a strelica predstavlja vektor pomaka. [13]



Sl. 3.7. *Prikaz kretnje lopte kroz pet uzastopnih okvira [13]*

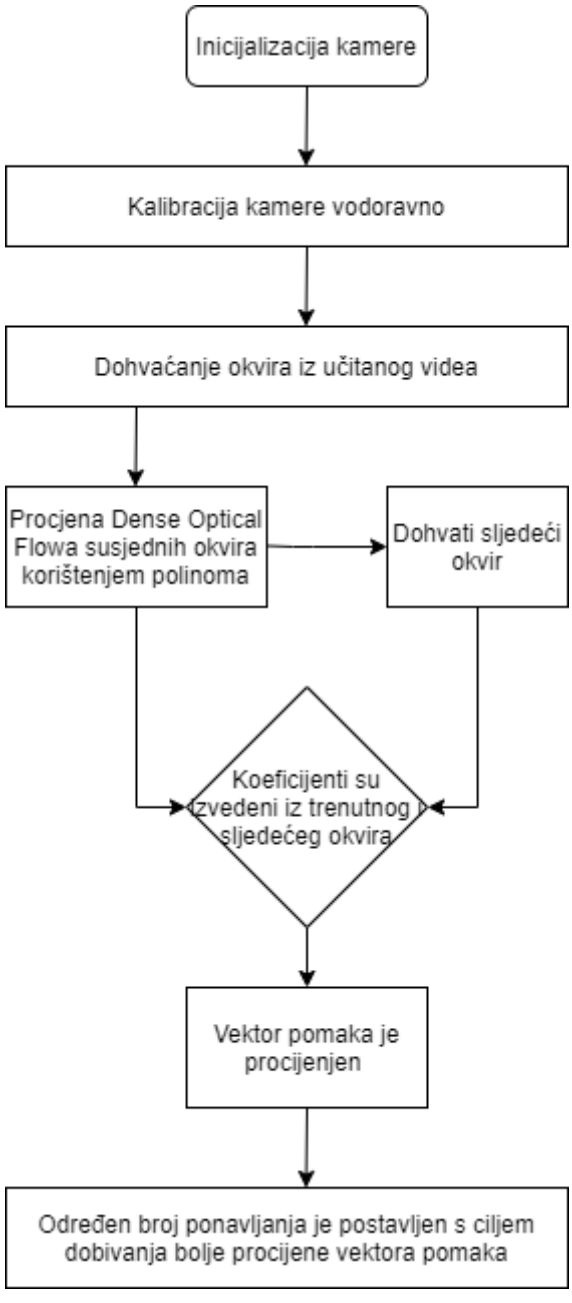
Optical flow radi na nekoliko pretpostavki:

1. Intenzitet piksela nekog objekta se ne mijenja između uzastopnih okvira
2. Susjedni pikseli imaju slične kretnje, kako je predstavljeno u [13]

Postoje dvije vrste *Optical Flow-a*, a to su *Sparse Optical Flow (Lucas Kanade)* metoda i *Dense Optical Flow (Gunnar Farneback)* metoda koja se koristi u ovom projektu te će biti objašnjena u daljnjem tekstu.

Dok se u *Lucas Kanade* metodi gledaju samo kutne točke objekata, odnosno računa se *optical flow* samo za set rijetkih značajki piksela, u *Gunnar Farneback* metodi se gledaju sve točke, i detektiraju se pikseli na kojima se događa nekakva promjena između dva uzastopna okvira, odnosno računa se *optical flow* vektor za svaki piksel svakog *frame-a* i to rezultira slikom s naglašenim pikselima. Dok takav izračun može biti sporiji, ipak daje točniji i primjereniji rezultat

za projekte kao što je ovaj. Postoje različite implementacije *Dense Optical Flow-a*, a u projektu se koristi *Farneback* metoda, jedna od najpopularnijih implementacija. Metoda računa magnitudu i smjer *optical flow-a* iz polja vektora, a kasnije pretpostavlja smjer *flow-a* prema nijansi i magnitudu, odnosno smjer *flow-a* prema vrijednosti HSV-a. HSV (*hue, saturation, value*) je alternativna reprezentacija RGB modela boja. Za optimalnu vidljivost, HSV vrijednost postavlja se na 255. OpenCV biblioteka omogućuje *CalcOpticalFlowFarneback* funkciju za primjenu *Dense Optical Flow-a*. Primjer rada *Gunnar Farneback* algoritma prikaza je slikom 3.8. [14]

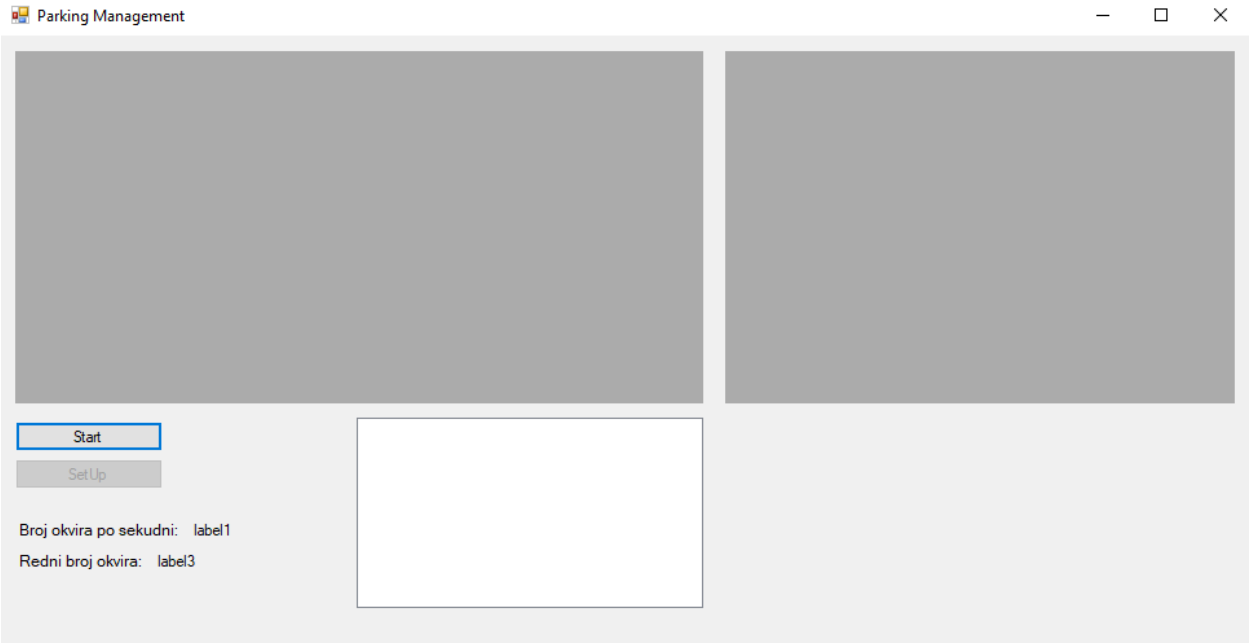


Sl. 3.8. *Dijagram toka za Gunnar Farneback algoritam kako je predstavljeno u [15]*

3.4. Detekcija ulaska/izlaska

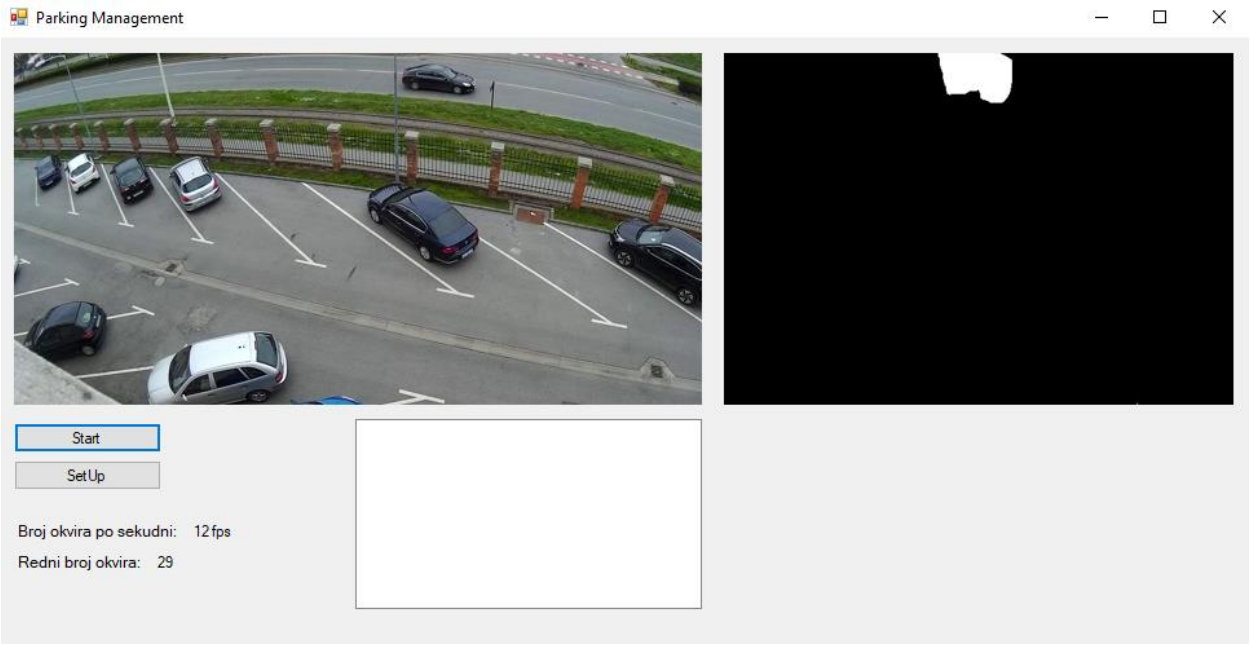
Rješenje problema detekcije ulaska/izlaska zamišljeno je korištenjem *Optical Flow* metode koja promatra sve točke svakog okvira video isječka te detektira svaki piksel na kojem se događa nekakva promjena između dva uzastopna okvira. *Optical Flow Farneback* metoda definirana je u funkciji *ProcessFrame* u kojoj se prati svaki okvir video isječka te se u svakom trenutku provjerava dolazi li do nekakve promjene na pet točaka detekcije koje su unaprijed definirane na svakom parkirališnom mjestu koje se promatra.

