

UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IAȘI  
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE  
DOMENIUL: Ingineria Sistemelor  
SPECIALIZAREA: Automatică și Informatică Aplicată

## LUCRARE DE DIPLOMĂ

**Absolvent:**  
**Teodor-Florin BUTNARU**

**Coordonator științific:**  
**ș.l. dr. ing. Alexandru-Tudor POPOVICI**

**Iași, 2024**



UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IAȘI  
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE  
DOMENIUL: Ingineria Sistemelor  
SPECIALIZAREA: Automatică și Informatică Aplicată

# **Proiectarea și Testarea unui Sistem de Conducere Autonomă pentru Competiții Studentești**

LUCRARE DE DIPLOMĂ

**Coordonator științific:  
ș.l. dr. ing. Alexandru-Tudor POPOVICI**

**Absolvent:  
Teodor-Florin BUTNARU**

**Iași, 2024**

## **DECLARAȚIE DE ASUMARE A AUTENTICITĂȚII PROIECTULUI DE DIPLOMĂ**

Subsemnatul BUTNARU TEODOR-FLORIN,  
legitimat cu Ci seria MZ nr. 874992, CNP 5020323226879  
autorul lucrării PROIECTAREA ȘI TESTAREA UNUI SISTEM  
DE CONDUCERE AUTONOMĂ PENTRU COMPETIȚII STUDENȚESTI

elaborată în vederea susținerii examenului de finalizare a studiilor de licență, programul de studii AUTOMATICĂ ȘI INFORMATICĂ APLICATĂ organizat de către Facultatea de Automatică și Calculatoare din cadrul Universității Tehnice „Gheorghe Asachi” din Iași, sesiunea IULIE a anului universitar 2024, luând în considerare conținutul Art. 34 din Codul de etică universitară al Universității Tehnice „Gheorghe Asachi” din Iași (Manualul Procedurilor, UTI.POM.02 - Funcționarea Comisiei de etică universitară), declar pe proprie răspundere, că această lucrare este rezultatul propriei activități intelectuale, nu conține porțiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislației române (legea 8/1996) și a convențiilor internaționale privind drepturile de autor.

Data

01.07.2024

Semnătura

Butnaru



# Cuprins

<b>Introducere</b>	<b>1</b>
<b>1 Fundamentarea teoretică și documentarea bibliografică</b>	<b>3</b>
1.1 Tema . . . . .	3
1.2 Istoria mașinii fără șofer . . . . .	3
1.3 iDEAS Engineering Competition . . . . .	5
1.4 Tehnologia de ultimă oră . . . . .	6
1.4.1 Sisteme avansate de asistență la conducere . . . . .	6
1.4.1.1 Lane Assist . . . . .	7
1.4.1.2 Adaptive Cruise Control . . . . .	7
1.4.1.3 Automated Emergency Braking . . . . .	8
1.4.1.4 Parking Assistance . . . . .	8
1.4.2 Sisteme de conducere autonomă . . . . .	8
1.4.3 Sisteme end to end de conducere . . . . .	9
1.4.4 Sisteme modulare de conducere . . . . .	11
<b>2 Proiectarea sistemului</b>	<b>15</b>
2.1 Hardware . . . . .	15
2.2 Simulatorul . . . . .	16
2.3 Arhitectura software . . . . .	17
2.3.1 Tehnologii utilizate și modulele principale . . . . .	19
2.3.1.1 Modulul de percepție . . . . .	19
2.3.1.2 Modulul de planificare . . . . .	22
2.3.1.3 Modulul de control . . . . .	22
2.3.2 Antrenarea retelelor neuronale . . . . .	23
2.3.2.1 YOLOv3 . . . . .	23
2.3.2.2 FastSCNN . . . . .	25
<b>3 Implementarea aplicației</b>	<b>31</b>
<b>4 Testarea aplicației și rezultate experimentale</b>	<b>41</b>
4.1 Elemente de configurare și instalare . . . . .	41
4.1.1 Laptop . . . . .	41
4.1.1.1 MediaMTX . . . . .	41
4.1.1.2 OBS . . . . .	41
4.1.1.3 Unity . . . . .	42
4.1.2 Jetson . . . . .	42
4.2 Performanța hardware și utilizarea resurselor sistemului . . . . .	43
4.2.1 Utilizare resurse . . . . .	43
4.2.2 Analiza latentei . . . . .	43
4.3 Testarea sistemului . . . . .	45

