SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 3949

Lokalizacija parkirnih mjesta u nadzornom videu

Domagoj Vukadin

Zagreb, lipanj 2015.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Da bi ste uklonili ovu stranicu obrišite naredbu `\izvornik`.

SADRŽAJ

1. Uvod

Pregledati video, pregledati slike ili jednostavno pogledati okolinu oko sebe i donijeti određene zaključke na temelju viđenoga, trivijalno je za čovjeka. Čovjek jednostavno određuje oblik nekog predmeta, raščlanjuje različite objekte, prepoznaje emocije kod ljudi itd. Lako raspoznaje detalje budući da ima osjetilo vida koje mu to omogućuje.

Pojavom umjetne inteligencije, prirodno se javlja čovjekova potreba za stvaranjem sustava koji bi mogao dovoljno kvalitetno i samostalno razumjeti podatke iz slike. Sukladno tome, sredinom 20. stoljeća razvija se znanstvena disciplina računalni vid koja se bavi analiziranjem, razumijevanjem i obrađivanjem podataka dobivenih iz slike ili slijeda slika. Računalni vid ima široku primjenu u društvu, npr. dijagnostika u medicini, biometrija, navigacija autonomnih vozila, vojni sustavi i sl.

U ovom radu opisuje se moguće rješenje problema lokaliziranja parkirnih mjesta u nadzornom videu. Nadzorni video snima se kamerom iz ptičje perspektive. Zanimljivo je vidjeti može li se i kako otkriti uparkiravanje automobila budući da promatranjem iz ptičje perspektive nema objekata koji bi zaklanjali automobile. U rješenju problema koristi se metoda optičkog toka za praćenje točaka između dviju slika. Najprije se pronađu točke za praćenje koje predstavljaju automobil, a zatim ih se prati kroz slijed slika. Nakon što automobil zastane, pokušava se utvrditi položaj te obilježiti parkirno mjesto ako je zaustavljen duže od određenog vremena. Postoje već rješenja koja nude sličan pristup lokaliziranju parkirnih mjesta, neka od njih postignuta su algoritmom klasificiranja histograma boja u [4] i modeliranjem pozadine u [14].

U drugom poglavlju opisani su postojeći sustavi koji se trenutno koriste u nadziranju parkirnih mjesta. U trećem poglavlju definira se optički tok koji se koristi u radu kao osnovna metoda za praćenje točaka u slijedu slika. U četvrtom i petom poglavlju objašnjeni su dodatno potrebni algoritmi za rad sustava kao što su analiza svojstvenih komponenata (engl. *PCA*) i *RANSAC* (engl. *Random sample consensus*). U petom poglavlju pregled je programske podrške korištene za razvijanje sustava. U šestom i sedmom poglavlju prikazana je implementacija te eksperimenti i rezultati rada.

2. Postojeća rješenja za detektiranje parkirnih mjesta

Sustavi za nadzor parkinga zadnjih su 10-tak godina posebno zanimljivi zbog velikog broja praktičnih primjena. Pronalazak slobodnih parkirnih mjesta u velikim gradovima često je naporno. Neizbježno je uložiti određeno vrijeme, potrošiti gorivo i zagaditi okoliš ispušnim plinovima dok se kruži i traži slobodno mjesto. Kako bi se riješili takvi problemi ili barem umanjili, industrija predlaže rješenja temeljena na različitim sustavima koji se mogu podijeliti na:

- induktivne petlje
- ultrazvučni senzor zauzetosti
- nadzorne kamere.

2.1. Induktivne petlje

Jedno rješenje za brojanje slobodnih i zauzetih parkirnih mjesta nudi upravo sustav temeljen na induktivnoj petlji. Broje se vozila koja ulaze u prostor i izlaze iz prostora parkinga. Za potrebe sustava koristi se detektor induktivne petlje [1] koji se postavlja točno na ulaze i izlaze parkinga. Instalacija induktivne petlje relativno je jeftina pa se često koristi. Takav sustav može točno izvještavati o broju slobodnih i broju zauzetih parkirnih mjesta, ali npr. ne može usmjeravati vozače prema slobodnim parkirnim mjestima, što bi bilo poželjno na velikim parkiralištima.

2.2. Ultrazvučni senzor zauzetosti

Rješenje koje nudi uz brojanje slobodnih i zauzetih parkirnih mjesta i usmjeravanje vozača prema slobodnim mjestima, jest ultrazvučni senzor zauzetosti. Primjenjuje se na način da se na svako parkirno mjesto ugrađuje monitor i ultrazvučni senzor koji

prati zauzetost parkirnog mjesta. Na monitoru se prikazuju potrebne informacije o parkingu - zauzetost, smjer prema parkirnom mjestu i vodič za vozače do tog parkirnog mjesta. Monitor se postavlja iznad parkirnog mjesta kako bi ga se moglo uočiti s veće udaljenosti. Mana ovakvog sustava je jako velik trošak izgradnje zbog toga što se na svako parkirno mjesto treba ugraditi i senzor i monitor.

2.3. Nadzorne kamere

Zadnje navedeno rješenje temelji se na kamerama. Smatra se izvrsnim načinom za praćenje slobodnih mjesta velikih vanjskih parkinga jer nije potrebno instalirati veliki broj senzora niti bilo kakve druge uređaje. Potrebno je poznavanje računalnog vida i naprednije matematike za prikupljanje podataka iz kamere. Pokazalo se da se sustav može implementirati na postojeće nadzorne kamere koje su već povezane na središnji sustav praćenja parkinga. U [3] je opisano jedno rješenje koje uključuje nadzorne kamere. Rješenje se temelji na tome da se ručno zadaju parkirna mjesta i zatim se detektira je li parkirno mjesto slobodno ili zauzeto. Takav pristup je bitno drugačiji od ponuđenog pristupa u ovom radu gdje se prati automobil i zatim zaključuje je li se automobil u nekom trenutku parkirao.

3. Optički tok

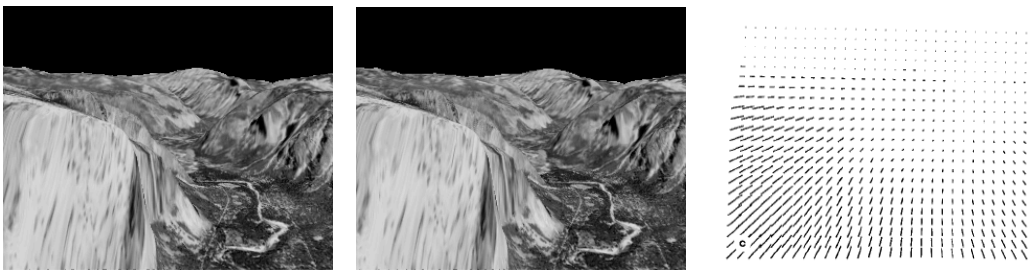
3.1. Definicija

Praćenje objekata u slijedu slika jedno je od najvažnijih područja u računalnom vidu. Praćenjem se želi procijeniti putanja nekog objekta. Moguće je pratiti različitim pristupima na razne načine koji su objašnjeni u [13]:

- praćenje jednog objekta
- praćenje više objekata
- praćenje statičnom kamerom
- praćenje kamerom u pokretu
- praćenje temeljeno na više kamera.

