Sustav za upozoravanje na nepoželjno napuštanje vozne trake

Sadržaj

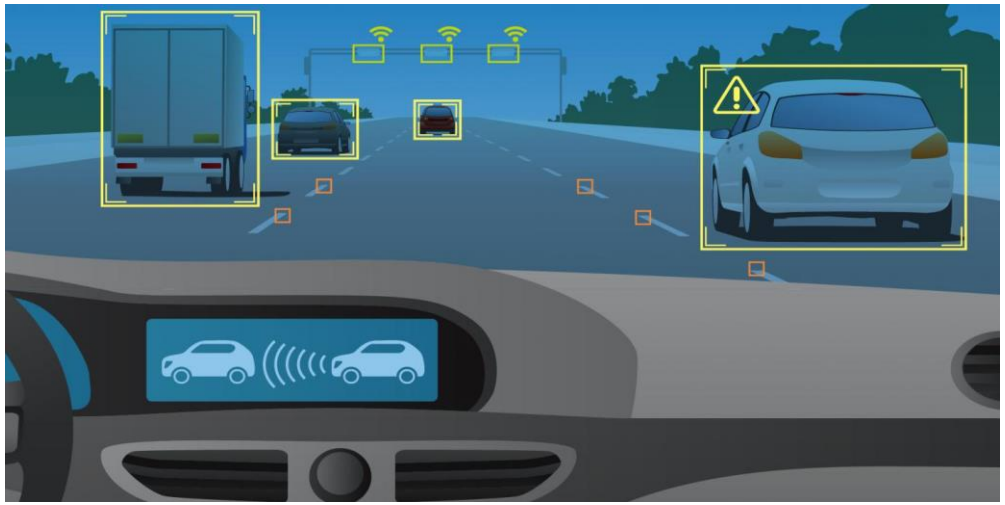
- Sustavi za pomoć vozaču u vožnji - ADAS
- Sustav za upozoravanje vozača na nepoželjno napuštanje vozne trake (eng. Lane Departure Warning System – LDWS)
- Implementacija LDWS u Python-u + OpenCV:
  - **Zadatak 1** – implementacija osnovnog LDWS
  - **Zadatak 2** – implementacija LDWS koji koristi transformaciju perspektive
  - **Zadatak 3** – implementacija LDWS koji transformaciju perspektive i zakrivljene linije za označavanje vozne trake

Što je sustav za upozoravanje vozača na nepoželjno napuštanje vozne trake?

ADAS

- moderna vozila opremljena su različitim sustavima za pomoć vozaču u vožnji – Advanced Driver Assistant Systems (ADAS)

  - surrround view
  - forward collision avoidance system
  - zamjena retrovizora kamerama
  - sustav za upozoravanje na nepoželjno napuštanje vozne trake (eng. lane departure warning system - LDWS)
  - sustav za održavanje vozila u voznoj traci (engl. lane keeping assist system)
  - nadzor vozača (spava li vozač , prati li cestu i sl.)

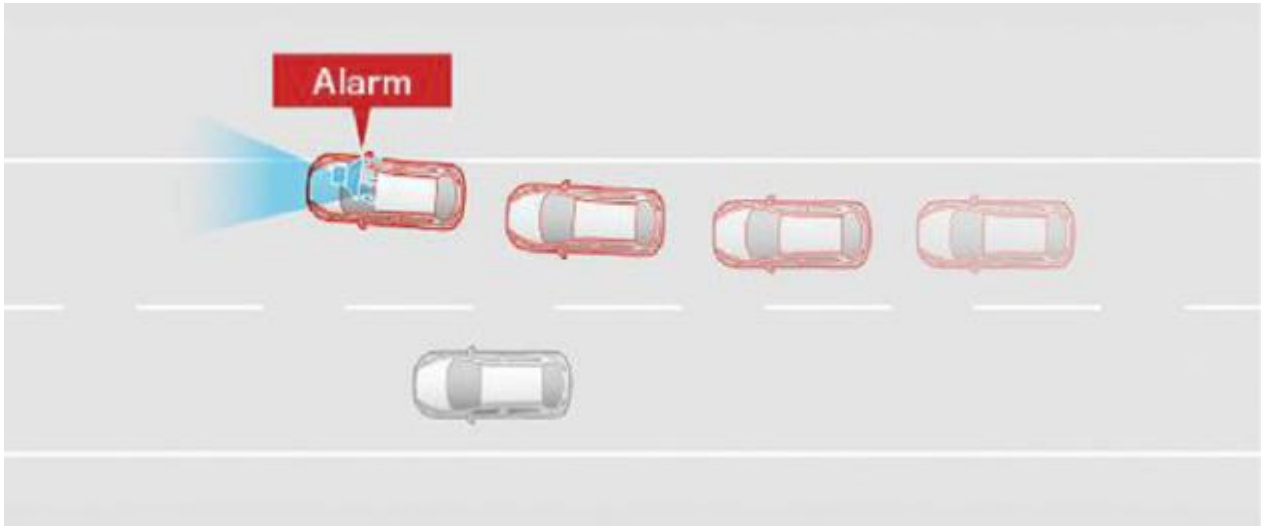


LDWS

- sustav za upozoravanje vozača na nepoželjno napuštanje vozne trake
  - sigurnosni sustav koji upozorava vozača kada nastaje prijelaz iz jednu u drugu prometnu traku, a pokazivač smjera nije uključen
  - praktički detektira nenamjerno napuštanje vozne trake vozila u slučaju nepažnje vozača te ga upozorava odgovarajućim svjetlosnim signalom na instrument tabli i zvučnim signalom
  - ovaj sustav koristi informacije (slike) koje dobiva s kamere montirane na prednjoj strani vozila (najčešće na ili iznad središnjeg retrovizora)

LDWS

- Prema podacima istraživanja Federal Highway Administration (FHWA), upotrebom LDWS-a može se učinkovito izbjeći oko 50% prometnih nesreća uzrokovanih napuštanjem traka
- ako se upozorenje za napuštanje trake može izdati za 0.5 s unaprijed, vozači mogu upravljati vozilom kako bi ispravili smjer vožnje kako bi izbjegli najmanje 60% prometnih nesreća