Prilikom pokretanja izrađene aplikacije dobije se prikaz dva *ImageBox*-a na kojima će se prikazati video isječak nakon njegovog pokretanja (lijevo u svojem izvornom obliku, a desno u binariziranom obliku), prikaz *ListBox*-a za ispis stanja promatranih parkirališnih mjesta, dva gumba (jedan za pokretanje i zaustavljanje video isječka, jedan za postavljanje točaka detekcije) i dvije oznake koje prikazuju broj okvira po sekundi te redni broj okvira. Slika 3.9. prikazuje aplikaciju prije prvog pokretanja video isječka, sadrži sve navedene elemente, ali video isječak još nije prikazan ni u jednom *ImageBox*-u.



Sl. 3.9. *Prikaz aplikacije prije pokretanja video isječka*

Nakon pokretanja video isječka vidi se učitani video isječak u svojem izvornom obliku te u binariziranom obliku (Sl.3.10.).



Sl. 3.10. *Prikaz aplikacije nakon pokretanja video isječka*

Nakon pokretanja video isječka prikazuje se broj okvira koji se prikazuju u jednoj sekundi (engl. *frames per second* - *fps*) te se prikazuje redni broj okvira koji se neprestano mijenja. Oba broja ispisuju se pomoću oznake (engl. *label*) koja se dodala u *Windows Formu*.

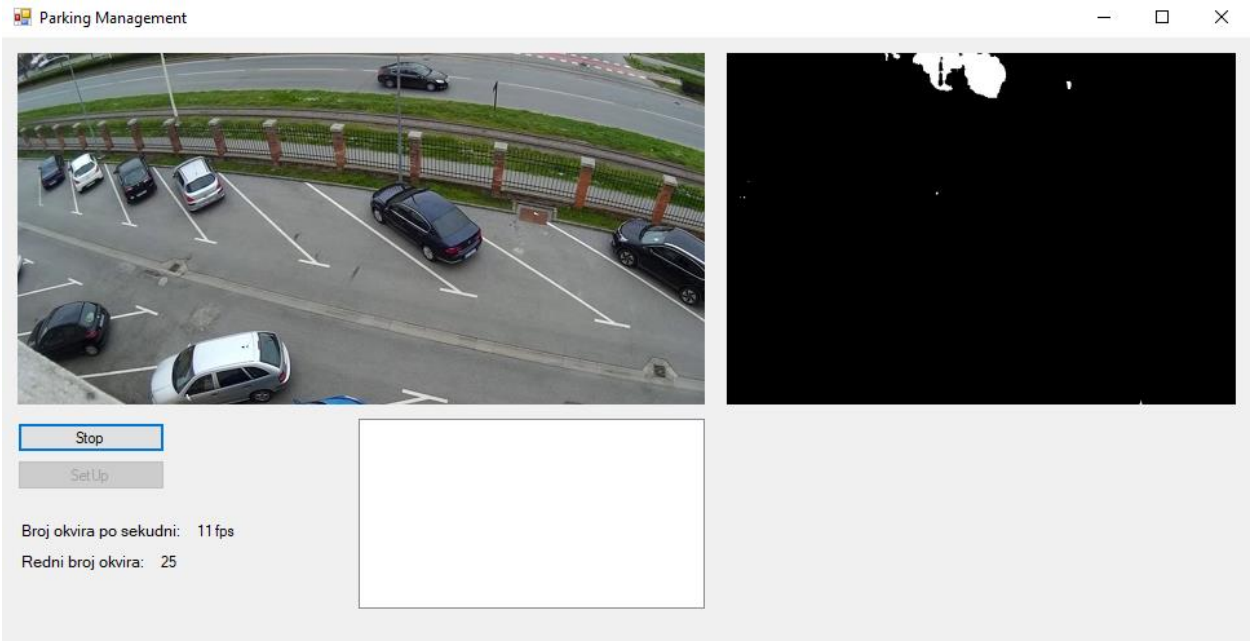
U projektu se promatra i ulazak i izlazak automobila na parkirališnim mjestima. Detekcija ulaska, odnosno izlaska se obavlja promatranjem pet postavljenih točaka koje su se dobile dijeljenjem povučene linije na četiri jednaka dijela. Točke dobivene povlačenjem linije su točke koje se provjeravaju jedna po jedna te se tako detektira ulazak odnosno izlazak automobila. Smjer povlačenja linije je bitan zbog provjere ulazi li automobil ili izlazi s parkirališnog mjesta te se povlači od mjesta ulaska prema kraju parkirnog mjesta. Smjer je bitan jer se točke provjeravaju odgovarajućim redoslijedom. Ukoliko se provjerava ulazak automobila, točke se provjeravaju redom od 0 do 4. Prva točka koja se provjerava je točka 0, provjeri se intenzitet boje koji se pojavljuje na toj točki te vrijednost te točke. Ukoliko je intenzitet 255, odnosno, ukoliko se na točki pojavila bijela boja te ukoliko je vrijednost točke 0, potrebno je povećati tu točku za jedan, to jest, vrijednost točke mijenja se iz nula u jedan. Nakon točke nula provjerava se točka jedan gdje se ponovno provjerava intenzitet točke jedan, provjerava se vrijednost točke jedan, ali i vrijednost točke nula. Ukoliko je intenzitet točke 1 255, vrijednost točke 1 nula te vrijednost točke 0 jedan, točka 1 se povećava za 1, odnosno vrijednost se mijenja iz nula u jedan. Ista provjera obavlja se nas svim točkama, provjera intenziteta boje, vrijednost trenutne točke i vrijednost prethodne točke. Na slici 3.11. može se vidjeti kod za provjeru, to jest popunjavanje točaka detekcije, prikazan je kod za provjeru prve tri točke.

```
if (image[parking[i].chkPoints[0]].Intensity == 255 && parking[i].chkPointsVal[0] == 0)
{
    parking[i].chkPointsVal[0]++;
}
else
{
    if (image[parking[i].chkPoints[1]].Intensity == 255 && parking[i].chkPointsVal[1]
        == 0 && parking[i].chkPointsVal[0] == 1)
    {
         parking[i].chkPointsVal[1]++;
    }

    else
    {
         if (image[parking[i].chkPoints[2]].Intensity == 255 &&
             parking [i].chkPointsVal[2] == 0 && parking[i].chkPointsVal[1] == 1)
         {
              parking[i].chkPointsVal[2]++;
         }

     }

}
```

Sl. 3.11. *Prikaz koda za provjeru prve tri točke pri ulasku automobila*

Ukoliko se provjerava izlazak automobila s parkirališnog mjesta, provjera točka obavlja se jednako, ali drugim redoslijedno. Provjeravaju se točke od 4 do 0 i vrijednost točaka se smanjuje za jedan, to jest mijenja se iz jedan u nula.