Praćenje optičkim tokom primjenjuje se unazad 35 godina. Optički tok doslovno se odnosi na pomake intenziteta uzoraka, odnosno koristi se za praćenje uzoraka kroz slijed slika. Podaci koji se žele dobiti zapravo su projekcije na sliku iz 3D prostora. Takvi podaci zovu se **polje gibanja**.

Svakom pikselu u trenutnoj slici pokušava se pridijeliti dvokomponentni vektor brzine $\vec{v}$ koji pokazuje položaj piksela u odnosu na referentnu sliku. Na slici 3.1 prikazan je optički tok između dvije slike.



(a) Vrijeme t **(b)** Vrijeme t + dt **(c)** Vektori pomaka

Slika 3.1: Optički tok

3.2. Metoda Lucas-Kanade

3.2.1. Osnovna metoda

U ovom radu koristi se metoda optičkog toka Lucas-Kanade. Metoda je izvedena na temelju pretpostavke da jednadžba ograničenja optičkog toka[10] vrijedi u okolini nekog razmatranog elementa e te da su pomaci u okolini tog elementa mali i uglavnom jednaki. Postupak pripada kategoriji lokalnih postupaka. Neka P označava skup svih točaka okoline elementa e. U literaturi se okolina često naziva i prozor (engl. *window*)i najčešće je kvadratnog oblika s razmatranim elementom u središtu. Definiraju se jednadžbe koje vektor lokalnog toka $\mathbf{v} \equiv (v_x, v_y)^T$ mora zadovoljiti:

$$\nabla I(q_i, t) \cdot \mathbf{v} + I_t(q_i, t) = 0, \qquad \forall q_i \in P, \tag{3.1}$$

odnosno po komponentama:

$$I_x(q_i)V_x + I_y(q_i)V_y + I_t(q_i) = 0, \qquad \forall q_i \in P, \tag{3.2}$$

gdje $\nabla I(q_i) \equiv (I_x(q_i), I_y(q_i))$ i $I_t(q_i)$ označavaju prostornu i vremensku parcijalnu derivaciju slike *I* u točki q_i u trenutku *t* koja je opisana u [8].

Jednadžbe se mogu prikazati u matričnom obliku $\mathbf{Av} = \mathbf{b}$

$$\mathbf{A} = \begin{bmatrix} I_x(q_1) & I_y(q_1) \\ I_x(q_2) & I_y(q_2) \\ \vdots & \vdots \\ I_x(q_n) & I_y(q_n) \end{bmatrix}, \quad \mathbf{v} = \begin{bmatrix} v_x \\ v_y \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} -I_t(q_1) \\ -I_t(q_2) \\ \vdots \\ -I_t(q_n) \end{bmatrix}$$

gdje je n broj elemenata okoline *P*, n = *card*(*P*). U sustavu ima više jednadžbi od nepoznanica pa je sustav pre-definiran. Metoda Lucas-Kanade procjenjuje rješenje metodom najmanjih kvadrata koja je predstavljena u [12]

$$min \quad E(v) = \|\mathbf{Av} - \mathbf{b}\|^2. \tag{3.3}$$

Problem minimizacije ima jedinstveno rješenje, što je pokazano u [2]:

$$\mathbf{A}^T\mathbf{Av} = \mathbf{A}^T\mathbf{b} \tag{3.4}$$

$$\mathbf{v} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b} \tag{3.5}$$

što se matrično može zapisati kao

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} \sum_i I_x(q_i)^2 & \sum_i I_x(q_i)I_y(q_i) \\ \sum_i I_y(q_i)I_x(q_i) & \sum_i I_y(q_i)^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i I_x(q_i)I_t(q_i) \\ -\sum_i I_y(q_i)I_t(q_i) \end{bmatrix}$$

pri čemu sume iteriraju po svim elementima okoline razmatranog elementa e. Postupak najmanjih kvadrata daje jednaku važnost svim elementima q_i okoline P razmatranog elementa e. U praksi je bolje dati veću težinu elementima koji su bliže razmatranom elementu e pa je uobičajeno koristiti težinsku inačicu najmanjih kvadrata:

$$\mathbf{A}^T\mathbf{W}\mathbf{A}\mathbf{v} = \mathbf{A}^T\mathbf{W}\mathbf{b} \tag{3.6}$$

$$\mathbf{v} = (\mathbf{A}^T\mathbf{W}\mathbf{A})^{-1}\mathbf{A}^T\mathbf{W}\mathbf{b} \tag{3.7}$$

gdje je **W** dijagonalna n x n matrica koja sadrži težine koje se pridružuju jednadžbi elementa q_i. Matrica **W** obično se definira kao Gaussova funkcija udaljenosti između elementa q_i i razmatranog elementa e. Matrični zapis definiran je na sljedeći nacin:

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} \sum_i \omega_i I_x(q_i)^2 & \sum_i \omega_i I_x(q_i)I_y(q_i) \\ \sum_i \omega_i I_x(q_i)I_y(q_i) & \sum_i \omega_i I_y(q_i)^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i \omega_i I_x(q_i)I_t(q_i) \\ -\sum_i \omega_i I_y(q_i)I_t(q_i) \end{bmatrix}$$

Težina ω_i odnosi se na težinu između elementa q_i i razmatranog središnjeg elementa e.