LDWS

- Primjer: https://www.youtube.com/watch?v=rjSSkKCcOLw
- u mnogim automobilima postoje ograničenja na upotrebu ovog sustava:
  - može se aktivirati ako se vozilo kreće iznad određene brzine (npr. 65 km/h)
  - ravna cesta i blago zakrivljena
  - efikasnost sustava ovisi o vremenskim uvjetima, brzini vozila i uvjetima na kolniku (npr. vidljivost uzdužnih kolničkih oznaka)
  - napredniji sustavi (npr. Toyota) prati i vozilo ispred sebe ako ono postoji (jer u tom slučaju mogu biti zaklonjene kolničke oznake) te omogućuju automatski zakret volana kako bi centrirale vozilo
    - ne može se smatrati autonomnom vožnjom jer čim vozač makne ruke s volana sustav se isključuje
    - https://www.youtube.com/watch?v=QVyRsdILbRw

LDWS

- LDWS općenito uključuje:
  - filtriranje slike i detekcija rubova prometnih traka
  - opis detektiranih linija pravcima ili krivuljama drugog reda
  - detekcija napuštanja vozne trake i slanje upozorenja vozaču
  - također može davati informaciju o položaju vozila unutar vozne trake i zakrivljenosti kolnika

Zadatak 1 - Implementacija osnovnog LDWS

Zadatak 1 – implementacija osnovnog LDWS

- **Ulaz** u algoritam:
  - video signal, odnosno video okviri dobiveni s kamere montirane na prednjoj strani vozila
- **Izlaz**:
  - prikaz slike na ekranu pri čemu je vlastita vozna traka prikazano zelenom bojom (overlay) kada se vozilo nalazi u prometnoj traci
  - ispis poruke upozorenja kod napuštanja vozne trake
  - izračunati brzinu obrade te ju ispisati na svakom video okviru ili u command prompt

- za izradu navedenog algoritma koristit ćemo standardne metode iz područja računalnog vida i biblioteku OpenCV
- detektirati što više prelazaka iz jedne trake u drugu uz što manje lažnih upozorenja; pokušati s istim parametrima na različitim video signalima

Zadatak 1 – output

- primjer

Zadatak 1 – output

- primjer

Zadatak 1 – načelni blok dijagram algoritma

Ulazna slika

Rezultat

- ovo je prijedlog algoritma, možete ubaciti i ostale korake (filtriranja i sl.)
- npr. erozija binarne slike na kojoj se nalaze detektirani rubovi

1. Učitavanje video signala s diska okvir po okvir

- video signali se nalaze u poddirektoriju `/Videos`
- skriptu nadopunjavati na `TODO` mjestima:

- otvorite `LWS_1.py` skriptu te ju nadopunite na `TODO` mjestima kako biste omogucili:
  - učitavanje željenog video signala pomoću OpenCV naredbe `cv2.VideoCapture()`
  - pomoću kreiranog objekta moguće je čitati okvir po okvir iz video signala pomoću funkcije `read()`
  - okvire čitajte u beskonačnoj petlji i prikazujte ih na ekranu pomoću naredbe `cv2.imshow()`

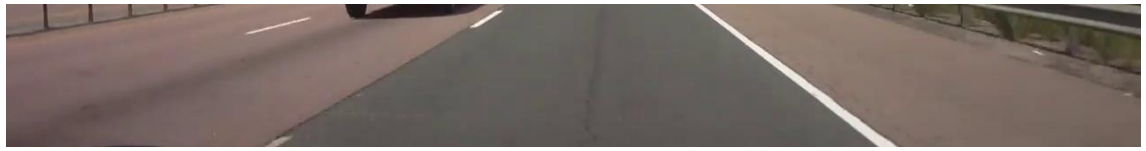
2. Izdvajanje regije od interesa - RoI

- mogu li se linije vozne trake pojaviti u gornjem dijelu slike? obično je gornji dio slike nebo :)
- ako je kamera montirana unutar vozila moguće je vidjeti u donjem dijelu slike haubu vozila
- pametno je izdvojiti samo regiju od interesa (engl. Region of Interest - RoI)
- izbacivanjem dijelova koji ne mogu sadržavati linije vozne trake značajno smanjujemo količinu piksela koji se obrađuju u narednim koracima algoritma --> važno za implementaciju na embedded platformu

2. Izdvajanje regije od interesa - RoI

- možemo li to još smanjiti?