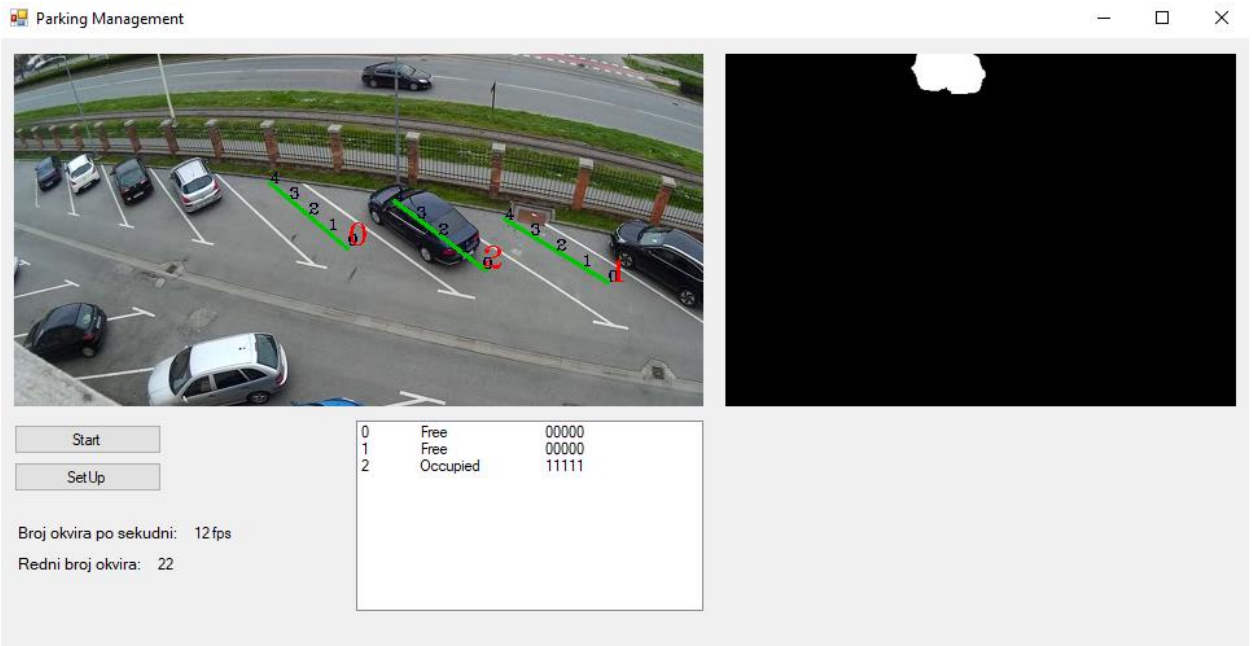
Redni broj označenog parkirališnog mjesta, status parkirališnog mjesta (*free* ili *occupied*) te pet točaka se ispisuje u *ListBoxu*. Praćenje tih točaka može se obaviti promatranjem navedenog *ListBox*-a. Točke mogu biti nule i jedinice. Ukoliko je svih pet točaka nula, znači da je parkirališno mjesto slobodno, a to može potvrditi i status *free*, a ukoliko je svih pet točaka jedan, mjesto je zauzeto, odnosno status je *occupied*. Linija se postavlja nakon zaustavljanja video isječka jer je samo u toj situaciji omogućeno korištenje gumba *SetUp* koji omogućuje postavljanje upravo spomenute linije. Na slici 3.11. može se vidjeti onemogućeni *SetUp* gumb jer je video isječak u tijeku, odnosno nije zaustavljen. Nakon povučenih linija na svim parkirališnim mjestima koja se želi promatrati, odnosno na kojima se želi obaviti detekcija ulaska i izlaska, potrebno je ponovno pokrenuti video isječak klikom na gumb *Start* kako bi sama detekcija mogla započeti.



Sl. 3.12. *Prikaz onemogućenog SetUp gumba*

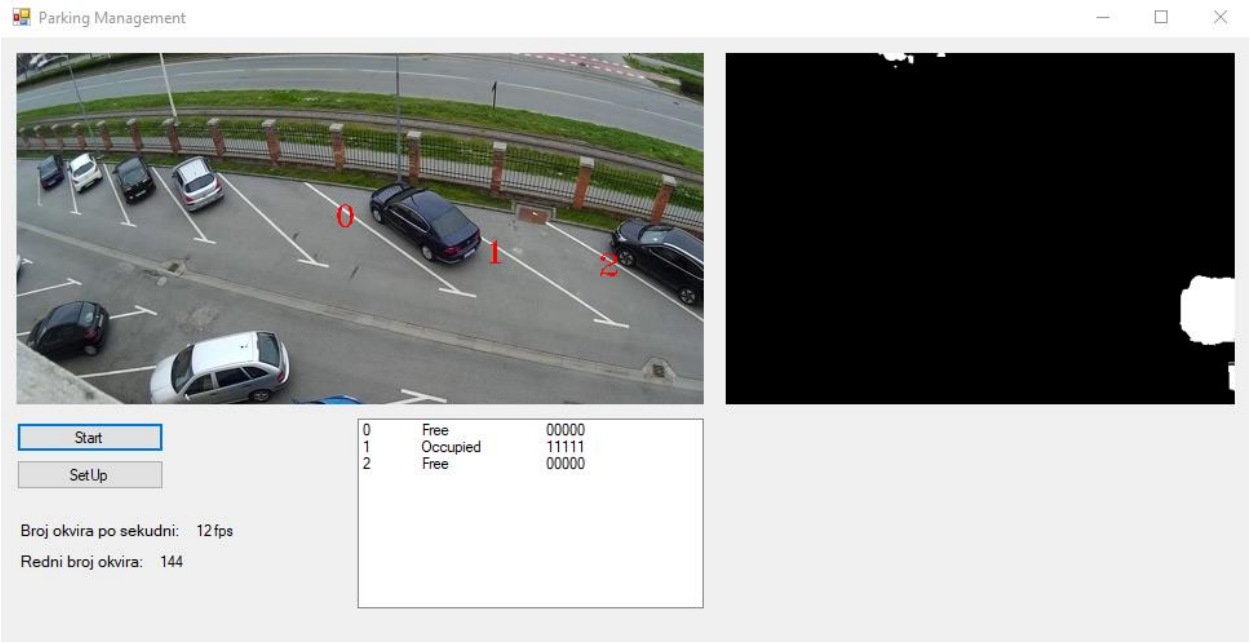
Sl. 3.13. *Prikaz povučenih linija te ispisanih stanja u ListBoxu*

Povučene linije predefinirano imaju status *free*, a ukoliko se želi promatrati već zauzeto parkirališno mjesto, klikom na status u *ListBox*-u za željeno parkirališno mjesto može se promijeniti status iz *free* u *occupied*. Na slikama 3.13. i 3.14. može se vidjeti primjer povlačenja linija te promjena jednog stanja u *ListBox*-u nakon što se kliknulo na isto.



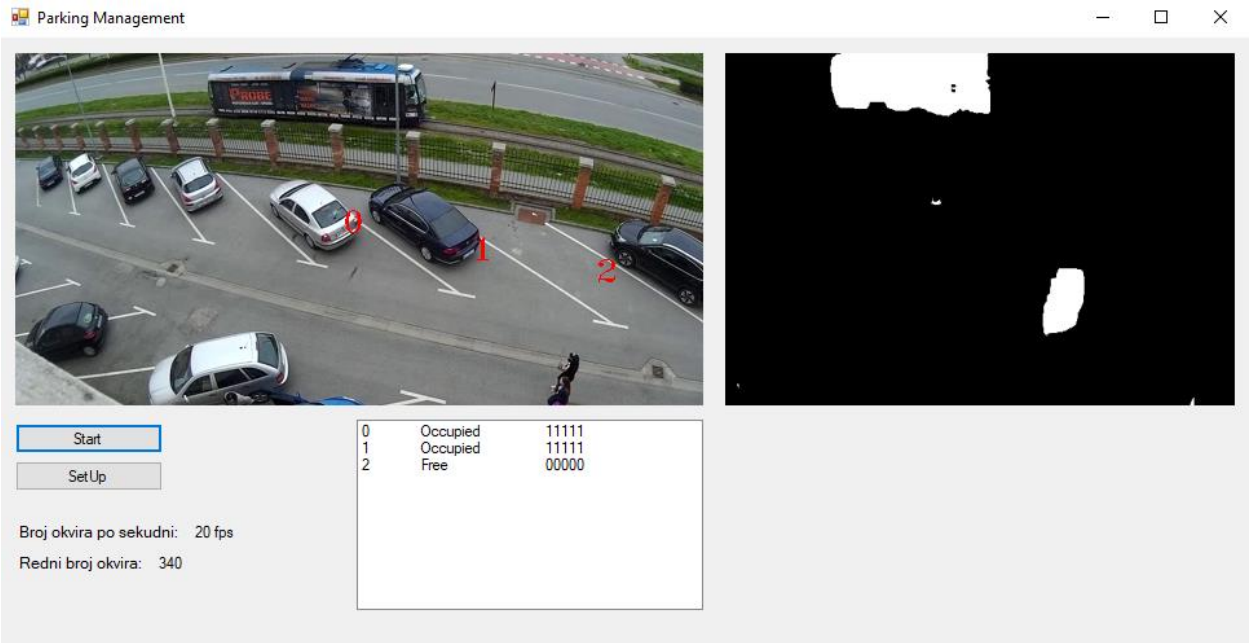
Sl. 3.14. *Promjena stanja već zauzetog parkirališnog mjesta (parkirališno mjesto broj 2)*

Za lakšu detekciju, obrada se vrši na binariziranom video isječku, desno učitanom. Binarizacija video isječka ostvarena je *Dense Optical Flow (Gunnar Farneback)* metodom kojom se gledaju sve točke i detektiraju se pikseli na kojima se događa nekakva promjena između dva uzastopna okvira, odnosno računa se *optical flow* vektor za svaki piksel svakog *frame-a* i to rezultira slikom s naglašenim pikselima. Na svakom parkirališnom mjestu na kojem se želi obavljati detekcija postavlja se pet točaka od kojih svaka prati promjenu boje koja se događa na mjestu na kojem je postavljena. Promjena boje, odnosno svaki piksel bijele boje označava promjenu između dva uzastopna okvira, odnosno označava nekakvu kretnju u video isječku. Prilikom ulaska automobila na parkirališno mjesto prati se promjena boje na točkama redom od nula do četiri i ukoliko svaka točka tim redom promijeni svoje stanje iz nula u jedan, detektira se ulazak automobila te stanje parkirališnog mjesta postaje *occupied* ili zauzeto. Upravo zbog tog redoslijeda kojim se detektira promjena na točkama, odnosno detektira ulazak ili izlazak automobila, bitan je smjer povlačenja linije kojom se dobije pet spomenutih točaka. Prilikom izlaska automobila s parkirališnog mjesta prati se promjena boje na točkama obrnutim redoslijedom, odnosno redom od četiri do nula i ukoliko svaka točka tim redom promijeni svoje stanje iz jedan u nula, detektira se izlazak automobila te se stanje parkirališnog mjesta koje je bilo *occupied* mijenja u *free*.

