**Детекција и броење на возила**

**Дигитално процесирање на слика**

Никола Здравески Никола Недески

211054 211208

ФИНКИ

*Апстракт*—Оваа студија претставува проект за детекција и броење на возила во видео записи користејќи Python и OpenCV. Главната цел на проектот е автоматско идентификување и следење на возилата кои поминуваат одредена линија во видео кадрите. Системот ги обработува видео фрејмовите со примена на техники за обработка на слики како што се разлика помеѓу фрејмови, конверзија во сива скала, филтрирање и бинаризација. Потоа, се идентификуваат контурите на објектите и се пресметуваат нивните центроиди за понатамошно следење. За секое возило се доделува уникатен идентификатор, и се следи низ видео кадрите додека не се изброи преминувањето на линијата. Системот користи параметри како минимална големина на контури, максимално растојание помеѓу следените центроиди и максимална возраст на следените објекти за прецизно следење и броење на возилата. Овој проект е применлив за анализа на сообраќај, подобрување на управувањето со сообраќајот и собирање статистички податоци за сообраќајниот проток.

# Вовед

Системите за детекција на возила се користат за автоматско откривање и броење на возила во одредена област. Овие системи се применуваат во различни области како што се управување со сообраќајот, управување со паркинзи, наплата на патарина и урбано планирање. Во денешниот брзоразвивачки свет, каде што бројот на возила значително се зголемува, потребата за ефективно и ефикасно управување со сообраќајот станува се поактуелна.

Покрај тоа, зголемувањето на бројот на возила на патиштата предизвикува проблеми како што се сообраќајни застои, зголемена емисија на штетни гасови и поголем ризик од сообраќајни незгоди. Затоа, постои потреба за интелигентни системи кои можат да го олеснат управувањето со сообраќајот преку автоматско детектирање и броење на возилата. Овие системи можат да помогнат во анализа на сообраќајот, идентификација на густината на сообраќајот и донесување на одлуки за оптимизација на сообраќајниот проток.

Овој проект е фокусиран на развивање на систем за детекција и броење на возила во видео записи користејќи Python и OpenCV. Главната цел е автоматско идентификување и следење на возилата кои поминуваат одредена линија во видео кадрите. Со користење на напредни техники за дигитална обработка на слики и машинско учење, овој систем ќе може да ги процесира видео фрејмовите и да ги идентификува возилата со голема прецизност.

***Компоненти на Системот***

*Сензори*

* Камери: Камерите се користат за снимање на видео и слики од областа за детекција. Камерите се поставуваат на стратешки локации за да обезбедат добар преглед.
* LIDAR: Користи ласерска светлина за да создаде 3D мапи на околината и прецизно открива објекти.
* Радар: Користи радиобранови за да открие објекти и нивната брзина.
* Инфрацрвени сензори: Користат топлински потписи за откривање на возила.

*Единица за обработка*

* Централен процесор (CPU) и графички процесор (GPU) кои извршуваат алгоритми за детекција и обработка на слики.

*Софтвер*

* Алгоритми за компјутерска визија: Техники како што се детекција на објекти, сегментација на слики и класификација за идентификација на возила. Популарни алгоритми се YOLO (You Only Look Once), SSD (Single Shot MultiBox Detector) и Faster R-CNN (Region-based Convolutional Neural Networks).
* Модели за машинско учење: Тренирани со големи сетови на податоци за препознавање и класификација на различни видови на возила.
* Анализа на податоци: Софтверски алатки за анализа на сообраќајни податоци, броење на возила и генерирање на извештаи.

*Модули за комуникација*

* Пренос на податоци од сензорите до централни сервери или облачни платформи за понатамошна обработка и анализа. Ова може да вклучува безжични технологии како Wi-Fi, LTE или 5G.

***Работен Принцип***

*Снимање на податоци*

* Сензорите (камери, LIDAR, радар и др.) континуирано снимаат податоци од областите каде е потребна детекција на возила.

*Предобработка на податоците*

* Суровите податоци се обработуваат за да се подобри квалитетот и да се извлечат релевантни карактеристики. Ова може да вклучува техники како намалување на шум, подобрување на контраст и селекција на рамки.