koristite izdvajanje dijelova iz numpy polja

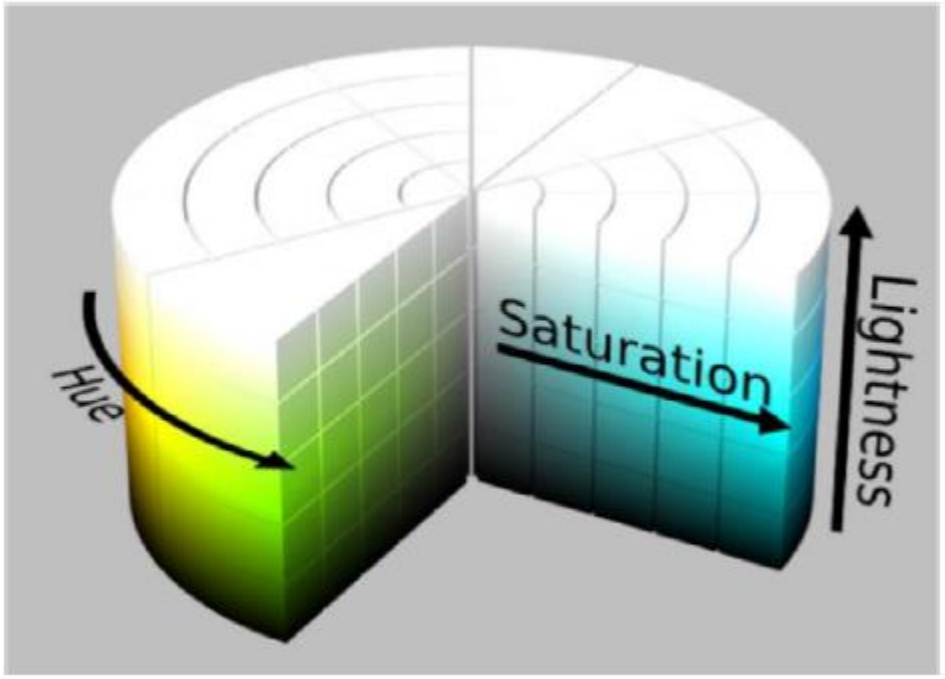


3. Filtriranje po boji

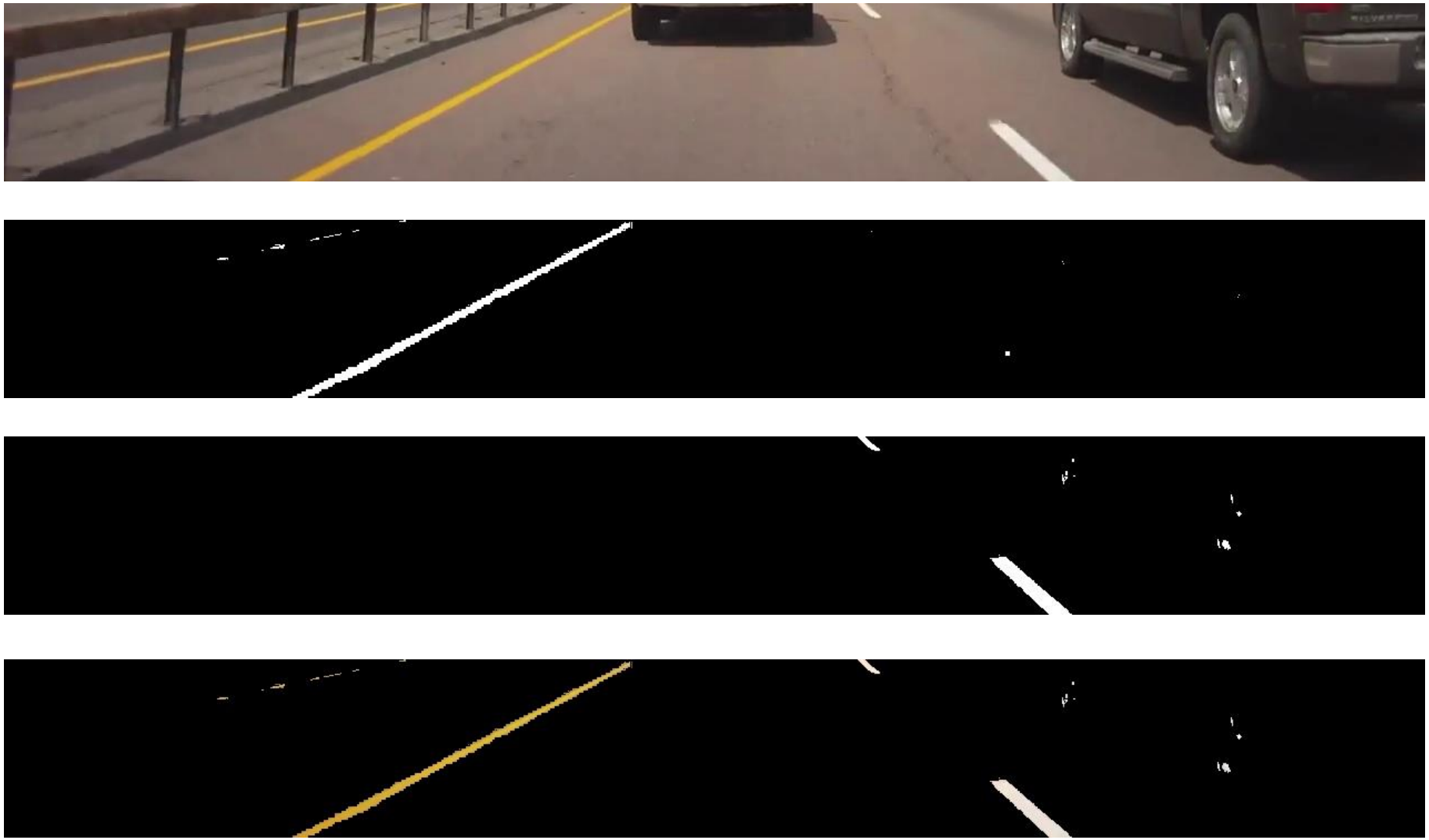
- uzdužne kolničke oznake mogu se pojaviti u dvije boje
  - bijela
  - žuta
- stoga je potrebno filtrirati sliku tako da na slici ostanu samo objekti bijele boje i žute boje

3. Filtriranje po boji

- efikasno se može napraviti u HSL prostoru boja
- koristite OpenCV funkcije: `cv2.cvtColor()`, `cv2.inRange()`



3. Filtriranje po boji



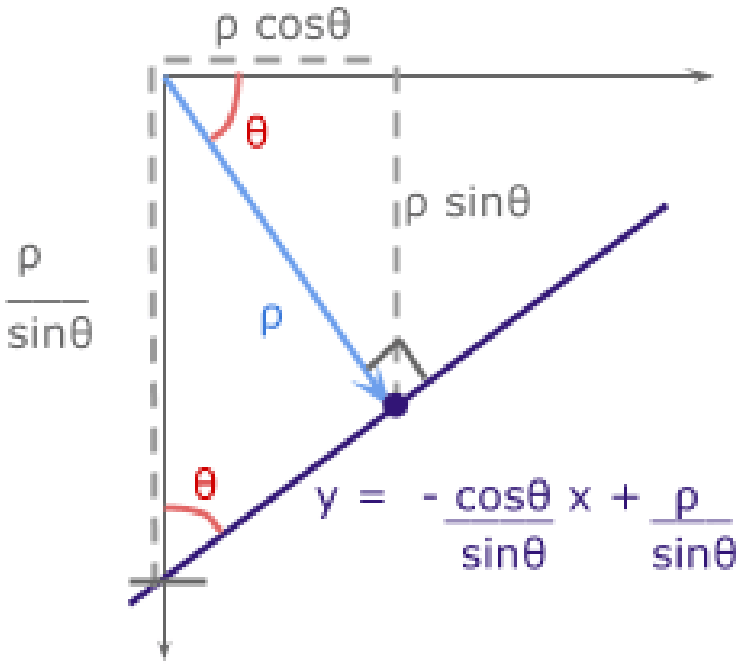
4. Detekcija rubova

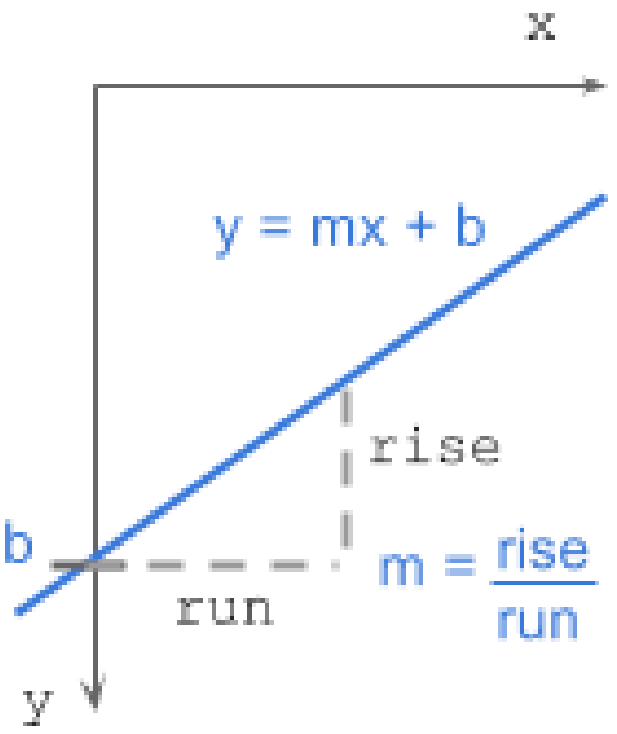
- primijenite detektor rubova na filtriranu sliku kako bi smanjili količinu piksela koja se obrađuje u sljedećem koraku (samo nam je potreban rub uzdužne kolničke linije)
- Canny, Sobel…
- moguće se riješiti dodatnog „smeća” morfološkim filtriranjem