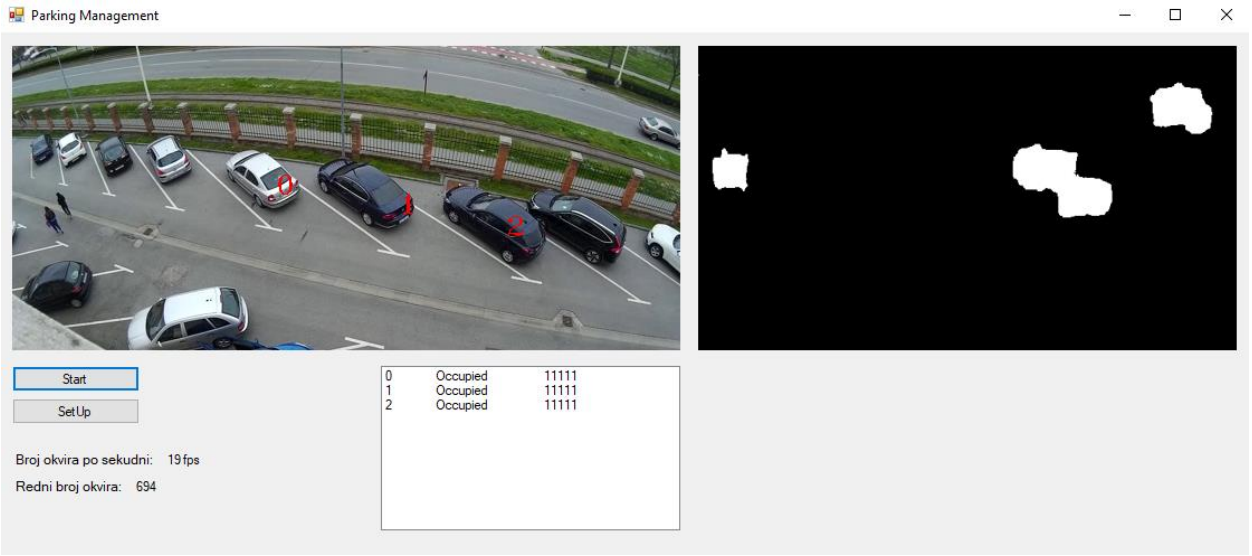


Sl. 3.15. *Prikaz stanja promatranih mjesta prije ulaska prvog automobila*

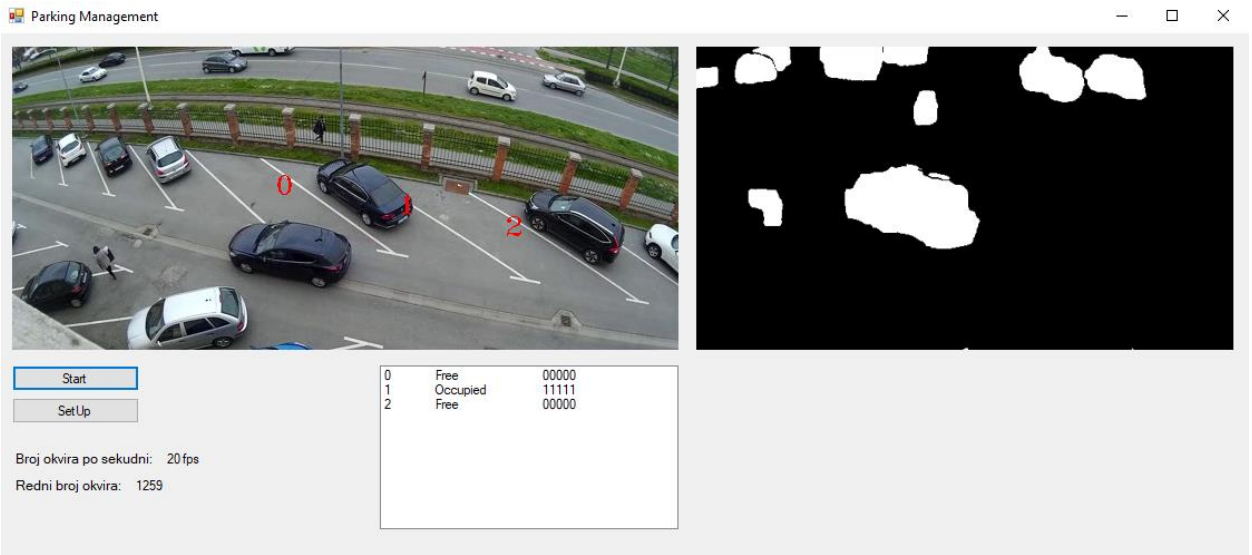
Na slici 3.15. može se vidjeti ispis stanja za tri parkirališna mjesta na kojima su postavljane točke detekcije. Za parkirališna mjesta 0 i 2, stanje je *free* što je i prethodno definirano stanje za svako mjesto, dok je za parkirališno mjesto 1 stanje *occupied* što se izmijenilo lijevim klikom miša na stanje ispisano u *ListBox*-u. Promjenom stanja *free* u *occupied*, mijenjaju se i točke iz 0 u 1 jer jedinice označavaju zauzeto parkirališno mjesto.



Sl. 3.16. *Prikaz stanja promatranih mjesta nakon ulaska prvog automobila*



Sl. 3.17. *Prikaz stanja promatranih mjesta nakon ulaska drugog automobila*



Sl. 3.18. *Prikaz stanja promatranih mjesta nakon izlaska oba automobila*

Na slikama 3.16., 3.17. i 3.18. može se vidjeti promjena stanja prilikom ulaska i izlaska automobila. Prilikom ulaska automobila na parkirališno mjesto stanje koje je prethodno bilo *free* prelazi u stanje *occupied* i nule se prebacuju u jedinice što također označava zauzeto parkirališno mjesto. Nule se prebacuju u jedinice jedna po jedna redom točkama od nula do pet. Na prvoj slici (Sl. 3.16.) vidi se promjena stanja nakon ulaska automobila na parkirališno mjesto 0 (dva mjesta su *occupied*, jedno mjesto je *free*), na drugoj slici (Sl. 3.17.) vidi se promjena stanja nakon ulaska automobila na parkirališno mjesto 2 (tri mjesta su *occupied*, nijedno mjesto nije *free*), te na trećoj slici (Sl. 3.18.) vidi se promjena stanja nakon izlaska oba automobila koja su prethodno ušla na parkirališna mjesta 0 i 2 (stanja se vraćaju na početna, jedno mjesto je *occupied*, dva mjesta su *free*).

4. TESTIRANJE

U svrhu ovog projekta snimljena su dva video isječka na kojima su se obavljala različita testiranja aplikacije. Testirala se preciznost detekcije ulaska i izlaska s parkirališnog mjesta te brzina obrade video isječka, odnosno broj okvira koji se prikažu u jednoj sekundi pri nadzoru manjeg i većeg broja parkirališnih mjesta.

4.1. Preciznost

Testiranje projekta obavljalo se provjerom preciznosti koju program ostvaruje detektiranjem ulaska automobila na parkirališno mjesto i izlaska s parkirališnog mjesta.