*Детекција на возила*

* Преработените податоци се внесуваат во алгоритми за компјутерска визија и машинско учење. Алгоритмите идентификуваат возила во сликите или видеата врз основа на облици, шаблони и карактеристики.

*Броење на возила*

* Откако возилата се детектирани, системот ги следи нивните движења за да обезбеди секое возило да биде броено само еднаш. Ова се постигнува со користење на алгоритми за следење на објекти.

*Анализа и извештаи*

* Податоците за детекција и броење на возила се анализираат за да се добијат увиди како што се густина на сообраќај, брзина на проток на возила и врвни часови на сообраќај. Генерираат се извештаи за донесување одлуки.

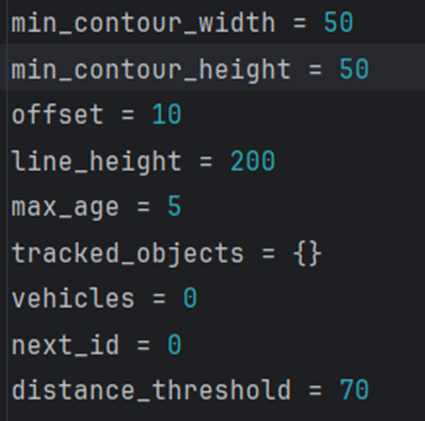
***Примена***

* Управување со сообраќај: Реално време следење на сообраќајот за да се намали застојот и да се подобри безбедноста на патиштата.
* Управување со паркинг: Следење и управување на паркинг простори во реално време за оптимизација на користењето и намалување на времето за барање паркинг место.
* Наплата на патарина: Автоматска детекција и броење на возила за ефикасна наплата на патарина без потреба од рачна интервенција.
* Урбано планирање: Увиди врз основа на сообраќајни модели за подобро планирање и развој на урбана инфраструктура.

Системите за детекција на возила се клучни за модерното управување со сообраќајот и урбаното планирање, помагајќи да се зголеми ефикасноста и безбедноста. Со правилна имплементација и одржување, овие системи можат значително да го подобрат управувањето со сообраќајот и инфраструктурата.

# Процесирање на рамка

## Иницијализација и константи



**min\_contour\_width и min\_contour\_height:**

Овие ги дефинираат минималната ширина и висина на контурите кои се сметаат за валидни. Ова е за да се филтрираат помалите објекти или шумот кој можеби не е релевантен (на пример, мали остатоци или сенки).

За ова видео овие вредности се поставени на 50, што значи дека контурите помали од 50 пиксели во ширина или висина ќе бидат игнорирани.

Ова не важи са секое видео кое ке биде разгледувано, за секое видео треба да се внесат разлицни вредности бидејќи во секое видео објектите ке имаат различна димензија.

**offset:**

Оваа вредност се користи за да се создаде бафер зона околу line\_height. Помага во одредувањето дали возилото е доволно блиску до линијата за да биде изброено.

**line\_height:**

Ова е вертикалната позиција на линијата преку која се бројат возилата.

За ова видео е поставено на 200. За други видеа ке има различна вредност.

**max\_age:**

Максималниот број на фрејмови кои објектот може да постои без повторно да биде детектиран пред да биде отстранет од следење. Ова помага во управување со животниот циклус на следените објекти и отстранување на оние кои повеќе не се во фрејмот.

**tracked\_objects:**

Листа за следење на објекти кои моментално се следат. Секој објект добива уникатен ID и има асоцирани информации како позицијата на центроидот и возраста

**vehicles:**

Бројач за следење на бројот на возила кои биле детектирани при преминување на линијата.

**next\_id:**

Уникатен идентификатор за следниот објект што ќе биде следен. Ова осигурува дека секој следен објект има уникатен ID.

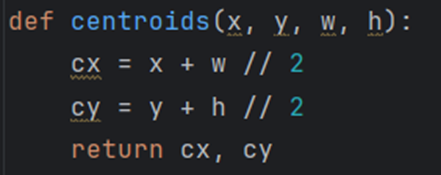
**distance\_threshold:**

Максималното растојание дозволено помеѓу центроидите на контурите во последователни фрејмови за да се сметаат за ист објект. Ова помага во асоцирање на новите детекции со постојните следени објекти.