4. Detekcija pravaca

- na dobivenoj binarnoj slici s rubovima potrebno je odrediti dva pravca koja odgovaraju lijevoj i desnoj kolničkoj oznaci
- Ovdje ćemo iskorisiti Houghovu transformaciju:
  - `cv2.HoughLinesP()`
- obratiti pozornost što vraća funkcija

4. Detekcija pravaca





4. Detekcija pravaca

- nakon što se provede Houghova transformacija trebaju se filtrirati vrijednosti – ne pripadaju svi pravci kolničkim oznakama
- potrebno ih je filtrirati → uzdužne kolničke oznake mogu se pojaviti samo pod određenim kutem
  - provjera nagiba detektiranog pravca
  - odaberite lokalni maksimum koji odgovara lijevoj uzdužnoj oznaci i lokalni maksimum koji odgovara desnoj uzdužnoj oznaci
  - Budući da radimo s jednadžbom pravca y=ax+b, pripazite na pojavu okomitih pravaca

4. Detekcija pravaca

- Plavo su označene dužine koje daje funkcija `cv2.HoughLinesP()`

5. Označavanje vozne trake i upozoravanje

- kada ste odredili pravac koji odgovara lijevoj i desnoj uzdužnoj oznaci, tada označite voznu traku sa zelenom površinom (overlay) u ulaznoj slici
- napravite konturu od 4 točke i ispunite ju zelenom bojom
- OpenCV funkcije:
  - `cv2.fillPoly()`
  - `cv2.addWeighted()`
- uzmite u obzir da su detektirani pravci definirani s obzirom na RoI, a ne na ulaznu sliku s kamere!

5. Označavanje vozne trake i upozoravanje

- u slučaju prelaska vozila u drugu voznu traku isključite zelenu oznaku vozne trake i prikažite upozorenje na ekranu
- razmislite što se događa s nagibima i udaljenosti detektiranih pravaca od ishodišta u slučaju prelaska u drugu voznu liniju → potrebni su odgovarajući uvjeti (rasponi), a koje ćemo koristiti za signalizacju „Upozorenja”
- dodavanje teksta na video okvir moguće je pomoću naredbe `cv2.putText()`

Izračunavanje brzine izvođenja algoritma

- mjerenje vremena obrade odnosno rada algoritma moguće je dobiti na način:

```
e1 = cv.getTickCount()
# ovdje ide vas algoritam
e2 = cv.getTickCount()
time = (e2 - e1)/ cv.getTickFrequency()
```

- koliko okvira algoritam može obraditi u sekundi (engl. Frames Per Second) dobiva se kao `int(1.0/time)`

Ostalo

- dodatne opcije koje mogu pomoći prilikom razvoja algoritma:
  - prikaz i spremanje međurezultata na disk pomoću funkcija `cv2.imshow()` i `cv2.imwrite()`
- Koristite debugger!!!
- i sve ostalo što vam može pomoći kako biste razvili što bolje rješenje…

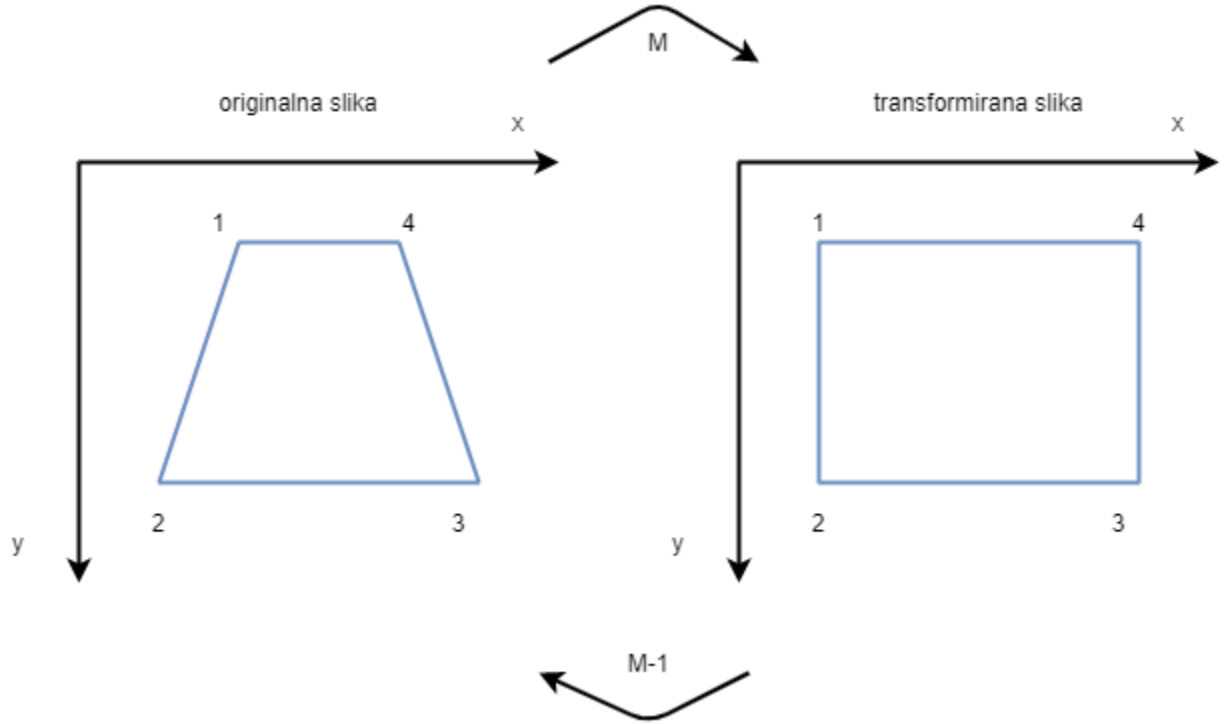
Zadatak 2 - Implementacija DWS koji koristi transformaciju perspektive