Tab. 1. *Preciznost detekcija ulaska i izlaska automobila – video isječak 1*

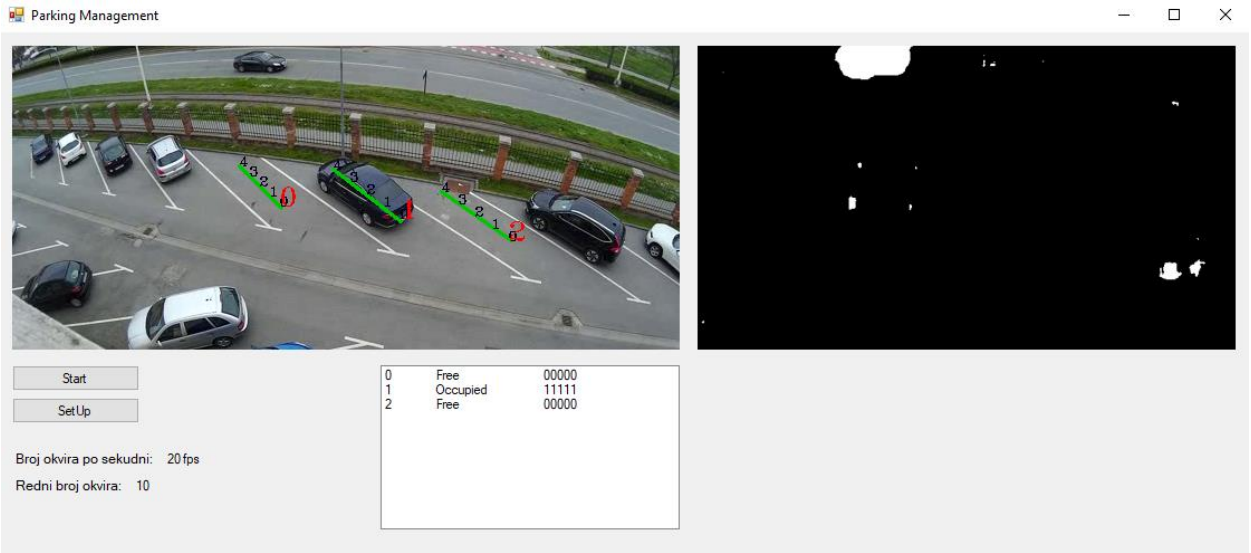
	Auto br. 1		Auto br. 2	
	Ulaz	Izlaz	Ulaz	Izlaz
Video isječak 1	✓	✓	✓	✓

Tab. 2. *Preciznost detekcija ulaska i izlaska automobila – video isječak 2*

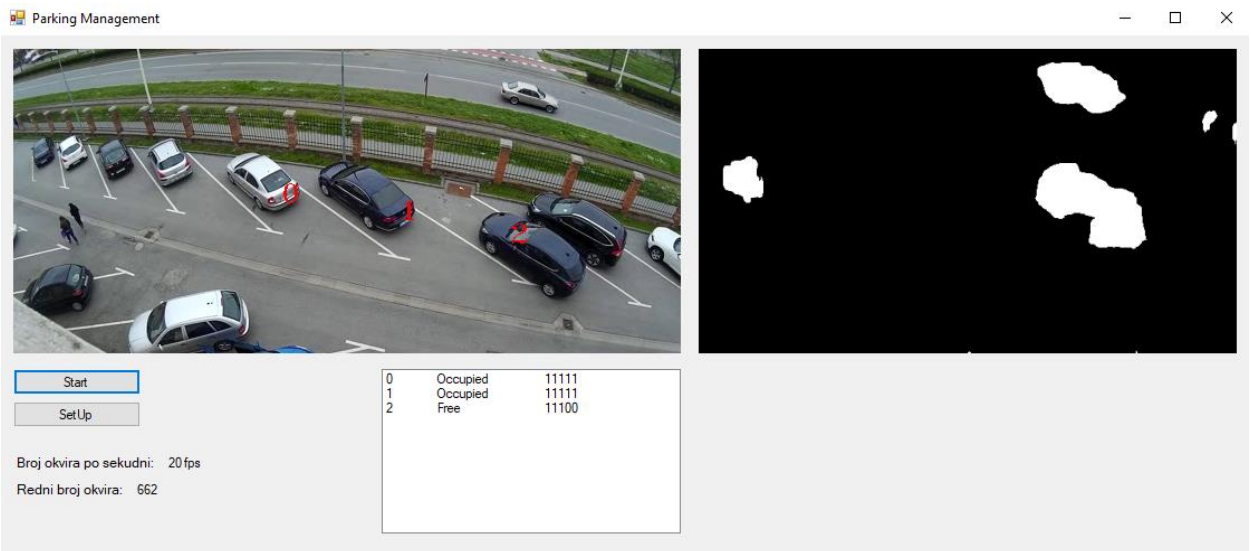
	Auto br. 1	Auto br. 2	Auto br. 3
	Ulaz	Izlaz	Izlaz
Video isječak 2	✗	✓	✓

Tablice 4.1. i 4.2. prikazuju testiranje preciznosti detekcije ulaska i izlaska automobila. U video isječku 1 se prate ulazak i izlazak dva automobila dok se u video isječku 2 prati ulazak jednog, a izlazak dva automobila. Za svaku uspješnu detekcija ulaska ili izlaska stavljen je simbol ✓ koji označava potvrdu uspješne detekcije, dok je svaka neuspješna detekcija zabilježena simbolom ✗. Iz tablice se može vidjeti da je preciznost za video isječak 1 potpuna, odnosno ostvaruje se 100% točna detekcija, a za video isječak 2, preciznost nije potpuna zbog različitih faktora koji utječu na detekciju, odnosno ostvaruje se 66,66%. U video isječku 2, faktori koji utječu na detekciju, odnosno faktori koji smanjuju postotak preciznosti su kvaliteta video isječka, dobra preglednost parkirališnog mjesta koje se promatra (udaljenije parkirališno mjesto od postavljene kamere u odnosu na druga dva promatrana) te rasvjetni stup koji je statičan i zbog kojeg se ne detektira nikakva promjena na mjestu u okviru na kojem je on.

Preciznost se može potvrditi i sljedećim slikama na kojima se u *ListBox*-u može vidjeti ispis stanja parkirališnih mjesta prije ulaska automobila i nakon izlaska automobila s parkirališnog mjesta.

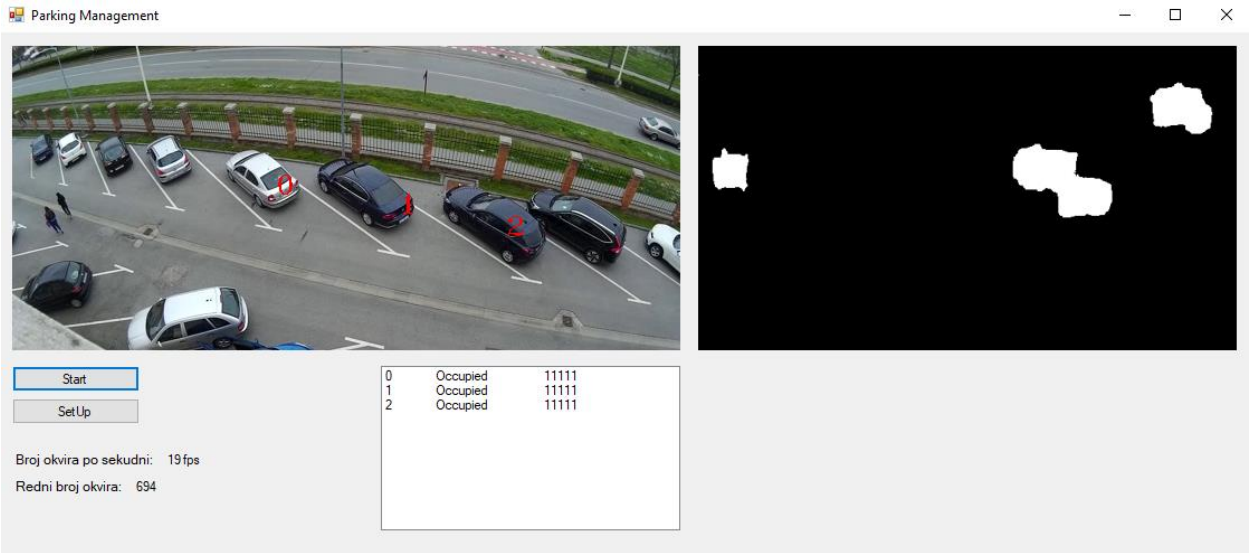


Sl. 4.1. *Prikaz stanja promatranih mjesta prije ulaska prvog automobila – video isječak 1*