3.2.2. Piramidalna inačica

Osnovni algoritam optičkog toka pronalazi dovoljno dobro rješenje ako su pomaci mali u okolini razmatranog elementa e. Međutim, zanima nas što se događa s optičkim tokom ako pomaci nisu mali. Za takve pomake opisana je piramidalna inačica algoritma u [9] na sljedeći način:

1. Izračunaj optički tok na najvišoj razini piramide
2. na razini i:
   (a) interpoliraj optički tok $\mathbf{v}_{i-1}$ razine i - 1 kako bi se dobio tok $\mathbf{v}_i^*$ dvostruke rezolucije
   (b) skaliraj vektore optičkog toka s 2, $\mathbf{v}_i^* := 2\mathbf{v}_i^*$
   (c) I_t izračunaj iz bloka pomaknutog za $\mathbf{v}_i$

(d) koristeći I_t izračunaj optički tok kako bi se dobile korekcije $\mathbf{v}'_i$

(e) primijeni korekcije $\mathbf{v}_i = \mathbf{v}^*_i + \mathbf{v}'_i$

Piramida predstavlja skup slika pri čemu je svaka duplo manja od prethodne. Izvorna slika može se označiti s I^0. To je slika najveće rezolucije. Piramidalna reprezentacija gradi se rekurzivnim načinom: I^1 izračuna se iz I^0 smanjivanjem rezolucije dva puta, a zatim na isti način I^2 iz I^1 itd. Detalji su objašnjeni u [11].

4. Analiza svojstvenih komponenata

Analiza svojstvenih komponenata (engl. *Principal components Analysis*, kraće *PCA*) statistička je metoda koja se koristi u području prepoznavanja lica i kompresije slika, ali je također pouzdana i u pretvaranju podataka iz n dimenzija u manji broj dimenzija. Za razumijevanje *PCA* metode potrebno je shvatiti kako funkcioniraju osnovni matematički koncepti koji se koriste u metodi - statistika i linearna algebra.

4.1. Statistički pokazatelji skupa podataka

Princip statistike je analiziranje skupine velikog broja podataka na način da se pokuša razumjeti veza između svakog podatka u skupini. U nastavku se opisuje nekoliko mjerenja koja se primjenjuju na takav skup podataka.

4.1.1. Aritmetička sredina

Aritmetička sredina jedna je od najvažnijih vrijednosti u statistici. Formula za izračunavanje aritmetičke sredine u X skupu s n podataka je:

$$\bar{X} = \sum_{i=1}^{n} \frac{X_i}{n} \tag{4.1}$$

Formula daje srednju vrijednost skupa podataka, međutim iz te vrijednosti nije moguće shvatiti kako su podaci raspršeni u skupu. Aritmetička sredina za dva skupa podataka:

$$[0, 8, 12, 20] \text{ i } [8, 9, 11, 12]$$

je ista, a očito je da su podaci drugačije raspršeni.

4.1.2. Standardna devijacija

Standardna devijacija koristi se za izračunavanje prosječne udaljenosti svih podataka od aritmetičke sredine, odnosno govori o tom koliko su podaci raspršeni u skupu. Računa se na način da se uzme uzorak nekog skupa podataka te se zbroje kvadrati udaljenosti svakog podatka od aritmetičke sredine, zatim se podijeli s $n-1$ te se korjenuje dobiveni rezultat. Oznaka za standardnu devijaciju je s, a formula:

$$s = \sqrt{\frac{\sum_{i=1}^{n}(X_i - \bar{X})^2}{(n-1)}} \tag{4.2}$$

Dijeli se s $n-1$, a ne s n, jer se time dobiva bliži rezultat onome kada bi se izračunavalo s cijelim skupom podataka. Primjer računanja standardne devijacije za skup $[0, 8, 12, 20]$ prikazan je u tablici:

X	$(X - \bar{X})$	$(X - \bar{X})^2$
0	-10	100
8	-2	4
12	2	4
20	10	100
Ukupno		208
Podijeljeno s $(n-1)$		69.333
Drugi korijen		8.3266

Tablica 4.1: Izračunavanje standardne devijacije

4.1.3. Varijanca

Varijanca je drugi način za mjerenje raspršenosti podataka u skupu podataka. Formula je vrlo slična formuli za izračunavanje standardne devijacije (4.2). Formula je:

$$s^2 = \frac{\sum_{i=1}^{n}(X_i - \bar{X})^2}{(n-1)} \tag{4.3}$$

Dakle, varijanca je također mjera za raspršenost, ali je prikladnija za izračunavanje kovarijance koja će se objasniti u nastavku.

4.1.4. Kovarijanca

Standardna devijacija i varijanca primjenjuju se na 1-dimenzionalne podatke. Ali što je s podacima koji su višedimenzionalni kao što su npr. ocjene i broj sati učenja studenata? Korisno je utvrditi vezu između ocjena i sati učenja. Standardna devijacija i

varijanca ne mogu pomoći pri rješavanju takvog problema. Za utvrđivanje tog odnosa služi kovarijanca koja se uvijek računa između dvije dimenzije. Ako se računa između jedne dimenzije, tada je jednaka varijanci. Formula za izračunavanje varijance može se preoblikovati da izgleda ovako:

$$var(X) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(X_i - \bar{X})}{(n-1)} \tag{4.4}$$

gdje se kvadrat rastavio na dva faktora. Tako zapisanom formulom lako se dolazi do formule za kovarijancu u kojoj se u brojniku množe dvije dimenzije:

$$cov(X, Y) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{(n-1)} \tag{4.5}$$

Dobiveni rezultat ne promatra se kao vrijednost, nego je li on pozitivan ili negativan. Ako je pozitivan, znači da obe dimenzije rastu proporcionalno, tj. da u ovom slučaju raste ocjena ako raste broj sati učenja. Ako je negativan, tada su dimenzije obrnuto proporcionalne, tj. ocjena se smanjuje ako raste broj sati učenja. Rezultat može biti i 0, to znači da dimenzije ne ovise jedna o drugoj, odnosno da su neovisne.

4.1.5. Kovarijacijska matrica

Ako imamo višedimenzionalne podatke, npr. (dimenzije x, y, z) računa se više kovarijanci - $cov(x, y), cov(x, z), cov(y, z)$. Rezultate je prikladno smjestiti u matricu radi lakšeg prikaza. Kovarijacijska matrica definira se kao:

$$C^{n\mathbf{x}n} = (c_{i,j}, c_{i,j} = cov(Dim_i, Dim_j)),$$

gdje je $C^{n\mathbf{x}n}$ matrica s n redaka i n stupaca, a Dim_n je n-ta dimenzija. Primjer kovarijacijske matrice za 3 dimenzije (x, y, z) izgleda ovako:

$$C = \begin{bmatrix} cov(x,x) & cov(x,y) & cov(x,z) \\ cov(y,x) & cov(y,y) & cov(y,z) \\ cov(z,x) & cov(z,y) & cov(z,z) \end{bmatrix}$$

Matrica je simetrična oko glavne dijagonale jer je $cov(x, y) = cov(y, x)$. Po glavnoj dijagonali su kovarijance istih dimenzija, što je jednako varijanci te dimenzije $cov(x, x) = var(x)$.