Implementacija LDWS koji koristi transformaciju perspektive

- U ovom zadatku potrebno je koristiti transformaciju perspektive – dobiti pogled odzgor na prometnu traku
- Budući da su u bliskom polju automobila uzdužne kolničke oznake približno ravne, one će u transformiranoj slici biti približno okomite
- U tom slučaju se može koristiti „histogram” kako bi se detektirale pozicije uzdužnih kolničkih oznaka – sumira se binarna slika po vertikali

Transformacija perspektive

- u okviru algoritma slika dobivena s kamere se prebacuje u „pogled odozgor”
- za ovo je potrebno pronaći četiri točke na ulaznoj slici koje kada se gledaju od gore definiraju 4 pravca koja su paralelna
- Točke se pohranjuju u numpy polja



Zadatak 2 – načelni blok dijagram algoritma

Ulazna slika

Rezultat

1. Filtriranje po boji

- Slično kao u prethodnom primjeru, potrebno je ulazni okvir filtrirati po boji (propustiti bijelu i žutu boju)

2. Transformacija perspektive

- Kako biste definirali željeno područje možete napisati funkciju koja ga crta pomoću `cv2.line()` kako biste bili sigurni da ste dobro odredili područje

1. Transformacija perspektive

- kreirajte numpy polje `dst` sa željenim pozicijama sve 4 točke s ulazne slike
- pozovite OpenCV funkciju kako biste dobili matricu perspektivne transformacije:

```
M = cv2.getPerspectiveTransform(src, dst)
M_inv = cv2.getPerspectiveTransform(dst, src)
```

- sada je moguće koristiti funkciju `cv2.warpPerspective()` kako biste dobili od ulazne slike pogled odozgor

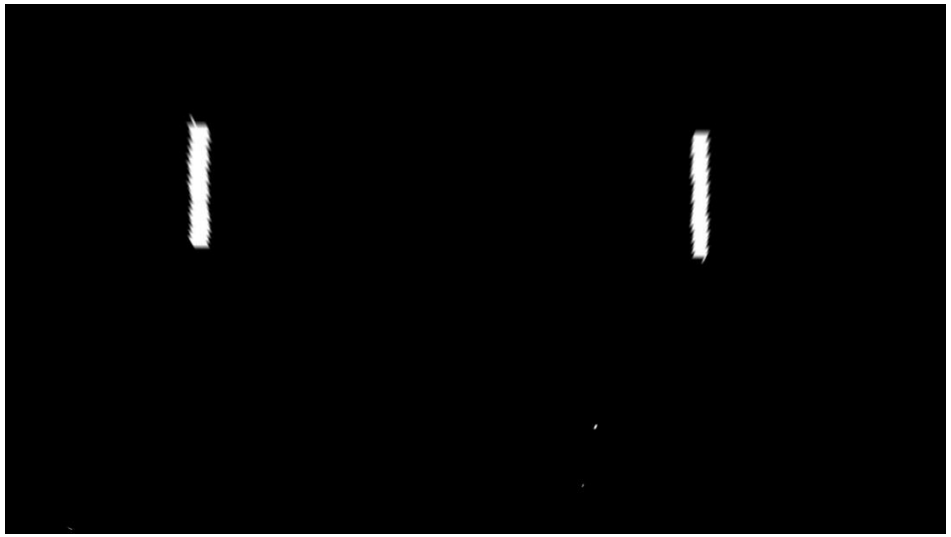
2. Transformacija perspektive dijela filtrirane slike

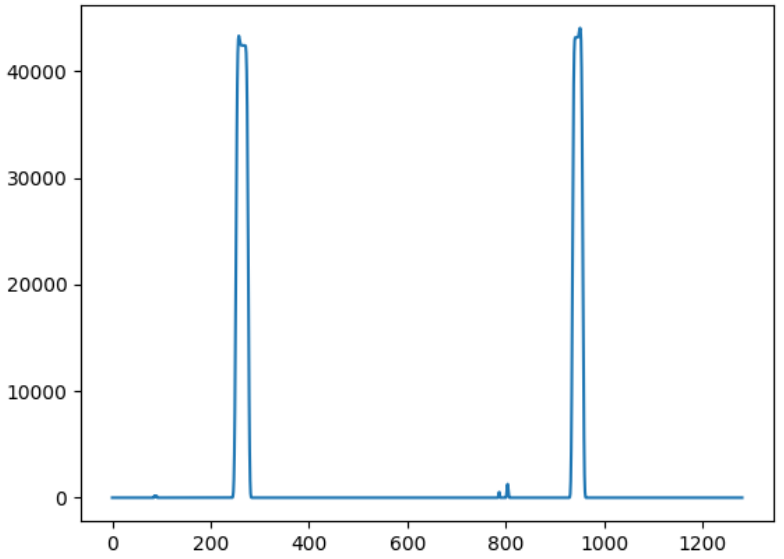
M

M^{-1}

3. Detekcija vrhova u transformiranoj slici

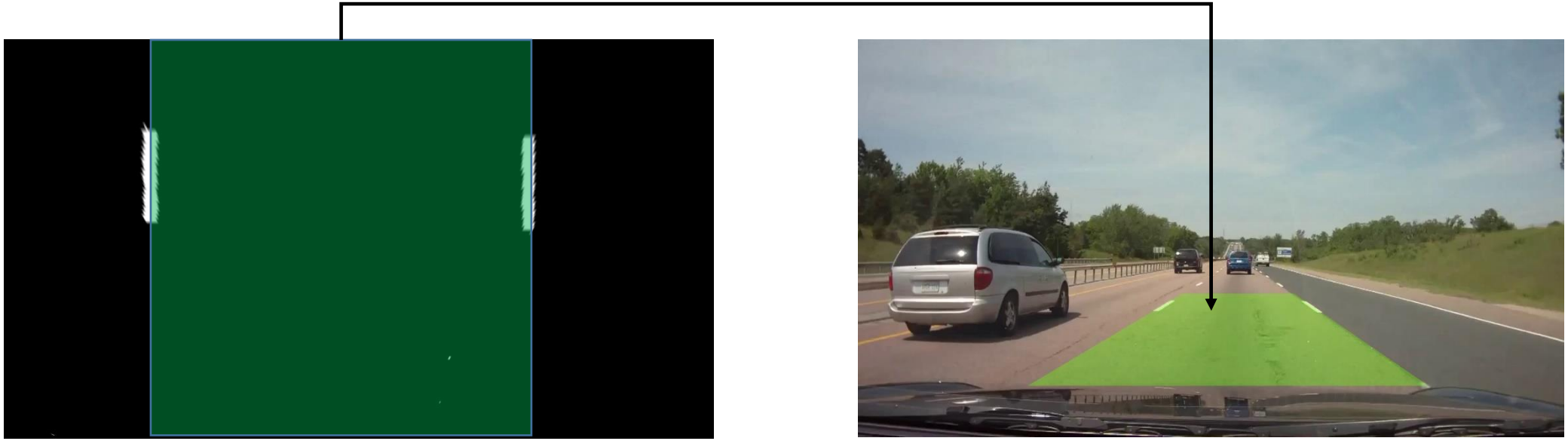
- Ako se transformirana slika „sumira” po vertikalnoj osi, tada se dobiva krivulja koja ima dva izražena vrha
- x koordinate ovih vrhova definiraju položaj pravaca koji odgovaraju uzdužnim kolničkim oznakama u transformiranoj slici