**Процесирање на рамка:**

* **Пресметка на цетроиди:**

Следната функција е дизајнирана за пресметка на центроидите на правоаголни објекти. Центроидот на правоаголник е точката која го претставува неговиот геометриски центар.



Оваа линија дефинира функција centroids која зема четири параметри:

x: x-координата на горниот лев агол на правоаголникот.

y: y-координата на горниот лев агол на правоаголникот.

w: ширина на правоаголникот.

h: висина на правоаголникот.

cx е x-координатата на центроидот. Таа се пресметува како x-координата на горниот лев агол на правоаголникот плус половина од неговата ширина.

cy е y-координатата на центроидот. Таа се пресметува како y-координата на горниот лев агол на правоаголникот плус половина од неговата висина.

**Пример за Користење**

Доколку имаме правоаголник со горен лев агол на координати (50, 100), ширина од 200 пиксели и висина од 150 пиксели, функцијата centroids ќе ги пресмета координатите на неговиот центроид на следниов начин:

x, y, w, h = 50, 100, 200, 150

center = centroids(x, y, w, h)

print(center) # Излез: (150, 175)

**Читање на видео**



Следниот код е дел од процесот за читање и обработка на видео фајл користејќи ја библиотеката OpenCV. Кодот вклучува отворање на видео фајлот, поставување на резолуцијата и читање на првите две рамки од видеото.

cap = cv2.VideoCapture('Video.mp4')

• Оваа линија отвора видео фајл наречен 'Video.mp4' и создава објект cap кој се користи за пристап до видео податоците. Функцијата cv2.VideoCapture е дел од OpenCV и овозможува читање на видео фајлови.

**Поставување на резолуцијата**

cap.set(3, 1920)

cap.set(4, 1080)

• Овие линии ја поставуваат резолуцијата на видео прозорецот. cap.set(3, 1920) поставува ширина на видео прозорецот на 1920 пиксели, додека cap.set(4, 1080) поставува висина на видео прозорецот на 1080 пиксели.

**Проверка дали видео фајлот е отворен**

if cap.isOpened():

ret, frame1 = cap.read()

else:

ret = False

• Оваа условна структура проверува дали видео фајлот е успешно отворен со cap.isOpened(). Ако видео фајлот е отворен, првата рамка од видеото се чита и се зачувува во променливата frame1. Ако не е отворен, променливата ret се поставува на False.

**Читање на првите две рамки**

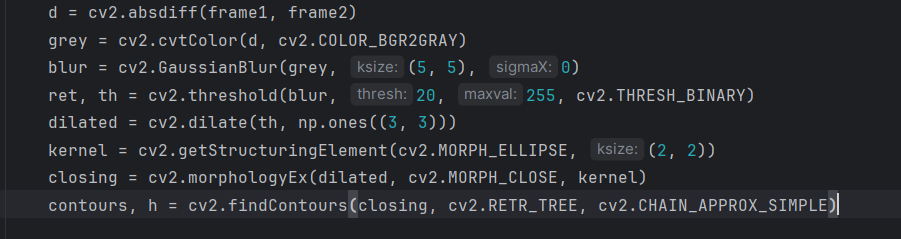
ret, frame1 = cap.read()

ret, frame2 = cap.read()

• Овие две линии читаат две последователни рамки од видеото и ги зачувуваат во променливите frame1 и frame2. cap.read() ја чита следната рамка од видеото и враќа две вредности: ret (булева вредност која покажува дали читањето било успешно) и frame (рамката која е прочитана).

**Диференцијација и обработка на рамка**

Следниот код е дел од процесот за обработка на слики кој се користи за детекција на возила. Кодот ја конвертира бојата на сликата во сива, ја применува Гаусова замагленост, праговна трансформација и морфолошки операции за да ги извлече контурите на објектите на сликата.