Primjer skupa podataka s izračunatim kovarijancama dan je u tablicama:

	$Sati(S)$	$Ocjena(O)$
Podaci	9	39
	15	56
	25	93
	14	61
	10	50
	18	75
	0	32
	16	85
	5	42
Ukupno	112	533
Prosjek	12.44	59.22

Tablica 4.2: Skup podataka

S	O	$(S_i - \bar{S})$	$(O_i - \bar{O})$	$(S_i - \bar{S})(O_i - \bar{O})$
9	39	-4.92	-23.42	115.23
15	56	1.08	-6.42	-6.93
25	93	11.08	30.58	338.83
14	61	0.08	-1.42	-0.11
10	50	-3.92	-12.42	48.69
18	75	4.08	12.58	51.33
0	32	-13.92	-30.42	423.45
16	85	2.08	22.58	46.97
5	42	-8.92	-20.42	182.15
Kovarijanca				149.95

Tablica 4.3: Izračunavanje kovarijance

4.2. Linearna algebra

Nakon što su utvrđene statističke varijable koje su potrebne za provedbu algoritma *PCA*, potrebno je objasniti matrično računanje koje je u pozadini algoritma. Potrebno je izračunati svojstvene vektore iz svojstvenih vrijednosti što je definirano u [5].

Vektor $\mathbf{v} \neq 0$ zovemo svojstvenim (vlastitim) vektorom matrice $\mathbf{A}$ ako postoji skalar λ takav da vrijedi:

$$\mathbf{Av} = \lambda\mathbf{v} \tag{4.6}$$

Ako je v svojstveni vektor, onda je i αv svojstveni vektor, za svaki $\alpha \neq 0$.

Nalaženje svojstvenih vektora svodi se na rješavanje homogenog linearnog sustava:

$$(\lambda\mathbf{I} - \mathbf{A})\mathbf{v} = 0 \tag{4.7}$$

Dakle, $\mathbf{v}$ je svojstveni vektor ako i samo ako ovaj homogeni sustav ima rješenje razli-

čito od 0. Da bi jednadžba (4.7) imala netrivijalno rješenje, matrica $\lambda\mathbf{I} - \mathbf{A}$ ne smije biti regularna. Zato njezina determinanta mora biti jednaka:

$$det(\lambda\mathbf{I} - \mathbf{A}) = \begin{vmatrix} \lambda - a_{11} & -a_{12} & \cdots & -a_{1n} \\ -a_{21} & \lambda - a_{22} & \cdots & -a_{2n} \\ \vdots & & \ddots & \\ -a_{n1} & -a_{n2} & \cdots & \lambda - a_{nn} \end{vmatrix} = 0 \tag{4.8}$$

Ova determinanta je polinom po nepoznanici λ, stupnja n. Nazivamo ga **karakteristični polinom** matrice **A** i označavamo ga s $\kappa(\lambda)$, $\kappa(\lambda) := det(\lambda\mathbf{I} - \mathbf{A})$.
Jednadžba $\kappa(\lambda) = det(\lambda\mathbf{I} - \mathbf{A}) = 0$ naziva se **karakteristična jednadžba** matrice **A**. Njezina rješenja su svojstvene vrijednosti matrice **A**.

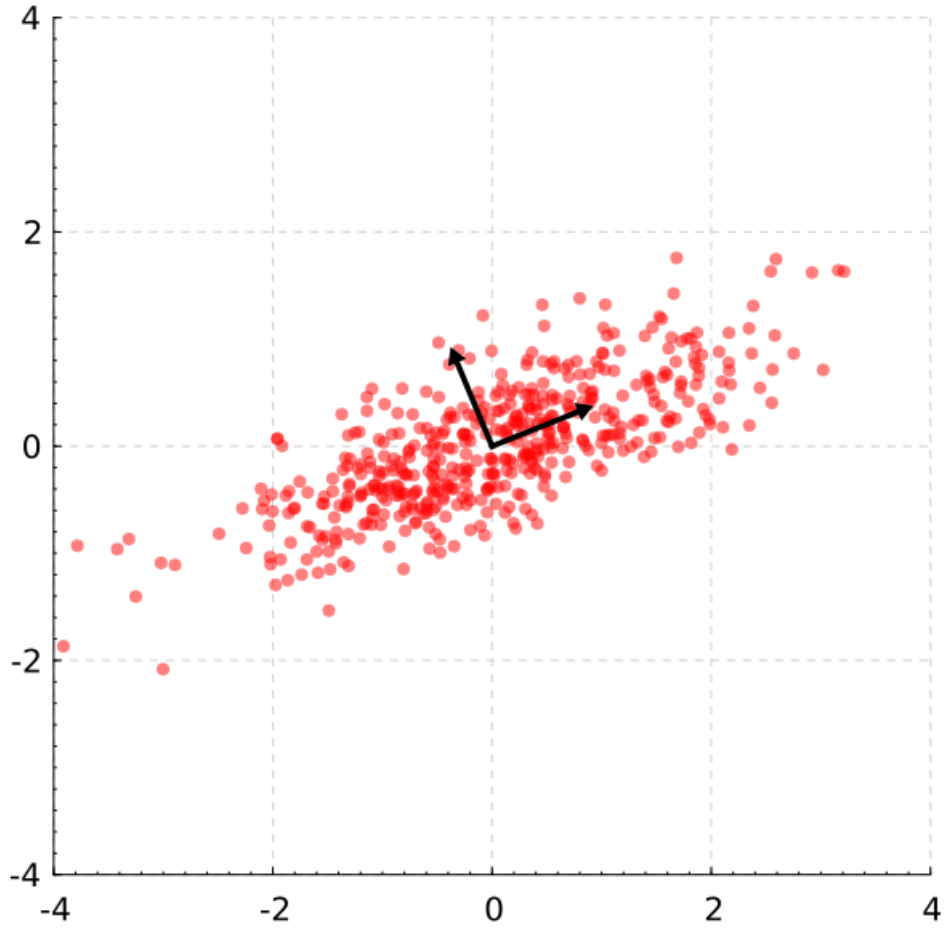
4.3. Algoritam traženja svojstvenih vektora

Neka je X skup podataka u dvije dimenzije, a X_i podatak u skupu definiran s x i y koordinatama.

1. od svakog X_i oduzmemo aritmetičku sredinu pa time dobivamo novu aritmetičku sredinu $\bar{X}_i$ = 0
2. izračuna se kovarijacijska matrica koja je zbog dvije dimenzije 2x2
3. iz kovarijacijske matrice izračunaju se svojstvene vrijednosti
4. za svaku svojstvenu vrijednost izračuna se jedinični svojstveni vektor; u ovom slučaju postojat će dvije svojstvene vrijednosti, stoga je potrebno izračunati dva jedinična svojstvena vektora

Svojstveni vektori su okomiti jedan na drugog, odnosno vrijedi:
$\vec{v_1} \cdot \vec{v_2} = 0$

Svojstveni vektor veće svojstvene vrijednosti naziva se **svojstvena komponenta** skupa podataka i pokazuje u kojem smjeru su raspršeni podaci. Na slici 4.1 prikazani su svojstveni vektori na skupu točaka u 2D prostoru.