4. Označavanje vozne trake u ulaznoj video okviru i upozoravanje

- Detektirani vrhovi definiraju područje u transformiranoj slici (4 točke) koje se pomoću matrice M^(-1) transformiraju na originalnu video okvir



Zadatak 3 - Implementacija LDWS koji koristi transformaciju perspektive i zakrivljene linije

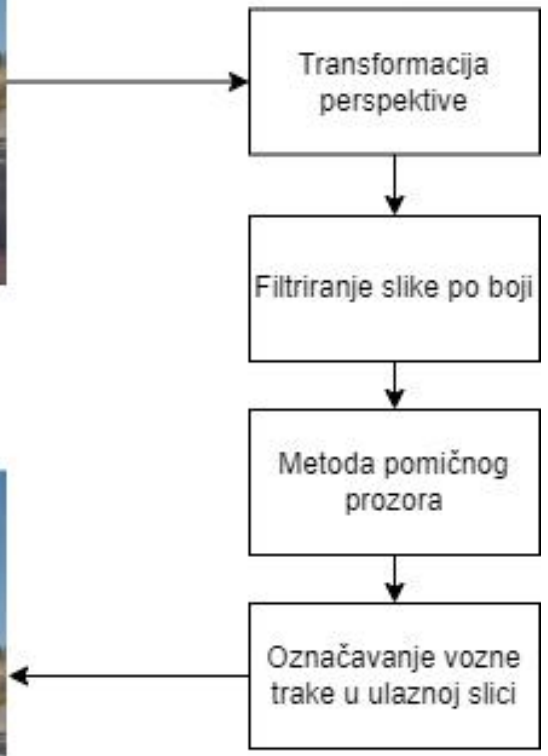
Implementacija LDWS koji koristi transformaciju perspektive i zakrivljene linije

- linije uzdužnih kolničkih linija mogu se aproksimirati pravcima u blizini vozila i kada je kolnik ravan
- međutim ako se hoće postići veća prezicnost detekcije vozne trake, tada je potrebno uzeti u obzir da kolnik ima odgovarajuću zakrivljenost (zavoji) te da kolničke oznake treba aproksimirati krivuljom drugog reda
- kako bi se olakšao postupak procjene navedenih krivulja, potrebno je sliku koja se dobiva s kamere montirane na prednjoj strani vozila transformirati u pogled odozgor te primijeniti metodu pomičnog prozora kako bi detektirali piksele koji pripadaju lijevoj odnosno desnoj kolničkoj oznaci

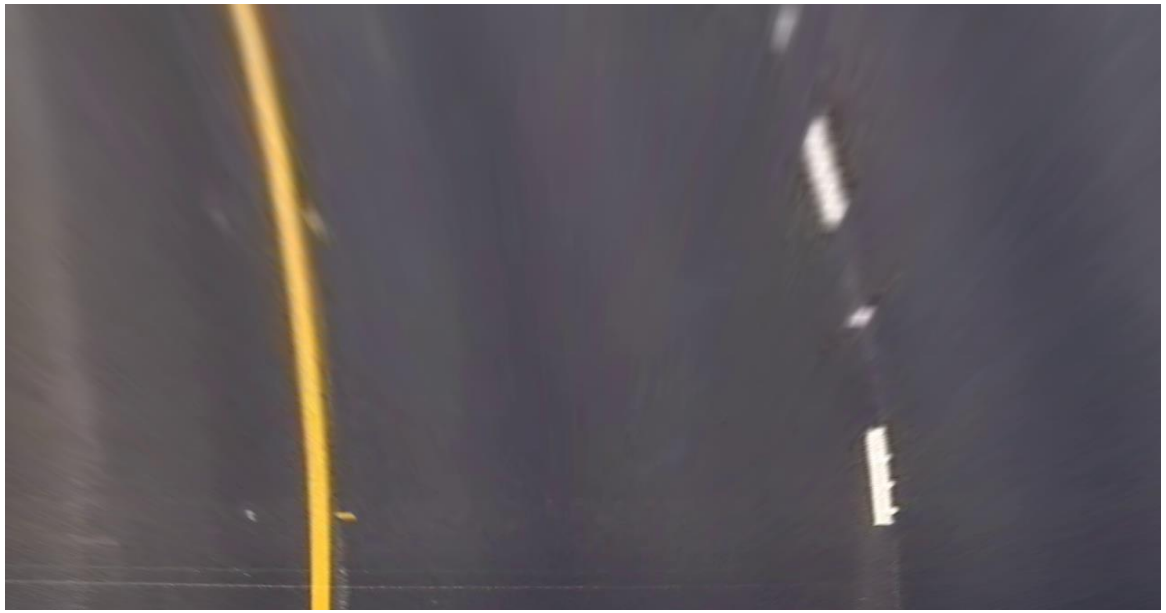
Zadatak 3 – načelni blok dijagram algoritma