**Конверзија во greyscale**

Грејскал сликите се основен дел од компјутерската визија и обработка на слики. Тие ја намалуваат комплексноста и овозможуваат фокусирање на структурата и формата на објектите,

grey=cv2.cvtColor(d,cv2.COLOR\_BGR2GRAY)

• Оваа линија од кодот ја конвертира влезната слика d од BGR (Боја) во сива скала користејќи ја функцијата cv2.cvtColor. Конвертирањето на сликата во сива скала е потребно затоа што тоа ги поедноставува последователните операции за обработка на сликата.

**Примена на Гаусова замагленост**

Гаусова замагленост (Gaussian Blur) е техника во обработка на слики која се користи за намалување на шумот и изгладување на сликата. Оваа техника е основана на математичкиот модел на Гаусово распределение и е ефективна за смекчување на рабовите и деталите во сликата,

blur = cv2.GaussianBlur(grey, (5, 5), 0)

• Овде се применува Гаусова замагленост на сивата слика со големина на јадро од 5x5. Оваа операција ја намалува шумот во сликата и ги прави рабовите помазни, што ја олеснува детекцијата на објектите.

**Праговна трансформација**

Праговна трансформација (Thresholding) е важна техника во обработка на слики која се користи за претворање на грејскал слика во бинарна (црно-бела) слика. Оваа техника функционира со поставување праг (threshold) врз вредностите на пикселите, каде што сите пиксели со вредност помала од прагот се поставуваат на една вредност (на пример, црна), а сите пиксели со вредност поголема или еднаква на прагот се поставуваат на друга вредност (на пример, бела).

ret, th = cv2.threshold(blur, 20, 255, cv2.THRESH\_BINARY)

• Праговната трансформација ја конвертира замаглената слика во бинарна слика (црно-бела) користејќи праг вредност од 20. Сите пиксели со вредност над прагот ќе бидат поставени на 255 (бела), а останатите на 0 (црна). Оваа операција помага во сегментација на објектите од позадината.

**Дилатација**

Дилатација (Dilation) е основна операција во обработка на слики која се користи за зголемување на објектите во сликата. Оваа операција ги зголемува објектите на сликата преку додавање на пиксели на границите на објектите, врз основа на структурниот елемент (kernel) што се користи за дилатација.

dilated = cv2.dilate(th, np.ones((3, 3)))

**•** Дилатацијата е морфолошка операција која ја проширува белата област на бинарната слика. Ова помага да се пополнат мали дупки во детектираните објекти. Користи јадро од 3x3.

**Морфолошко затворање**

Морфолошкото затворање (Morphological Closing) е операција во обработката на слики која се користи за затворање на мали дупки во објектите или мали отвори помеѓу објектите. Оваа операција често се користи за да се исправат недостатоците од други операции како што се дилатацијата и ерозијата.

kernel=cv2.getStructuringElement(cv2.MORPH\_ELLIPSE, (2, 2))

closing=cv2.morphologyEx(dilated,cv2.MORPH\_CLOSE, kernel)

• Затворањето е морфолошка операција која комбинира дилатација и ерозија користејќи елемент со елипсовидна форма. Ова помага да се пополнат мали дупки и празнини во објектите.

**Наоѓање на контури**

Наоѓањето на контури е важна операција во обработката на слики која се користи за откривање и идентификување на затворени региони (објекти) во сликата.

contours,h=cv2.findContours(closing,cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

• Контурите се откриваат користејќи ја функцијата cv2.findContours, која ги наоѓа контурите на објектите во бинарната слика. Овие контури потоа можат да се користат за идентификација и анализа на детектираните возила.

**Детекција на контури**



**Иницијализација на празна листа за нови објекти**

new\_tracks = []

Оваа линија креира празна листа наречена new\_tracks која ќе се користи за складирање на координатите на новите патеки (центроиди) на контурите.

**Итерација преку сите контури**

for (i, c) in enumerate(contours):

Оваа линија започнува for циклус која ќе се прошета низ секоја контура од листата contours. enumerate дава и индекс i и самата контура c.