Slika 4.1: PCA

5. RANSAC

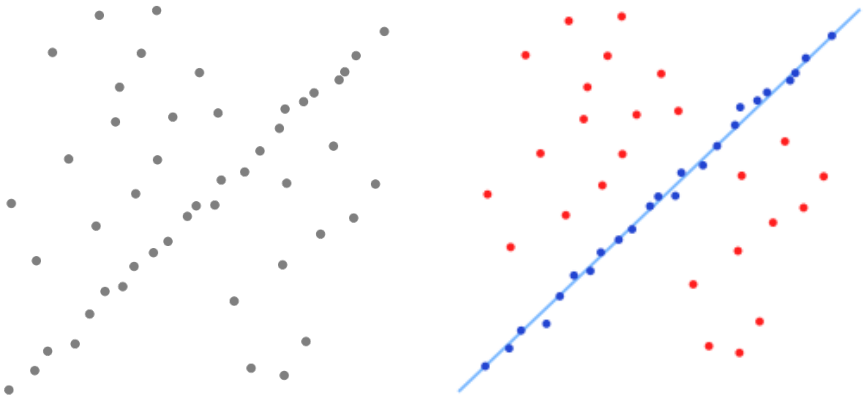
RANSAC (engl. *Random Sample Consensus*) predstavljen u [7] je algoritam za robusnu procjenu modela u skupu podataka. Temelji se na tom da se skup podataka sastoji od podataka koji su u modelu prema skupu parametara (engl. *inliears*) te podataka koji ne pripadaju modelu - vanpopulacijske značajke (engl. *outliers*). *RANSAC* je tehnika slučajnog ponovnog uzorkovanja koja odabire najmanji broj značajki potrebnih za procjenu modela i računa koliko se značajki može uklopiti u izračunati model.

5.1. Algoritam

1. slučajno odaberi minimalni broj podataka potrebnih za određivanje parametara modela
2. izračunaj parametre (broj iteracija, prag tolerancije, broj podataka)
3. odredi koliko podataka od cijelog skupa podataka pripada modelu e
4. ako postotak broja podataka značajki unutar modela u odnosu na ukupan broj podataka premašuje definirani prag τ, utvrdi da je to najbolji model koristeći pronađene unutarnje značajke i izađi iz algoritma
5. ako nije utvrđeno, ponovi korake 1-4 (najviše N puta)

N je najveći broj iteracija koji se odabire tako da se uvijek osigura da barem jedan slučajni uzorak ne sadrži vanpopulacijske značajke.

Na slici 5.1 prikazan je primjer algoritma *RANSAC* koji radi na modelu pravac. Plave točke su *inliears*, a crvene točke *outliers*.



(a) Skup podataka **(b)** Primjena algoritma

Slika 5.1: RANSAC

6. Programska podrška

Programske komponente razvijene su u programskom jeziku C++ na 64-bitnom operativnom sustavu Windows 7 u okruženju Microsoft Visual Studio 2013. Za rad s videozapisima korištena je vanjska biblioteka OpenCV 2.4.10. U nastavku su ukratko objašnjeni potrebni alati za implementaciju.

C++ je objektno-orijentirani programski jezik opće namjene. Razvio se 80-tih godina prošlog stoljeća, a temeljen je na programskom jeziku C. Nudi mogućnost programiranja na visokoj razini (korištenje objekata), ali i mogućnost programiranja na najnižoj razini poput C-a s osnovnim podatkovnim strukturama. Najčešće se koristi za izradu sustavskih i upravljačkih programa, klijent-poslužitelj aplikacija i memorijskih čipova. Također je predviđen je za razvijanje velikih robusnih i skalabilnih aplikacija jer pruža izrazito visoke performanse u odnosu na druge programske jezike.

OpenCV (engl. *Open Source Computer Vision*) je vanjska biblioteka otvorenog koda koja se koristi prvenstveno u računalnom vidu. Sadrži više od 2500 algoritama za računalni vid. Visoko je optimizirana što se tiče potrošnje memorije i brzine izvođenja, pisana je u programskim jezicima C i C++ i predviđena je za korištenje u aplikacijama koje rade u realnom vremenu. Pristup biblioteci omogućen je preko sučelja za programske jezike C, C++, Python i Javu. OpenCV je *cross-platform* biblioteka jer podržava gotovo sve najčešće operativne sustave kao što su Windows, Linux, MAC OS, iOS i Android. Najnovija verzija je 3.0 beta, a u ovom radu korištena je verzija 2.4.10. Cijela biblioteka podijeljena je na cjeline (module) radi lakšeg korištenja. Detaljniji opis biblioteke *OpenCV* nalazi se u [6], a za potrebe ovog rada korišteni su sljedeći moduli:

core
: definira osnovne strukture podataka za rad sa slikama i videozapisima i osnovne funkcije koje se koriste u svim ostalim modulima

features2d

sadrži osnovne detektore značajki i algoritme za usporedbu deskriptora

calib3d

sadrži algoritme za kalibraciju mono i stereo kamere, procjenu položaja objekta, 3D rekonstrukciju i algoritme za pronalaženje korespondencije između značajki

video

sadrži algoritme za analizu videozapisa što uključuje procjenu gibanja (optički tok), oduzimanje pozadine i praćenje objekata

imgproc

sadrži mnoge algoritme vezane za analizu slike, uključuje linearno i nelinearno filtriranje, geometrijske transformacije, pretvaranje boja i histograme

highgui

nudi sučelje za učitavanje videzapisa, čitanje različtiih slikovnih i video formata te funkcije za stvaranje grafičkog sučelja

7. Implementacija

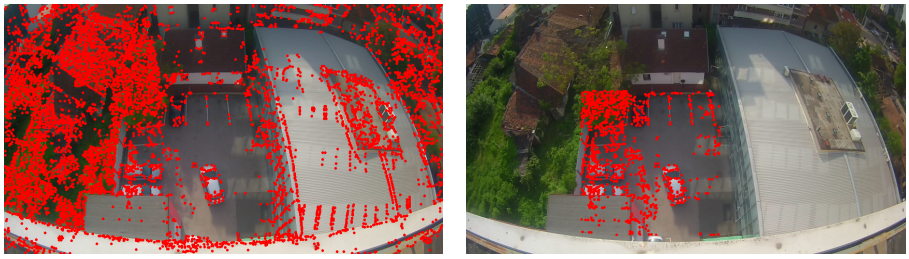
7.1. Traženje značajki

Program na ulazu prima sliku po sliku iz videozapisa parkinga te se na svakoj slici pretražuju potencijalne značajke koje bi trebale predstavljati automobil. Za izabiranje potencijalnih značajki koristi se gotova metoda iz biblioteke *OpenCV*:

Prototip metode 7.1: Pronalaženje potencijalno dobrih značajki

```
void goodFeaturesToTrack (InputArray image, OutputArray
    corners, int maxCorners, double qualityLevel, double
    minDistance, InputArray mask=noArray(), int lockSize=3,
    bool useHarrisDetector=false, double k=0.04)
```

Metoda daje rezultat tako da zapisuje koordinate potencijalnih značajki u polje *Output array Corners* koje se predaje kao argument funkcije. Kako nama trebaju značajke isključivo s parkinga, poželjno je u programu odrediti područje interesa (parking), dakle izuzeti okolne objekte kao što su drveća, zgrade, kuće i slično. Time se jako puno smanji broj operacija koje rade funkcije jer se više ne računa na cijeloj slici koja je veličine 848x480p nego na dijelu slike, odnosno otprilike 300x250p ako se gleda ispitni skup za ovaj rad. Smanjenjem broja operacija vrijeme izvođenja programa uvelike se smanjuje što je od velike važnosti za aplikaciju u realnom vremenu.



(a) Cijela slika **(b)** Područje interesa

Slika 7.1: Potencijalno dobre značajke

Na slici 7.1 prikazan je učinak metode *goodFeaturesToTrack* na cijeloj slici i na području interesa.

7.2. Optički tok

Nakon što se odrede značajke, primjenjuje se algoritam optičkog toka kako bi se pronašle značajke na sljedećoj slici koje su u korespondenciji sa značajkama iz prethodne slike, odnosno želi se izračunati novi položaj značajki iz prethodne slike. Izračunavanje optičkog toka provodi se gotovom metodom iz *OpenCV-a*:

Prototip metode 7.2: Izračunavanje optičkog toka algoritmom Lucas-Kanade

```
void calcOpticalFlowPyrLK(InputArray prevImg, InputArray
   nextImg, InputArray prevPts, InputOutputArray nextPts,
   OutputArray status, OutputArray err, Size
   winSize=Size(21,21), int maxLevel=3, TermCriteria
   criteria=TermCriteria(TermCriteria::COUNT+TermCriteria::EPS,
   30, 0.01), int flags=0, double minEigThreshold=1e-4)
```

Koordinate korespondentnih značajki zapisuju se u *InputOutputArray nextPts*. Zbog utjecaja vjetra i odnosa svjetla i sjene na videozapisu, moguće je da će algoritam optičkog toka pronaći značajke koje su udaljene od prvotnog položaja samo 1 piksel (engl. *pixel*, kraće px) ili manje. Kako se automobili ipak brže kreću od 1px po slici, potrebno je uvesti prag koji će takve sitne pomake odbacivati. Također je problem i krošnja drveta na parkingu koja se njiše pod utjecajem vjetra tako da se lako može krivo detektirati.

7.3. RANSAC

Budući da je parking mjesto gdje se i ljudi kreću osim automobila, potrebno je osmisliti kako razlikovati automobil od čovjeka, budući da se trebaju pratiti samo automobili. Za pretpostaviti je da će više značajki predstavljati automobil nego čovjeka, jer gledano iz ptičje perspektive, automobil zauzima veću površinu od čovjeka, time i veći broj piksela. Za određivanje veće skupine značajke, razvijena je metoda koja je zapravo modificirani *RANSAC*:

Programski odsječak 7.3: metoda za izračunavanje skupina značajki

```
void Tracking::neighborsCount(){

int neighbors;
int neighborsMax = 0;
float distance;
string key;
vector<Point2f> newPoint;

for (int j = 0; j < realPoints[1].size(); j++){
  Point2f p = realPoints[1][j];

  neighbors = 0;
  for (int k = 0; k < realPoints[1].size(); k++){
    Point2f q = realPoints[1][k];
    if (q.x == p.x && q.y == p.y)
      continue;

    distance = sqrt( square(p.y - q.y) + square(p.x - q.x));
    if (distance < THRESHOLD_NEIGHBORS){
      neighbors++;
      key = to_string(p.x) +","+ to_string(p.y);
      neighborsMap[key].push_back(q);

    }
  }

  neighborsGroups.insert(make_pair(neighbors, p));
}
//empty realPoints
realPoints[0].clear();
realPoints[1].clear();
}
```

U prvoj petlji prolazi se po svakoj koordinati p koja je pronađena optičkim tokom, a u drugoj petlji pronalaze se susjedi q, tj. točke koje su bliže od praga *THRESHOLD_NEIGHBOURS*. Ako se pronađe točka q koja ispunjava uvjet, sprema se u mapu *NeighboursMap* kao vrijednost pod ključem p. Varijabla *neighbours* pamti broj susjeda svake točke. Na kraju se u mapu *neighboursGroups* spremi par (*neighbours, p*). Nakon što su poznate koordinate koje imaju najveće skupine susjeda, prate se upravo takve koordinate, jer bi one trebale predstavljati automobil. Ipak, takva metoda nije 100% točna jer je nekad nemoguće razlikovati model automobila od čovjeka ako još postoji neka sjena pored njega. Na slici 7.2 prikazan je dobar i loš primjer grupiranja značajki.

(a) Dobro grupiranje

(b) Loše grupiranje

Slika 7.2: Grupiranje značajki metodom *RANSAC*

7.4. PCA

Nakon pronalaženja značajki koje predstavljaju automobil, potrebno je evaluirati točnost. Za potrebe evaluiranja na svakoj petoj slici ručno se označio rotirani pravokutnik oko automobila koji se kreće te su se vrhovi pravokutnika zapisali u tekstualnu datoteku. Kako bi se provjerila točnost, potrebno je na svakoj slici programski aproksimirati značajke rotiranim pravokutnikom kako bi se moglo usporediti s vrhovima koji su se ručno označili. Najjednostavnije je odrediti presjek programski i ručno iscrtanog rotiranog pravokutnika. Za aproksimaciju značajki rotiranim pravokutnikom, koristi se analiza svojstvenih komponenata (engl. *PCA*). U nastavku je priložena metoda koja računa svojstvene vektore potrebne za iscrtavanje rotiranog pravokutnika.