Ulazna slika

Rezultat

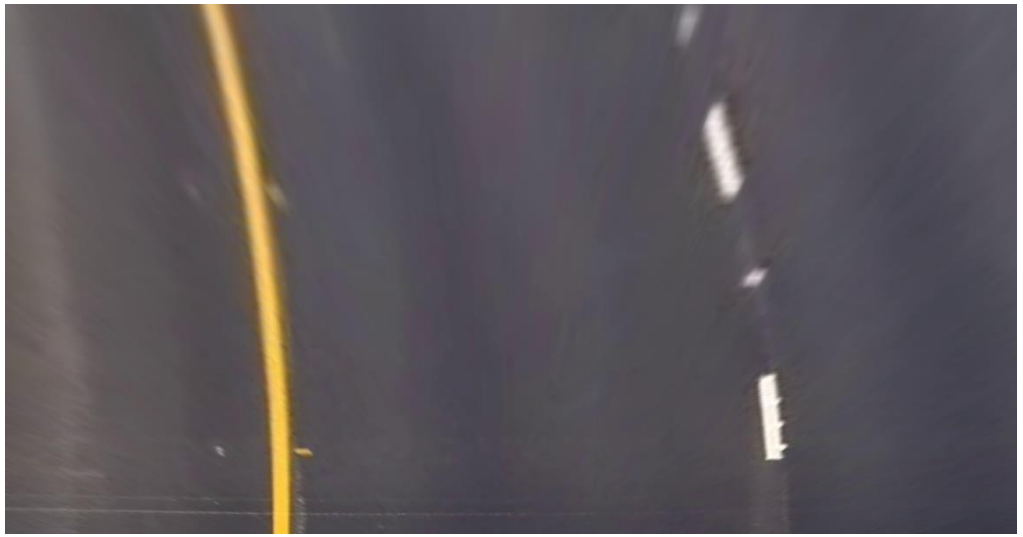


1. Transformacija perspektive



2. Filtriranje po boji

- slično kao i u osnovnom algoritmu
- koristite OpenCV funkcije: `cv2.cvtColor()`, `cv2.inRange()`

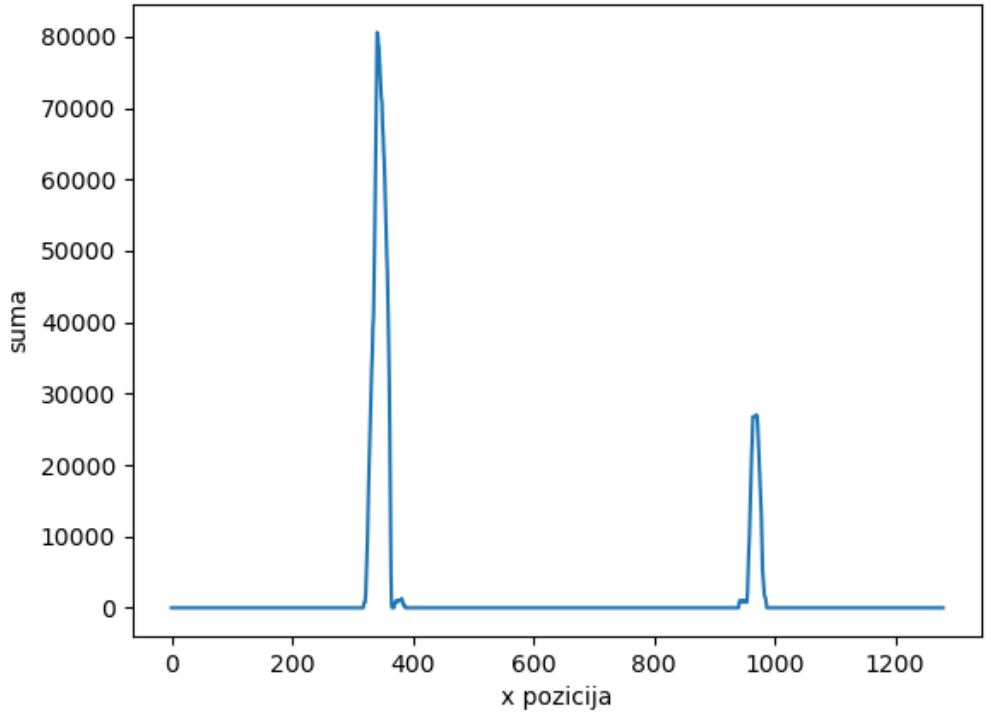


3. Metoda pomičnog prozora

- ovom metodom izdvajaju se pikseli koji pripadaju lijevoj i desnoj uzdužnoj kolničkoj oznaci
- filtrirana slika se konvertira u binarnu sliku, te se analizira samo donji dio slike (npr. donja polovica) – tu su linije na slici gotovo ravne
- zatim se sumiraju stupci te slike i prikaže se rezultat → horizontalna os predstavlja poziciju stupca, a vertikalna os sumu piksela
- moguće je uočiti dva vrha koji odgovaraju pravcima na slici i predstavljat će ishodište za pretraživanje slike (određivanja piksela koji pripadaju lijevoj odnosno desnoj kolničkoj oznaci) → krenut ćemo od dolje s prozorom