**Ги добиваме координатите и димензиите на правоаголникот што ја обвива контурата**

(x, y, w, h) = cv2.boundingRect(c)

cv2.boundingRect(c) пресметува правоаголник што ја обвива контурата c, и ги враќа координатите на горниот лев агол (x, y) и ширината w и висината h.

**Проверуваме дали контурата е валидна според минималната ширина и висина**

contour\_valid = (w >= min\_contour\_width) and (h >= min\_contour\_height)

if not contour\_valid:

continue

Овие линии проверуваат дали ширината и висината на правоаголникот ги исполнуваат минималните барања (min\_contour\_width и min\_contour\_height). Ако контурата не ги исполнува овие барања (contour\_valid е False), циклусот продолжува со следната итерација.

**Го пресметуваме центроидот на контурата**

centroid = centroids(x, y, w, h)

Оваа линија ја користи функцијата centroids за да го пресмета центроидот на правоаголникот. centroids(x, y, w, h) враќа координати на центроидот.

**Го додаваме центроидот во листата на нови објекти**

new\_tracks.append(centroid)

Оваа линија го додава центроидот на контурата во листата new\_tracks.

**Цртање правоаголник и центроид на фрејмот**

cv2.rectangle(frame1, (x, y), (x + w, y + h), (255, 0, 0), 2)

cv2.circle(frame1, centroid, 5, (0, 255, 0), -1)

cv2.rectangle црта правоаголник околу контурата на фрејмот frame1 со сина боја (BGR: (255, 0, 0)) и дебелина на линијата од 2 пиксели.

cv2.circle црта круг на позицијата на центроидот со зелена боја (BGR: (0, 255, 0)) и полупречник од 5 пиксели. Полупречникот е -1 за да се означи дека кругот е полн.



for obj\_id, track in list(tracked\_objects.items()):

Оваа линија започнува for циклус која ќе се прошета низ сите објекти во речникот tracked\_objects. list(tracked\_objects.items()) се користи за да се направи копија на речникот додека се итера преку него, бидејќи можеме да ги модифицираме објектите во текот на итерацијата.

**Иницијализација на променливата match\_found**

match\_found = False

Оваа линија ја иницијализира променливата match\_found на False. Оваа променлива ќе се користи за да се следи дали е пронајдено совпаѓање за тековниот објект.

**Прошетка низ новите објекти за да се најде совпаѓање**

for centroid in new\_tracks:

if np.linalg.norm(

np.array(track['centroid']) - np.array(centroid)) < distance\_threshold:

Овие линии започнуваат for циклус која ќе се прошета низ сите нови патеки (new\_tracks). Ако растојанието (евклидовото растојание) помеѓу тековниот центроид на следениот објект и новиот центроид е помало од зададениот праг (distance\_threshold), тоа значи дека е пронајдено совпаѓање.

**Ажурирање на следениот објект**

tracked\_objects[obj\_id]['centroid'] = centroid

tracked\_objects[obj\_id]['age'] = 0

match\_found = True

new\_tracks.remove(centroid)

break

Овие линии го ажурираат центроидот на следениот објект со новиот центроид. Исто така, староста на објектот се ресетира на 0. match\_found се поставува на True, новиот центроид се отстранува од new\_tracks, и се излегува од внатрешниот циклус (break).

**Ако не е пронајдено совпаѓање**

if not match\_found:

tracked\_objects[obj\_id]['age'] += 1

Ако не е пронајдено совпаѓање за тековниот објект (match\_found е False), age на објектот се зголемува за 1.

**Отстранување на стари објекти**

if tracked\_objects[obj\_id]['age'] > max\_age:

del tracked\_objects[obj\_id]

Ако age на објектот е поголем од зададениот максимален век (max\_age), објектот се отстранува од tracked\_objects.