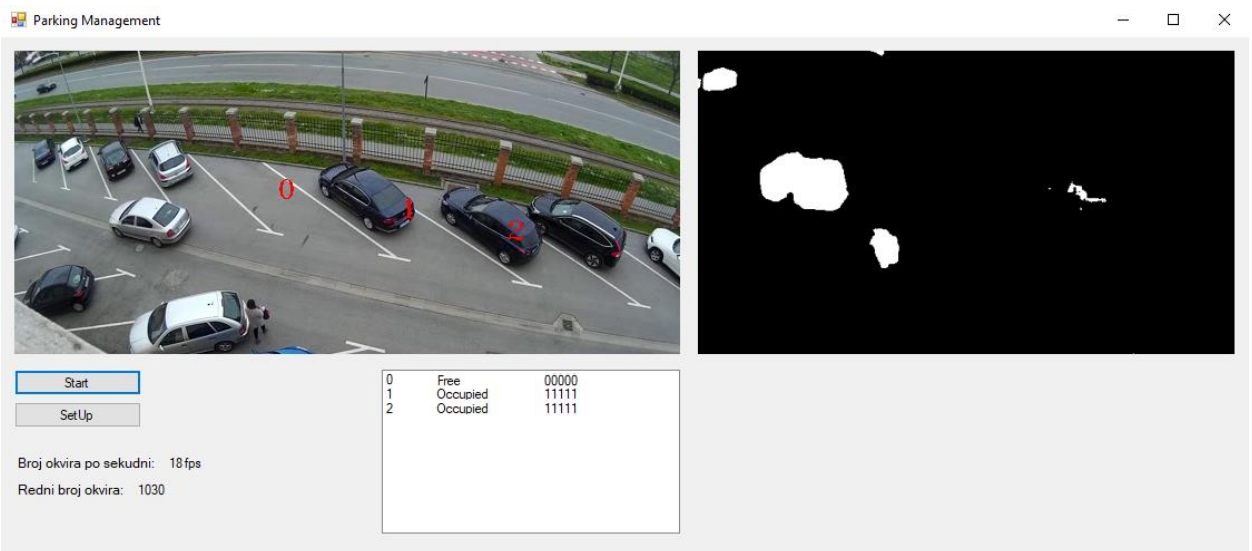
Sl. 4.2. *Prikaz trenutka ulaska drugog automobila – video isječak 1*

Na slici 4.2. može se vidjeti trenutak u kojem automobil još uvijek ulazi na parkirališno mjesto te se u *ListBox*-u stanje još nije promijenilo u *occupied*, ali se može vidjeti kako su se tri točke promijenile iz nula i jedan što znači da su prve tri točke detektirale promjenu.

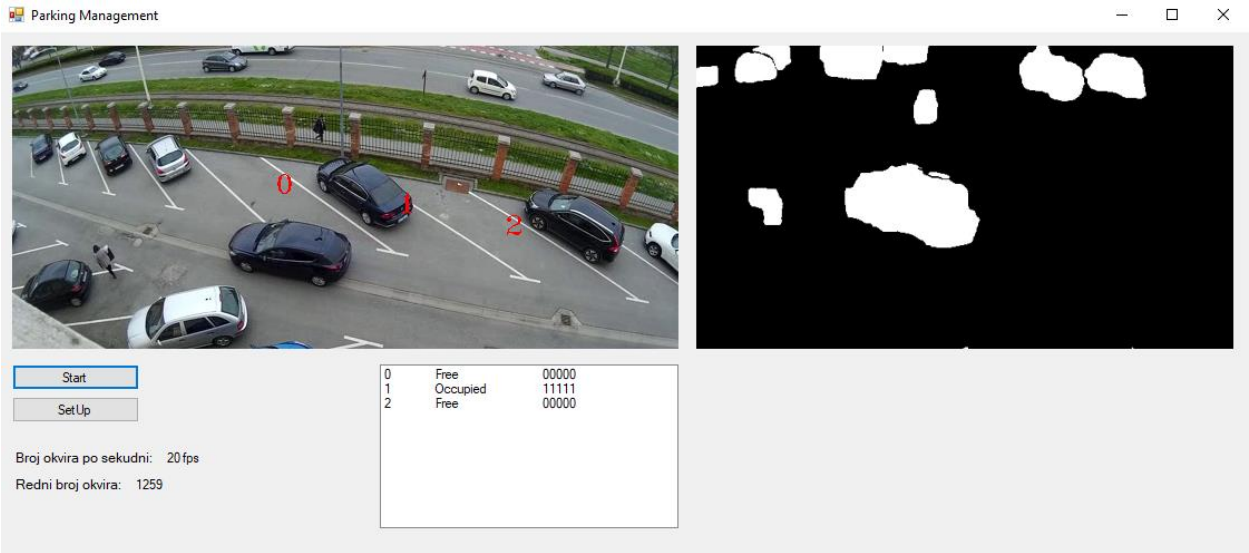


Sl. 4.3. *Prikaz stanja promatranih mjesta nakon ulaska oba automobila -video isječak 1*

Na slici 4.3. potvrđuje se preciznost zabilježene detekcije u tablici za ulazak oba automobila (parkirališna mjesta 0 i 2) ispisom stanja u *ListBox*-u koja, nakon ulaska automobila, postaju *occupied*. Navedena dva parkirališna mjesta su prije ulaska bila *free* što se može potvrditi slikom 4.1.



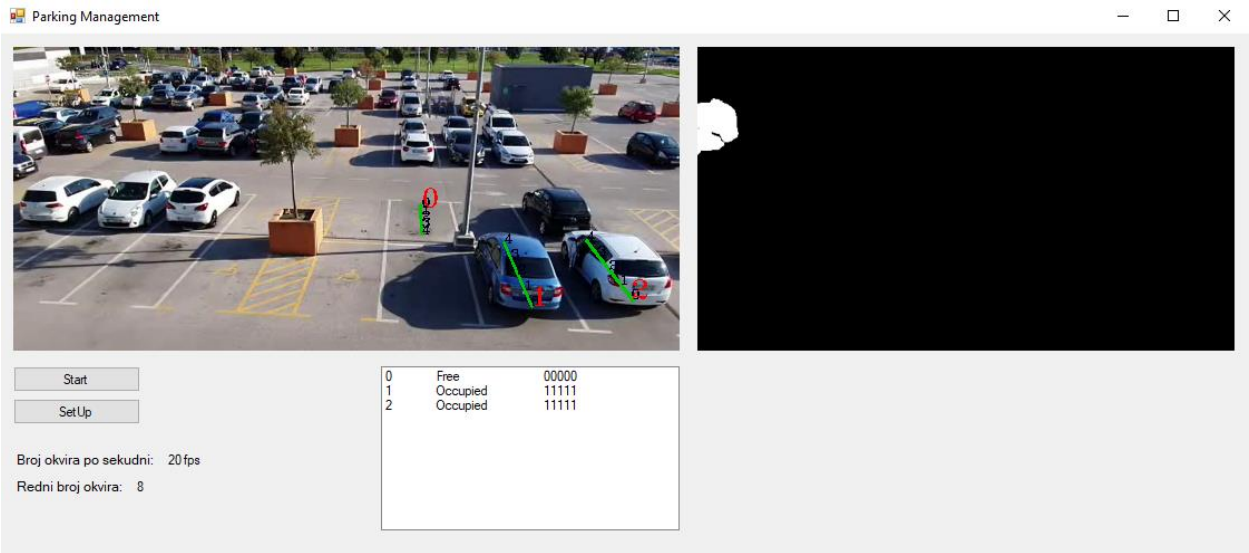
Sl. 4.4. *Prikaz stanja promatranih mjesta nakon izlaska prvog automobila – video isječak 1*



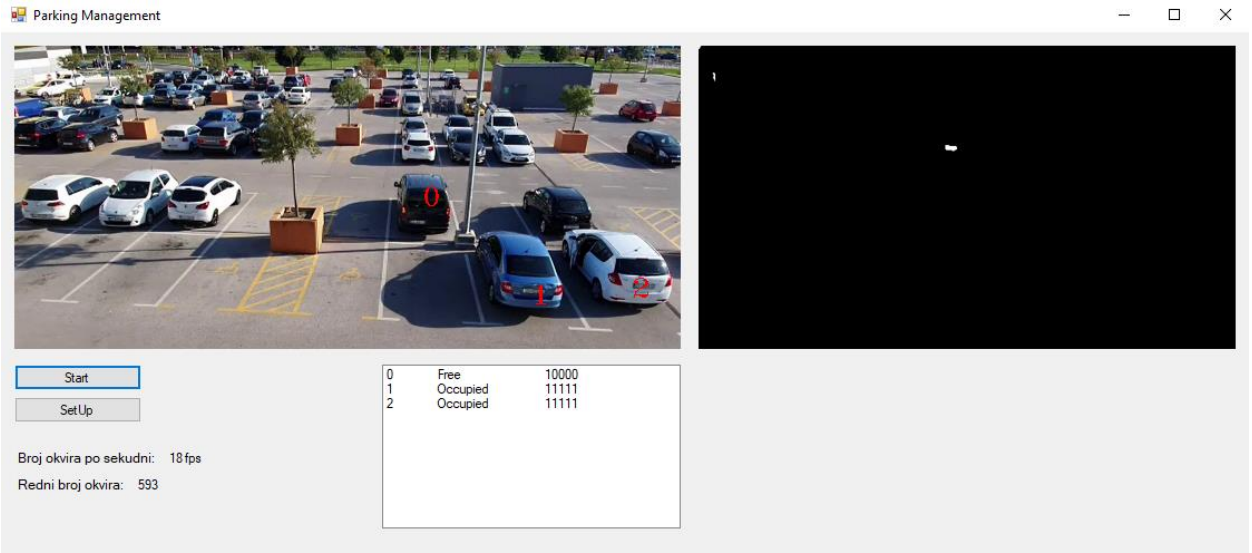
Sl. 4.5. *Prikaz stanja promatranih mjesta nakon izlaska drugog automobila – video isječak 1*

Preciznost detekcije izlaska automobila, također zabilježene u tablici, može se potvrditi slikama 4.4. i 4.5. gdje se ispis stanja u ListBox-u mijenja nakon izlaska automobila s parkirališnih mjesta 0 i 2 iz *occupied* u *free* te jedinice uz stanje *occupied* postaju nule uz stanje *free*.