Programski odsječak 7.4: metoda za izračunavanje svojstvenih vektora

```
void Tracking::myPCA(vector<Point2f> &convexPoints){

    vector<Point2f>dataAdj;
    Point2f qMean;
    float X[2], Y[2]; //eigenVectors

    dataAdjust(qMean, dataAdj, convexPoints);
    oldQmean = qMean;
    oldPoint[oldIndex + 1 % 3] = qMean;

    float cov[2][2] = {0.0f};
    getCovariance(dataAdj, cov[0]);
    getEigenVectors(cov[0], X, Y);

    setToBase(X); //normalize vectors
    setToBase(Y);

    //clear covariance matrix
    clearCovariance(cov[0]);
    drawPCA(qMean, X, Y);
}
```

Metoda *DataAdjust* zapisuje u vektor *DataAdj* normalizirane točke iz vektora *convexPoints*, odnosno od svake točke oduzima aritmetičku sredinu po x i y koordinatama. Vektor *oldPoint* sprema zadnjih nekoliko aritmetičkih sredina kako bi se kasnije mogla odrediti trajektorija automobila za uparkiravanje. Nakon što se normaliziraju točke, metoda *getCovariance* stvara kovarijacijsku matricu. Zatim se metodom *getEigenVectors* pronalaze svojstveni vektori iz svojstvenih vrijednosti kovarijacijske matrice. Svojstveni vektori se poslije toga postavljaju na jedinične vektore metodom *setToBase*. Na slici 7.3 prikazan je rezultat *PCA* algoritma. Svojstveni vektori označeni su plavom bojom, i na temelju njih konstruiran je rotirani pravokutnik.

Slika 7.3: Svojstveni vektori

8. Eksperimenti i rezultati

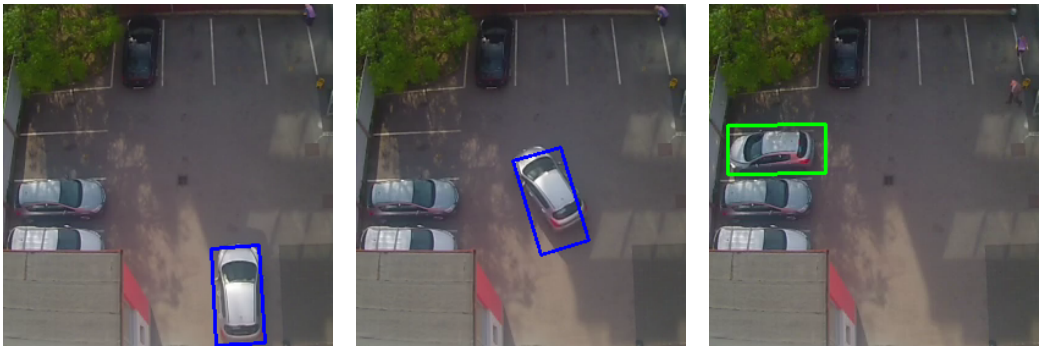
Za ispitni skup korišteno je više videozapisa parkinga iz ptičje perspektive snimljenih po danu uz sunčano vrijeme bez padalina. Videozapisi su snimljeni statičnom kamerom Overmax ActiveCam Sky i to na rezoluciji 848x480p sa 60 sličica po sekundi.

Slika 8.1: Overmax ActiveCam Sky

Za potrebe validiranja detektiranja automobila, potrebno je ručno označiti automobil kako bi se moglo precizno usporediti s programskom detekcijom. Za ručno označavanje koristio se skup od nekoliko stotina slika videozapisa. Razvijena je skripta koja računa postotak detekcije. Postotak detekcije vrši se na način da se izračuna presjek aproksimiranog rotiranog pravokutnika i rotiranog pravokutnika koji je ručno označen nad automobilima. Aproksimirani rotirani pravokutnik je rezultat primjene *PCA* algoritma nad pronađenim značajkama, odnosno iz skupa značajki pronađu se svojstveni vektori te se na osnovu njih iscrta takav pravokutnik. Na slici 8.2 zelenom bojom prikazan je presjek aproksimacije i ručno nacrtanog pravokutnika. Crvene točke predstavljaju pronađene značajke, plavi pravokutnik predstavlja aproksimirani, a crni ručno nacrtani pravokutnik.

Slika 8.2: Detekcija automobila

Za prvi eksperiment, odabrano je detektiranje uparkiravanja automobila svijetle boje. Kako je velika razlika u boji automobila i asfalta, detektiranje je u velikoj mjeri uspješno kao što se i očekivalo. U trenutku kad se automobil nalazi u sjeni, također nema problema s detekcijom. Jedino je teže bilo detektirati na dijelu kad se automobil kretao većom brzinom, tj. pri dolasku na mjesto skretanja za parkirno mjesto.



(a) 136. slika **(b)** 361. slika **(c)** 541. slika

Slika 8.3: Detektiranje automobila

U tablici 8.1 prikazan je postotak detekcije automobila na slikama (engl. *frame*) za vrijeme uparkiravanja. Prvi stupac tablice odgovara rednom broju slike u videozapisu, a drugi postotku uspješnosti detekcije. Ukupna uspješnost detekcije je vrlo dobra, čak **84.92%**.

r.b. slike	% detekcije	r.b. slike	% detekcije	r.b. slike	% detekcije
136	94.09	286	86.16	436	84.37
151	94.25	301	67.88	451	83.30
166	86.72	316	71.22	466	78.32
181	78.89	331	78.40	481	80.19
196	92.04	346	89.32	496	85.78
211	93.83	361	96.21	511	78.38
226	91.14	376	80.28	526	74.75
241	96.87	391	91.51	541	86.21
256	82.25	406	90.99		
271	92.92	421	71.36		

Tablica 8.1: Detekcija automobila svijetle boje

U drugom eksperimentu, detektira se automobil tamnije boje. Detekcija je lošija za vrijeme ulaska automobila na parking zbog povećane brzine pa optički tok s istim parametrima kao za prošli eksperiment ne može pronaći sve korespondentne značajke zbog prevelikog pomaka. Kako se približava parkirnom mjestu, automobil smanjuje brzinu pa su neki rezultati detekcije čak i 100%. Ovim eksperimentima je utvrđeno da praktički nema razlike u detekciji različitih boja automobila.