**Додавање на нови објекти**

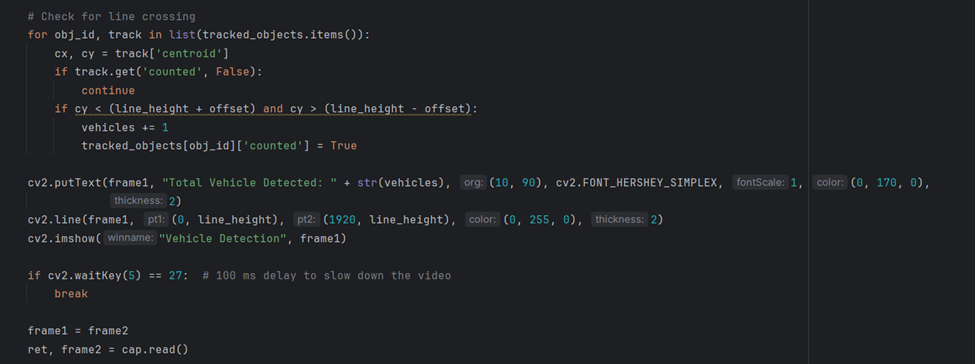
for centroid in new\_tracks:

tracked\_objects[next\_id] = {'centroid': centroid, 'age': 0}

next\_id += 1

Оваа петља ги додава сите преостанати нови патеки како нови објекти во tracked\_objects со нов уникатен идентификатор (next\_id). Новите објекти се иницијализираат со centroid и age поставен на 0. next\_id се зголемува за секој нов објект.

**Броење на возила**

****

**Итерација преку сите објекти кои се пратат**

for obj\_id, track in list(tracked\_objects.items()):

Оваа линија започнува for циклус која ќе се прошета низ сите објекти во листа tracked\_objects. list(tracked\_objects.items()) се користи за да се направи копија на листата додека се итерира преку него, бидејќи можеме да ги модифицираме објектите во текот на итерацијата.

**Ги добиваме координатите на центроидот на објектот што се прати**

cx, cy = track['centroid']

Оваа линија ги извлекува координатите cx и cy на центроидот на тековниот следен објект.

**Проверка дали објектот е веќе преброен**

if track.get('counted', False):

continue

Оваа линија проверува дали објектот веќе е преброен. Ако е (track.get('counted', False) враќа True), циклусот продолжува со следниот објект (continue).

**Проверка дали објектот ја поминал линијата за детекција за пребројување**

if cy < (line\_height + offset) and cy > (line\_height - offset):

vehicles += 1

tracked\_objects[obj\_id]['counted'] = True

Овие линии проверуваат дали координатата cy на центроидот е во рамките на одредено растојание (офсет) од висината на линијата за пребројување (line\_height). Ако објектот ја пресекол линијата, бројот на возила (vehicles) се зголемува за 1, и објектот се означува како преброен (tracked\_objects[obj\_id]['counted'] = True).

**Прикажување на вкупниот број на преброени возила на фрејмот**

cv2.putText(frame1, "Total Vehicle Detected: " + str(vehicles),(10,90),

cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 170, 0), 2)

Оваа линија додава текст на фрејмот frame1 кој го прикажува вкупниот број на преброени возила. Текстот е позициониран на координатите (10, 90), користи FONT\_HERSHEY\_SIMPLEX фонт, има големина на текстот 1, зелена боја (BGR: (0, 170, 0)), и дебелина на линијата 2.

**Цртање на линијата за пребројување на фрејмот**

cv2.line(frame1, (0, line\_height), (1920, line\_height), (0, 255, 0), 2)

Оваа линија црта хоризонтална линија за пребројување на фрејмот frame1. Линијата е позиционирана на висина line\_height, и се протега од левиот (0) до десниот крај (1920) на фрејмот. Линијата е зелена (BGR: (0, 255, 0)) и има дебелина на линијата 2.

**Прикажување на фрејмот со детекција на возила**

cv2.imshow("Vehicle Detection", frame1)

Оваа линија го прикажува фрејмот frame1 во прозорец со наслов "Vehicle Detection".

**Чекање на притискање на копче од тастатура**

if cv2.waitKey(5) == 27: # 100 ms delay to slow down the video

break

Оваа линија чека за 5 милисекунди за притискање на тастер. Ако тастерот Esc (код 27) е притиснат, петљата се прекинува и излегува од програмата. Ова исто така дава мала пауза за да се успори прикажувањето на видеото.