Testiranje preciznosti na drugom video isječku nije bilo uspješno kao na prvom. Za razliku od prvog video isječka, preciznost detekcije u drugom je nešto manja zbog različitih faktora kao što su kvaliteta video isječka, udaljenost promatranog parkirališnog mjesta i slično.

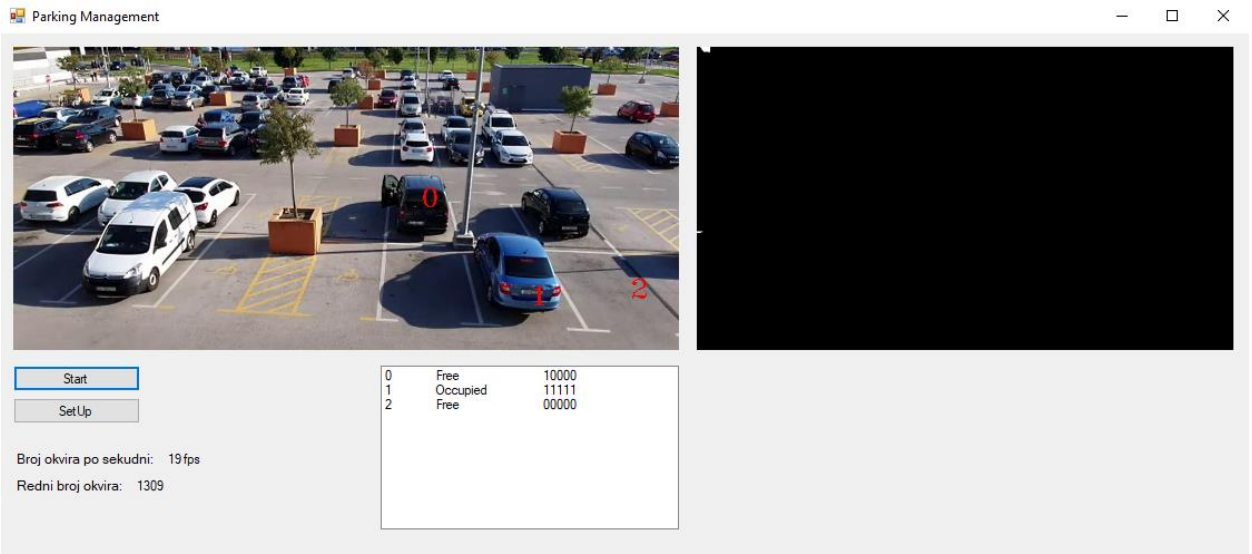


Sl. 4.6. *Prikaz stanja promatranih mjesta prije testiranih promjena – video isječak 2*

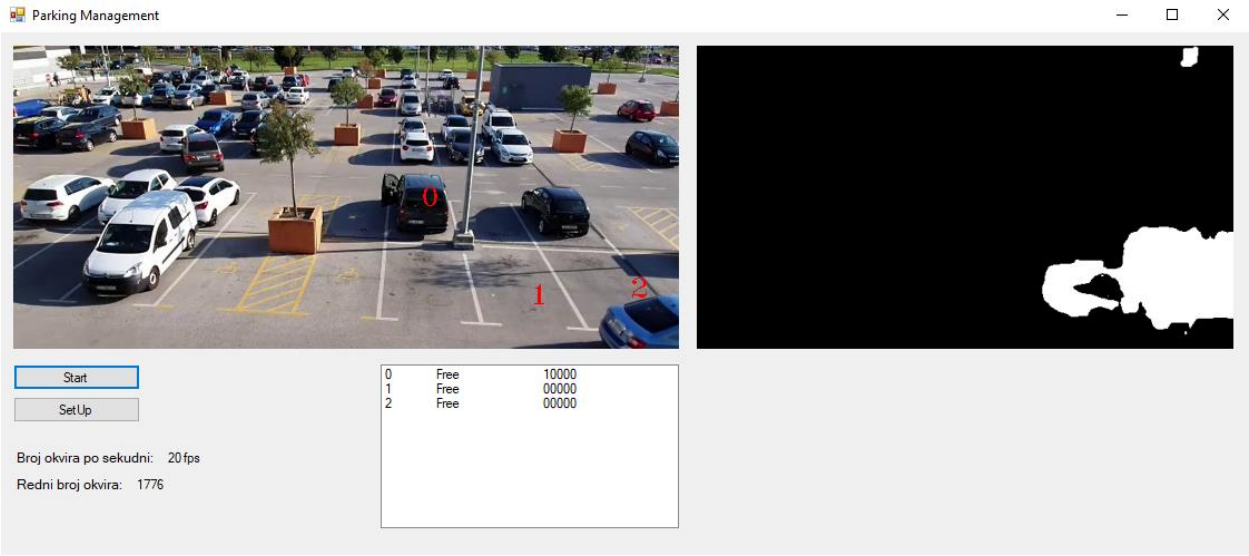


Sl. 4.7. *Prikaz stanja nakon ulaska automobila (parkirališno mjesto 0) – video isječak 2*

Na slici 4.7. promatra se ulazak automobila na parkirališno mjesto 0 te prikaz promjene stanja u *ListBox*-u nakon ulaska. Program nije uspješno detektirao ulazak automobila na parkirališno mjesto što se vidi iz prikazanog stanja koje se nije promijenilo iz *free* u *occupied* te se nule nisu promijenile u jedinice. Uspješno je detektirana promjena između dva uzastopna okvira samo na jednoj od pet točaka koje su postavljene u svrhu detektiranja ulaska automobila. Navedena neuspješna detekcija je uzrokovana zbog različitih faktora ranije spomenutih u poglavlju te se i na slici 4.6. može vidjeti postavljanje točaka detekcije koje su slabije vidljive od točaka postavljenih na parkirališnim mjestima 1 i 2 koji su bliže statičnoj kameri.



Sl. 4.8. *Prikaz stanja nakon izlaska automobila (parkirališno mjesto 2) – video isječak 2*



Sl. 4.9. *Prikaz stanja nakon izlaska automobila (parkirališno mjesto 1) – video isječak 2*

Za razliku od detektiranja ulaska automobila na parkirališno mjesto 0 koje je bilo neuspješno, detektiranje izlaska automobila s parkirališnih mjesta 1 i 2 bilo je uspješno što se može i vidjeti na slikama 4.8. i 4.9. gdje se ispisuje promjena stanja nakon izlaska automobila iz *occupied* u *free* te se uz stanje jedinice mijenjaju u nule na svim postavljenim točkama na pojedinom parkirališnom mjestu.

4.2. Brzina obrade

Testiranje projekta obavljalo se i provjerom brzine obrade video isječka, odnosno brojem prikazanih okvira po sekundi (engl. *frames per second* – *fps*). Projekt se testirao na dva različita prijenosna računala od kojih je Lenovo prijenosno računalo sljedećih specifikacija:

- Procesor: Intel Core i3-6100U CPU 2.3GHz
- RAM: 4.00GB
- Integrirana grafička kartica: Intel HD Graphics 520,

te Asus prijenosno računalo sljedećih specifikacija:

- Procesor: Intel Core i5-8250U CPU 1.6GHz
- RAM:8.00GB
- Integrirana grafička kartica: Intel UHD Graphics 620

Brzina se iskazuje u broju okvira po sekundi (fps) i taj broj se ispisuje na oznaku (engl. *label*) u stvarnom vremenu (Sl. 4.10.).

```
lblInfo.Text = (1000 / mpf.ElapsedMilliseconds).ToString() + " fps";
```

Sl. 4.10. *Ispis informacije o broju prikazanih okvira po sekundi*

Osim broja okvira po sekundi, na dodatnoj oznaci ispisuje se i redni broj okvira koji se također mijenja u stvarnom vremenu prikazivanja video isječka (Sl. 4.11.).

```
lblInfo2.Text = frameNum.ToString();
```

Sl. 4.11. *Ispis informacije o rednom broju prikazanog okvira*

Sl. 4.12. *Prikaz broja okvira po sekundi – testirano za tri parkirališna mjesta na Lenovo prijenosnom računalu*

Sl. 4.13. *Prikaz broja okvira po sekundi – testirano za tri parkirališna mjesta na Asus prijenosnom računalu*

Na slikama 4.12. i 4.13. prikazano je testiranje brzine obrade video isječka na dva različita prijenosna računala te se promatraju se samo tri parkirališna mjesta. Iz navedenih slika te specifikacija prijenosnih računala, navedenih ranije u poglavlju, može se zaključiti da brzina obrade video isječka, to jest broj prikazanih okvira uvelike ovisi o računalu na kojem je aplikacija pokrenuta. Što je veći broj okvira prikazanih po sekundi, znači da je obrada brža te prema tome sama aplikacija ima bolju funkcionalnost.