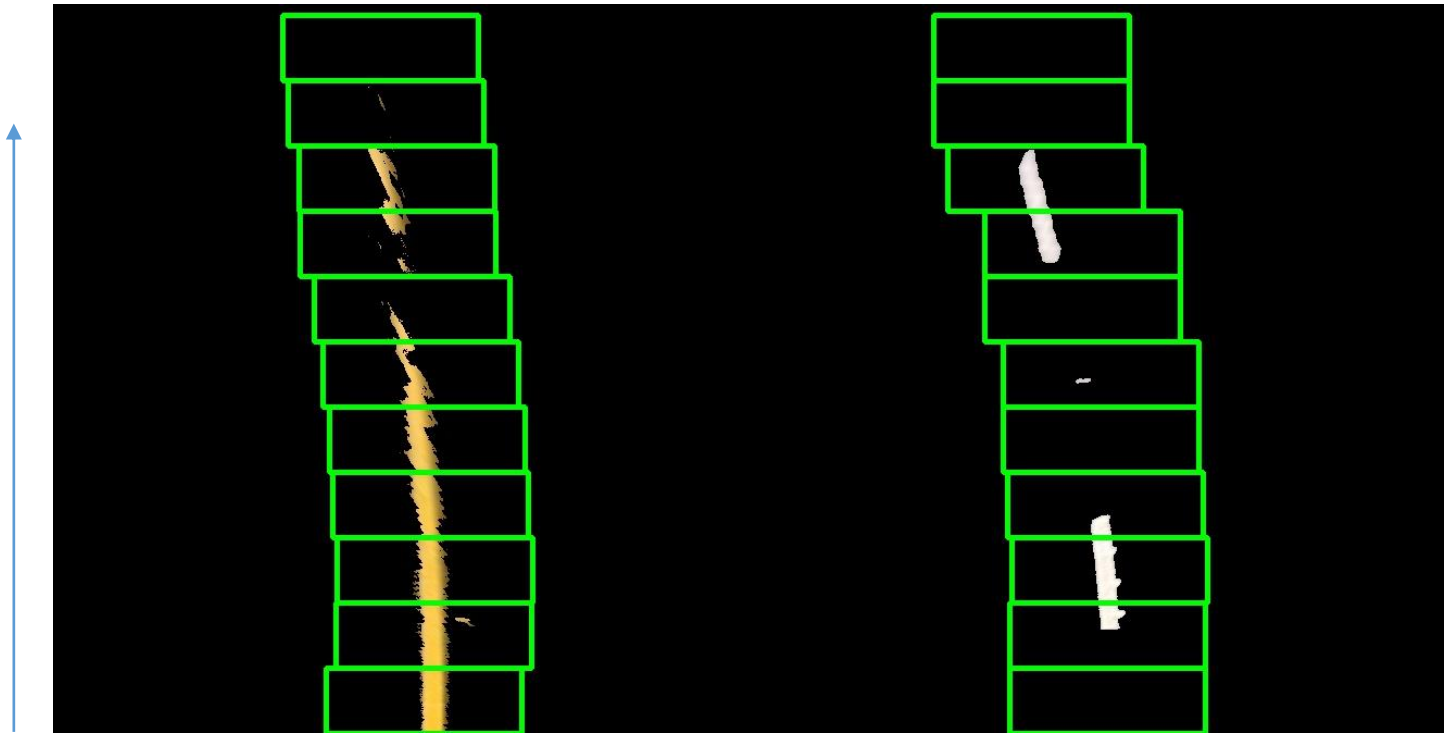
3. Metoda pomičnog prozora

donja polovica slike



3. Metoda pomičnog prozora

- ovi dva peaka predstavljaju početne pozicije malih pomičnih prozora – unutar svakog izračunavamo srednju vrijednost piksela (bijelih) → to je centar sljedećeg prozora po x-osi, dok se po y-osi pomičemo za visinu prozora
- svodi se na izdvajanje dijela iz numpy polja i pridruživanje piksela u novo polje



3. Metoda pomičnog prozora

- na raspolaganju su četiri numpy polja koja sadrže pozicije piksela koji pripadaju lijevoj odnosnoj desno kolničkoj oznaci,
  - `x_left, y_left`
  - `x_right, y_right`
- fitanje funkcije drugog reda kroz piksele s koordinatama spremljenim u numpy polja x i y

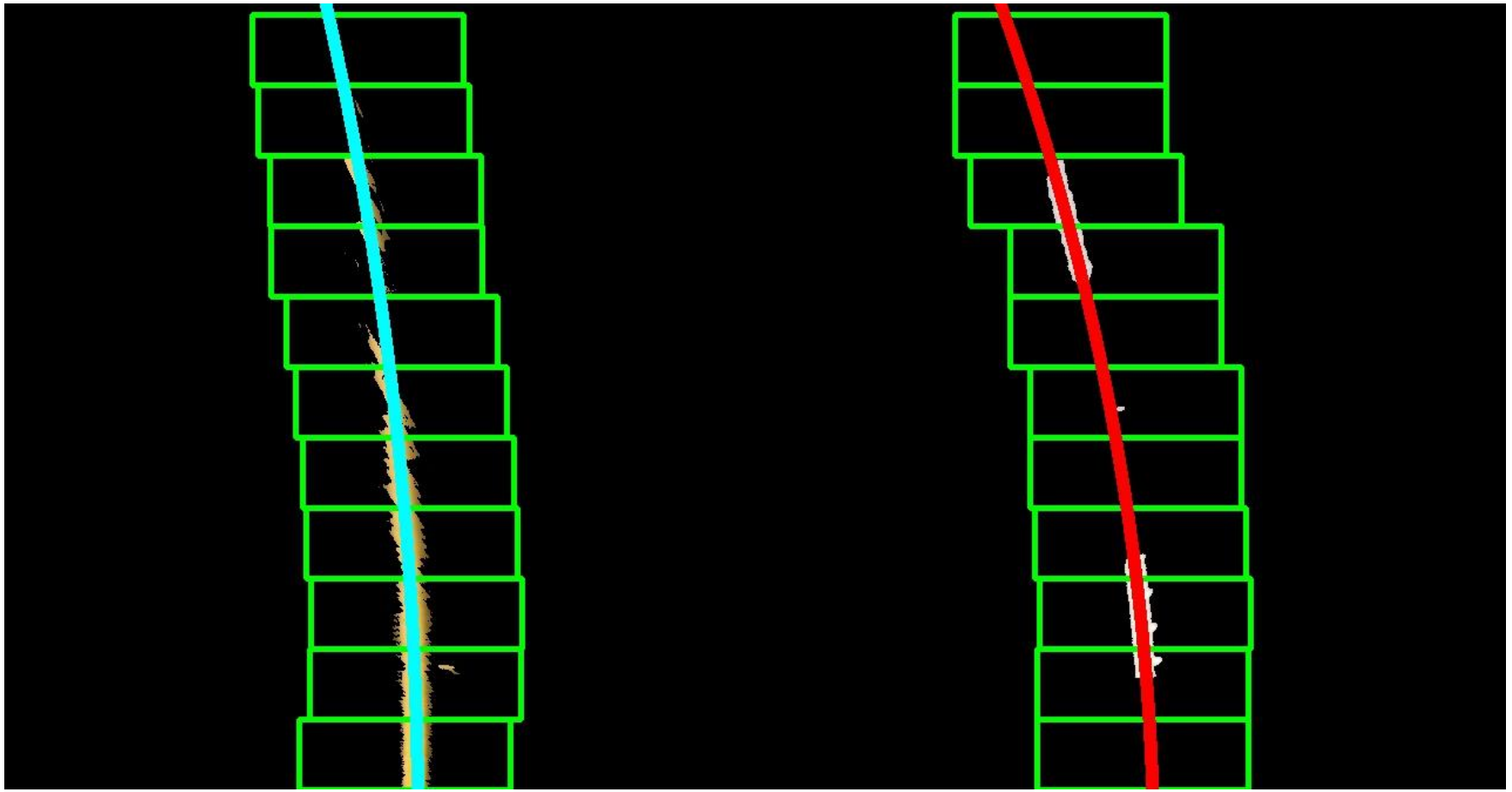
```
line = np.polyfit(x,y,2)
```

- funkcija je oblika:

```
x = line[0] * (y**2) +  line[1] * y + line[2]
```

- sada se može nacrtati krivulja na transformiranoj slici

3. Metoda pomičnog prozora



4. Prikaz vozne trake na ulaznoj slici

- krajnjem korisniku potrebno je prikazati voznu traku na ulaznoj slici
- stoga je potrebno transformirati točke koje pripadaju lijevoj i desnoj krivulji pomoću inverzne matrice M (dobili smo ju u prvom koraku)
- OpenCV funkcija:

  `cv2.perspectiveTransform()`
- obilježavanje vozne trake (overlay) radi se na isti način kao kod jednostavnog algoritma

5. Prikaz vozne trake na ulaznoj slici