**Преземање на следниот фрејм од видеото**

frame1 = frame2

ret, frame2 = cap.read()

Овие линии го поставуваат тековниот фрејм frame1 на вредноста на следниот фрејм frame2. Потоа, нов фрејм се презема од видеото (cap.read()) и се складира во frame2. ret враќа True ако читањето на фрејмот е успешно.

# **Заклучок**

Проектот за детекција и броење на возила користејќи техники на дигитална обработка на слики е значаен придонес во полето на интелигентните транспортни системи. Со користење на моќта на компјутерската визија и машинското учење, овој проект демонстрира робусен и ефикасен метод за следење и анализа на сообраќајот во реално време. Успешната имплементација на овој систем покажува неколку клучни достигнувања и увиди, кои се објаснети подолу.

Прво, проектот го истакнува значењето на прецизната детекција и следење на возилата за ефективно управување со сообраќајот. Користењето на сензори како камери, LIDAR, радар и инфрацрвени сензори ја нагласува разновидноста на пристапите достапни за снимање на податоци во различни услови на животната средина. Овој мулти-сензорски пристап осигурува дека системот може да функционира ефективно во различни сценарија, од добро осветлени автопати до слабо осветлени градски улици.

Процесната единица, составена од CPU и GPU, игра клучна улога во извршувањето на комплексни алгоритми и обработка на големи количини на податоци. Интеграцијата на компјутерски визии и машинско учење овозможува системот да ги идентификува и класифицира возилата со висока прецизност. Ова се постигнува преку техники како детекција на објекти, сегментација на слики и препознавање на шаблони, кои се неопходни за разликување на различни типови возила и други објекти во кадарот.

Една од најзначајните карактеристики на овој проект е користењето на разликување на кадри, конверзија во сива скала, филтрирање и бинаризација за претпроцесирање на видео кадрите. Овие техники се од клучно значење за подобрување на квалитетот на влезните податоци и изолирање на релевантни карактеристики за понатамошна анализа. Способноста за извлекување контури и пресметка на центроиди му овозможува на системот прецизно следење на возилата низ кадрите, обезбедувајќи доверливо броење на возила.

Практичните апликации на овој проект се обемни и значајни. Во управувањето со сообраќајот, следењето во реално време може да помогне во намалување на застојот, подобрување на безбедноста на патиштата и оптимизирање на сообраќајниот проток. За управување со паркинг, системот може да обезбеди податоци во реално време за слободни паркинг места, со што се намалува времето за барање на паркинг и се подобрува вкупната ефикасност. Процесот на наплата на патарини може да се подобри преку автоматска детекција и броење на возила, минимизирајќи ја потребата за рачна интервенција и забрзувајќи го процесот на наплата.

Урбанистичкото планирање може многу да има корист од увиди обезбедени од овој систем. Анализирајќи ги сообраќајните модели и стапките на проток на возила, градските планери можат да донесуваат информирани одлуки за развој на инфраструктурата, проширување на патиштата и потребите за јавен транспорт. Овој пристап базиран на податоци осигурува дека ресурсите се распределуваат ефикасно и дека урбаниот раст се управува одржливо.

Покрај тоа, рамката на проектот е дизајнирана да биде скалабилна и адаптибилна. Ова значи дека системот може да се имплементира на различни локации и да се приспособи за да ги исполни специфичните барања. Без разлика дали се следи зафатена градска крстосница или оддалечен селски пат, флексибилноста на системот осигурува дека може да одговори на различни потреби и предизвици.

Во заклучок, овој проект не само што ја демонстрира техничката изводливост на користење на дигитална обработка на слики за детекција и броење на возила, туку и ги истакнува неговите практични придобивки и апликации. Интеграцијата на напредни алгоритми, ефикасни техники за обработка на податоци и сеопфатен сет на сензори осигурува дека системот е и прецизен и доверлив. Како што градовите продолжуваат да растат и побарувачката за интелигентни транспортни системи се зголемува, проекти како овој ќе играат клучна улога во обликувањето на иднината на урбаната мобилност и управувањето со сообраќајот. Увидите и методологиите развиени преку овој проект обезбедуваат цврста основа за понатамошни истражувања и развој во оваа област, отворајќи пат за пософистицирани и пошироко применливи системи за детекција на возила.