Sl. 4.14. *Prikaz broja okvira po sekundi – testirano za osam parkirališnih mjesta na Lenovo prijenosnom računalu*

Sl. 4.15. *Prikaz broja okvira po sekundi – testirano za osam parkirališnih mjesta na Asus prijenosnom računalu*

Na slikama 4.14. i 4.15. prikazano je testiranje brzine obrade video isječka na dva različita prijenosna računala te se promatra sedam parkirališnih mjesta. Iz navedenih slika može se vidjeti da se brzina ne razlikuje uvelike od brzina pri promatranju samo tri parkirališna mjesta (broj okvira po sekundi na slikama 4.12. i 4.13.)

5. ZAKLJUČAK

Zadatak diplomskog rada bio je izraditi aplikaciju koja će detektirati ulazak i izlazak automobila na parkirališno mjesto i bilježiti tu detekciju u stvarnom vremenu. U izrađenoj aplikaciji korištena su dva video isječka na kojima se obavlja testiranje detekcije ulaska i izlaska automobila s parkirališnog mjesta. Za lakšu obradu video isječka, odnosno kako bi se omogućio lakši nadzor parkirališnih mjesta, koristi se metoda *Dense Optical Flow (Gunnar Farneback)* koja promatra svaku točku i detektira svaki piksel na kojemu se događa promjena između dva uzastopna okvira što omogućava popunjenost objekta koji se kreće, u ovom slučaju je to automobil, te to olakšava detektiranje kretnje preko točaka detekcije.

Ovakvo okruženje i ovakav pristup daju mogućnost nadzora bilo kojeg parkirališta. Uspješno je ostvaren cilj prikazivanja stanja parkirališnih mjesta u stvarnom vremenu, odnosno da se stanja mijenjaju kako automobili ulaze ili izlaze s parkirališnih mjesta. Iako je navedeni cilj uspješno ostvaren, projekt, kako je i testiranjem dokazano, ima svoje mane te ne radi savršeno. Različiti faktori, kao što je na primjer loša kvaliteta videa, koja stvara različite šumove, loše utječe na izvedbu projekta jer otežavaju proces binarizacije koji bi trebao omogućiti upravo suprotno. Nadalje, udaljenost parkirališnog mjesta može loše utjecati na izvedbu zbog slabije vidljivosti postavljenih točaka detekcije. Osim problematičnosti koje stvara kamera što zbog svoje pozicije, što zbog svoje kvalitete, loše rezultate može uzrokovati bilo kakva smetnja (npr. čovjek, životinja) koja uzrokuje promjenu na nekoj od početnih točaka detekcije koja se ne odnosi na automobil koji će možda idućeg trenutka započeti ulazak na parkirališno mjesto. Usprkos svim navedenim nedostacima, ipak postoje neke prednosti ovog projekta kao na primjer jednostavno obilježavanje parkirališnog mjesta na kojemu želimo pratiti promjenu stanja zauzetosti te jednostavna prilagodba bilo kojem parkiralištu. Upravo tim prednostima, kako je i navedeno, ostvaren je početno zadani cilj projekta.

LITERATURA

[1] „(PDF) An Image Feature-Based Method for Parking Lot Occupancy“, *ResearchGate*. https://www.researchgate.net/publication/334907158_An_Image_Feature-Based_Method_for_Parking_Lot_Occupancy (pristupljeno ruj. 08, 2020).

[2] „Kalman filter“, *Wikipedia*. kol. 25, 2020, Pristupljeno: ruj. 08, 2020. [Na internetu]. Dostupno na: https://en.wikipedia.org/w/index.php?title=Kalman_filter&oldid=974917947.

[3] M. Lopez, T. Griffin, K. Ellis, A. Enem, i C. Duhan, „Parking Lot Occupancy Tracking Through Image Processing“, str. 265–258, doi: 10.29007/69m7.

[4] BillWagner, „Introduction to the C# Language and the .NET Framework“. https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework (pristupljeno srp. 13, 2020).

[5] A. Gittleman, *Computing with C# and the .NET Framework*. Jones & Bartlett Publishers, 2011.

[6] „Overview- Windows Forms“. https://docs.microsoft.com/en-us/dotnet/framework/winforms/windows-forms-overview (pristupljeno kol. 18, 2020).

[7] What is Emgu CV? - Emgu CV Essentials. 2013.

[8] „Emgu.CV Namespace“. http://www.emgu.com/wiki/files/1.4.0.0/html/b72c032d-59ae-c36f-5e00-12f8d621dfb8.htm (pristupljeno kol. 20, 2020).

[9] „Emgu.CV.GPU Namespace“. http://www.emgu.com/wiki/files/2.4.0/document/html/0029e0f2-fc71-3cfc-2fa6-c5aa6f94f117.htm (pristupljeno kol. 20, 2020).

[10] „Emgu.CV.UI Namespace“. http://www.emgu.com/wiki/files/1.3.0.0/html/ec0d4663-4383-06f5-f0a4-deb44e42fafb.htm (pristupljeno kol. 20, 2020).

[11] „Emgu.CV.Util Namespace“. http://www.emgu.com/wiki/files/2.3.0/document/html/9146413d-68a0-bd0f-a0ae-d2af118e0c84.htm (pristupljeno kol. 20, 2020).

[12] „Microsoft Visual Studio“, *Wikipedia*. kol. 20, 2020, Pristupljeno: kol. 24, 2020. [Na internetu]. Dostupno na: https://en.wikipedia.org/w/index.php?title=Microsoft_Visual_Studio&oldid=974003111.

[13] „Optical Flow — OpenCV-Python Tutorials 1 documentation“. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_video/py_lucas_kanade/py_lucas_kanade.html (pristupljeno srp. 13, 2020).

[14] „OpenCV - The Gunnar-Farneback optical flow“, *GeeksforGeeks*, svi. 25, 2020. https://www.geeksforgeeks.org/opencv-the-gunnar-farneback-optical-flow/ (pristupljeno srp. 13, 2020).

[15] „(PDF) A Review On Particle Image Velocimetry And Optical Flow Methods In Riverine Environment.“, *ResearchGate*. https://www.researchgate.net/publication/320908264_A_Review_On_Particle_Image_Velocimetry_And_Optical_Flow_Methods_In_Riverine_Environment (pristupljeno ruj. 08, 2020).

SAŽETAK

Ovaj diplomski rad bavi se rješenjem problema nadzora slobodnih mjesta na parkiralištu korištenjem računalne obrade slike, odnosno problemom detekcije ulaska i izlaska automobila s određenog parkirališnog mjesta. Za rješavanje navedenog problema korištene su EmguCV tehnologija, C# programski jezik te Visual Studio razvojno okruženje koji su objašnjeni u samom radu. Kako bi se lakše dobili potrebni podaci, korišteni video isječak je binariziran, odnosno pretvoren u dvije boje, crnu i bijelu, a to je postignuto korištenjem Farneback Optical Flow metode koja gleda sve točke i detektira sve piksele na kojima se događa nekakva promjena između dva uzastopna okvira. Uspješno je postignuto rješenje detekcije ulaska i izlaska automobila s parkirališnog mjesta te prikaz stanja nadziranog mjesta u stvarnom vremenu.

Ključne riječi: C#, detekcija, EmguCv, Farneback, nadzor

ABSTRACT

C# parking monitoring application based on computer image processing

This thesis deals with solving the problem of monitoring free spaces in the parking lot using computer image processing, or the problem of detecting the entry and exit the certain parking space. To solve this problem, EmguCV technology, the C# programming language and Visual Studio development environment, was used and explained in the paper. To make it easier to get the data you need, used video clip is binary, that is converted to two colors, black and white, using Farneback Optical Flow method that look at all points and detect all pixels where some change occurs between two consecutive frames. The method is also explained in more detail in the paper itself. A solution for detecting the entry and exit of cars with parking spaces and showing the status of the monitored parking space in real time was successfully achieved.

Key words: C#, detection, EmguCV, Farneback, monitoring

ŽIVOTOPIS

Nikolina Češić rođena je 21.6.1995. godine u gradu Nova Gradiška. Osnovnu školu završila je 2010. godine u Osnovnoj školi Ante Starčevića u Rešetarima. Iste godine upisuje srednju školu u Gimnaziji Nova Gradiška koju završava 2014. godine. Nakon toga upisuje preddiplomski studij na Elektrotehničkom fakultetu u Osijeku (kasnije preimenovanom u Fakultet elektrotehnike, računarstva i informacijskih tehnologija), smjer računarstvo, koji završava 2017. godine. Iste godine upisuje diplomski studij na istom fakultetu, smjer informacijske i podatkovne znanosti koji i danas pohađa. Tijekom školovanja je sudjelovala nekoliko puta na školskom i županijskom natjecanju iz Informatike.

_____

Vlastoručni potpis