(a) 2661. slika **(b)** 2866. slika **(c)** 3016. slika

Slika 8.4: Detektiranje automobila

U tablici 8.2 prikazana je detekcija automobila tamnije boje pri uparkiravanju. Ukupna uspješnost detekcije je **84.98%**.

r.b. slike	% detekcije	r.b. slike	% detekcije	r.b. slike	% detekcije
2611	54.80	2761	86.23	2911	90.30
2626	57.24	2776	89.26	2926	84.44
2641	68.35	2791	93.69	2941	99.81
2656	85.17	2806	91.19	2956	98.04
2671	68.76	2821	94.18	2971	98.12
2686	77.81	2836	89.32	2986	99.94
2701	67.18	2851	88.38	3001	99.36
2716	65.07	2866	83.33	3016	100
2731	76.61	2881	92.64		
2746	83.90	2896	96.18		

Tablica 8.2: Detekcija automobila tamne boje

9. Zaključak

U ovom radu predložen je postupak lokaliziranja parkirnih mjesta u nadzornom videu. Najprije je potrebno detektirati automobil i zatim ga pratiti. Za detektiranje automobila nisu korištene uobičajene metode poput Haar klasifikatora i modeliranje pozadine, nego se pokušalo eksperimentirati metodom optičkog toka Lucas-Kanade. Optički tok funkcionira na način da u svakoj slici traži korespondentne točke u odnosu na točke na prethodnoj slici, dakle pokušava pronaći pomak neke točke iz prethodne na novoj slici. Kako bismo detektirali točne objekte, tj. isključivo automobile, koristili smo algoritam *RANSAC*. Svaki automobil opisali smo pravokutnikom algoritmom *PCA*za evaluiranje rezultata. Zbog snimanja videa iz ptičje perspektive, ne može se dogoditi da neki objekt zaklanja automobil za vrijeme praćenja. Detekcija automobila navedenim postupkom u radu iznosi oko **85%** što je vrlo dobar rezultat. Kako su videzapisi snimani po danu uz sunčano vrijeme, iz rezultata se lako iščitava da sjena i osvjetljenje u sceni praktički nisu važni za detekciju automobila.

Bilo bi dobro implementirati ovakav sustav u nadzorne kamere u jer je jeftin u odnosu na postojeća rješenja za nadzor parkinga i nudi lokalizaciju parkirnih mjesta.

Međutim, postoje događaji kada detekcija nije uspješna. Ako se ljudi nalaze u sceni i kreću se u društvu, moguće je da će ih sustav detektirati kao da su automobili. Taj problem javlja se zbog tog što je društvo također veća nakupina piksela (kao automobil) koja ima isti pomak kroz slijed slika. Još jedan problem su i drveća, odnosno krošnje, jer se može dogoditi da se pod utjecajem vjetra krošnja njiše i time algoritam pretpostavlja da je to kretanje automobila.

Za daljnji nastavak na ovaj rad, moguće je koristiti drugačije parametre i evaluirati detekciju za različite vremenske prognoze i pokušati napraviti detekciju za noćnu scenu.

LITERATURA

[1]

[2] *Linear least squares*, 2013. URL http://en.wikipedia.org/wiki/Linear_least_squares_(mathematics).

[3] Oliveira L. S. Britto Jr A. S. Silva Jr E. Koerich A. Almeida, P. Pklot - a robust dataset for parking lot classification˙ *Expert Systems with Applications*, 2015. URL http://www.inf.ufpr.br/lesoliveira/download/ESWA2015.pdf.

[4] Sandeep Lokala Ananth Nallamuthu. Vision based parking space classification. URL http://www.ces.clemson.edu/~stb/ece847/projects/Parking_Space_Classification.pdf.

[5] Neven Elezović Andrea Aglić-Aljinović. *Matematika 1*.

[6] OpenCV community. *OpenCV*. URL http://docs.opencv.org.

[7] R. C. Fischlerm M. A., Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. 24(6):381 – 395, 1981.

[8] Antonio Garcia-Dopico, Jose Pedraza, Manuel Nieto, Antonio Perez, Santiago Rodriguez, i Luis Osendi. Locating moving objects in car-driving sequences. 2014(1):24, 2014. ISSN 1687-5281. doi: 10.1186/1687-5281-2014-24. URL http://jivp.eurasipjournals.com/content/2014/1/24.

[9] Bahadir K. Gunturk. *Computer vision - optical flow*, 2007. URL http://www.ece.lsu.edu/gunturk/EE4780/Lecture%20-%20Optical%20Flow.ppt.

[10] Gabriele Facciolo Javier Sanchez, Enric Meinhardt-Llopis. Tv-l1 optical flow estimation. 2012.

[11] Srđan Rašić. *Određivanje prostorno-vremenskih deskriptora nad optičkim tokom*. 2013.

[12] Florian Raudies. Optic flow. 2013. URL http://www.scholarpedia.org/article/Optic_flow#A7_Method_of_Horn_.26_Schunck_.281981.29.

[13] Mubarak Shah. *KLT*. University of Central Florida, 2014. URL http://crcv.ucf.edu/videos/lectures/2014.php.

[14] Horiba I.-Ikeda K. Onodera H. Ueda, K. i S. Ozawa. An algorithm for detecting parking cars by the use of picture processing. 174, 1991. ISSN 1379-1389.

Lokalizacija parkirnih mjesta u nadzornom videu

Sažetak

Razmatramo detekciju događaja uparkiravanja automobila u videu snimljenom nadzornom kamerom. Predlažemo metodu koja se temelji na praćenju točkastih značajki kroz slijed slika. Pokretne objekte detektiramo kao prostorno bliske nakupine piksela s približno jednakim pomakom s obzirom na prethodnu sliku. Objekte predstavljamo opisanim pravokutnikom koji je poravnat s glavnom osi objekta. Ispitni skup sastojao se od videozapisa statične nadzorne kamere iz ptičje perspektive. U implementaciji je korišten programski jezik C++ i vanjska biblioteka OpenCV koja olakšava rad sa slikama. Metodu smo evaluirali nad uparkiravanjem dvaju automobila. Postignuti rezultati su prikazani i diskutirani.

Ključne riječi: Detekcija parkirnih mjesta, računalni vid, optički tok, Lucas-Kanade, PCA, RANSAC, nadzorni video, C++, OpenCV

Localization of parking spots in surveillance video

Abstract

We consider detection of car parking in video recorded by surveillance camera. The method which we propose is based on point features tracking through sequence of photos. We detect moving objects like close clusters of pixels with about same shift according to previous photo. Objects were presented with circumscribed rectangle which is aligned with main axis of object. Test set contains videos recorded by static surveillance camera with bird's-eye view. Solution was implemented in C++ with OpenCV library for easy work with photos. We evaluated method on two cars parking. Achieved results were displayed and discussed.

Keywords: Parking spots detection, computer vision, optical flow, Lucas-Kanade, PCA, RANSAC, surveillance video, C++, OpenCV