# **Анализа и Споредба на Различни Методи**

Во овој проект се користат различни алгоритми и техники за детекција и броење на возила. Некои од најчесто користените методи вклучуваат:

1. **Детекција на Движење**: Овој метод е основен и користи разлика помеѓу последователни видео кадри за да открие движење. Главната предност е едноставноста и брзината на обработка, но методот е чувствителен на промени во осветлувањето и може да произведе многу лажни позитиви.

* Алгоритми и Техники: Кодот користи детекција на движење преку апсолутна разлика помеѓу последователни видео кадри (cv2.absdiff). Се применуваат операции како претворање во сива скала, заматување, бинаризација и морфолошка трансформација за да се детектираат движењата.
* Предности и Недостатоци: Овој метод е едноставен и брз, но може да произведе лажни позитиви при промени во осветлувањето или сенки. Ние поставивме минимални ширина и висина за контурите за да ги избегнеме лажните позитиви.

1. **Моделирање на Позадина**: Овој метод користи техники за моделирање на позадината на сцената и откривање на објекти кои се разликуваат од позадината. Моделирањето на позадина може да биде статично или динамично, каде што се користат алгоритми како Gaussian Mixture Model (GMM) или K-nearest neighbors (KNN) за постојано ажурирање на позадината. Овој метод е покомплексен и бара повеќе ресурси, но е попрецизен и помалку чувствителен на промени во осветлувањето.

* Алгоритми и Техники: Иако во овој код не се користи комплетно моделирање на позадина, дел од морфолошките операции како затворање (cv2.morphologyEx) се користат за подобро издвојување на возилата од позадината.
* Предности и Недостатоци: Моделирањето на позадина е поефективно за статични сцени, а тука се користи дилатацијата за подобрување на контурата.

1. **Напредни Алгоритми за Машинско Учење (YOLO и Faster R-CNN)**:
   * **YOLO (You Only Look Once)**: Ова е еден од најпопуларните алгоритми за детекција на објекти. YOLO е брз и може да обработува видео во реално време. Главна предност е неговата брзина и способност да детектира повеќе објекти на различни локации во еден кадар. Меѓутоа, YOLO може да има проблеми со точноста при детекција на мали објекти.
   * **Faster R-CNN**: Ова е еден од најточните алгоритми за детекција на објекти, но е побавен од YOLO. Faster R-CNN користи регионални предложувачки мрежи за подобрување на точноста на детекција. Овој метод е погоден за апликации каде точноста е поважна од брзината.
2. **Резултати и Дискусија за Перформансите**

Резултатите од системот за детекција и броење на возила се оценуваат врз основа на неколку критериуми:

1. **Точност на Детекцијата и Броењето**:
   * Системите базирани на YOLO и Faster R-CNN покажуваат висока точност на детекција, особено во добро осветлени услови.
   * Детекцијата на движење и моделирањето на позадина може да произведат лажни позитиви, особено во променливи услови на осветлување.
2. **Време на Обработка**:
   * YOLO има значително побрзо време на обработка во споредба со Faster R-CNN, што го прави погоден за апликации во реално време.
   * Моделите за моделирање на позадина и детекција на движење се побрзи од напредните алгоритми за машинско учење, но жртвуваат точност за брзина.
3. **Способност за Работа во Различни Услови**:
   * YOLO и Faster R-CNN покажуваат подобри резултати во различни временски услови и при дневна и ноќна детекција, благодарение на нивната способност да научат сложени обрасци на објекти.
   * Детекцијата на движење и моделирањето на позадина се поефикасни во статични услови, но се помалку ефективни во различни временски услови и при ноќна детекција.

Анализата и споредбата на овие методи покажуваат дека изборот на алгоритам зависи од специфичните потреби на апликацијата, како што се потребната точност, брзина и услови на работа. Комбинацијата на различни методи може да понуди баланс помеѓу точност и брзина, обезбедувајќи оптимални резултати за детекција и броење на возила.