4.3.1	Metodologie . . . . .	45
4.3.2	Rezultate . . . . .	45
<b>Concluzii</b>		<b>47</b>
<b>Bibliografie</b>		<b>49</b>
<b>Anexe</b>		<b>53</b>
1	Clasa Perception . . . . .	53
2	Clasa Planning . . . . .	55
3	Algoritm mapare . . . . .	57
4	Functii initializare retele . . . . .	58
5	Utilizare GPU si CPU . . . . .	64

# Proiectarea și Testarea unui Sistem de Conducere Autonomă pentru Competiții Studențești

Teodor-Florin BUTNARU

## Rezumat

Această lucrare explorează dezvoltarea unui vehicul autonom pentru competiția iDEAS Engineering Competition. Sistemul propus utilizează tehnici avansate de învățare automată și procesare a imaginilor pentru navigarea autonomă pe o pistă, respectând cerințele impuse de competiție.

Lucrarea include o analiză teoretică a tehnologiilor moderne în conducerea autonomă, cum ar fi detectarea obiectelor, segmentarea semantică și planificarea traectoriei. Sunt discutate atât arhitecturile modulare, cât și cele end-to-end, evidențiind avantajele și dezavantajele fiecărei abordări.

Implementarea practică utilizează placa Jetson Nano, demonstrând capacitatea acesteia de a rula simultan două rețele neuronale cu o performanță de 10 cadre pe secundă.

Testele efectuate au confirmat că vehiculul poate respecta majoritatea cerințelor competiției, cum ar fi oprirea la semnele de circulație, evitarea obstacolelor și efectuarea manevrei de parcare. Rezultatele subliniază eficiența și fiabilitatea sistemului dezvoltat, oferind direcții pentru îmbunătățiri viitoare, cum ar fi testarea în condiții reale și explorarea platformelor Jetson mai avansate.

Contribuțiile personale includ dezvoltarea unui proces optimizat pentru calibrarea camerei și transformarea perspectivei. Lucrarea se încheie cu lecții învățate pe parcursul proiectului și sugestii pentru dezvoltări viitoare.



## Introducere

În elaborarea acestui document, a fost utilizata platforma ChatGPT pentru a clarifica și structura ideile cât mai concis și coerent. Este important de menționat că ChatGPT a fost folosit exclusiv pentru reformularea textului, în timp ce ideile și concepțele originale provin din creativitatea și cercetarea autorului.

În ultimii ani, conducerea autonomă a devenit o direcție de cercetare în creștere, cu aplicații practice în domeniul transporturilor. Această tehnologie are ca scop crearea de sisteme de transport care să poată funcționa fără intervenția umană, cu scopul de a crește siguranța și eficiența transportului. Sistemele autonome de conducere sunt bazate pe tehnologii avansate de procesare a imaginilor și de recunoaștere a obiectelor, utilizând metode bazate pe inteligență artificială. Implementarea unei astfel de tehnologii într-un vehicul autonom reprezintă un pas important spre un viitor în care transportul să fie mai sigur, mai eficient și mai accesibil pentru toată lumea.

Conducerea autonomă a devenit un subiect de discuție central în industria automobilelor, reflectând o combinație complexă de provocări tehnice, de siguranță și etice. Unul dintre scopurile acestei tehnologii este de a minimiza rata accidentelor rutiere datorată erorii umane. Conform studiului [1], din 2189000 de accidente 94% au fost datorate șoferului, din motive precum erori de decizie, aptitudine de conducere sau neatenție. Pe lângă reducerea accidentelor, se așteaptă ca vehiculele autonome să reducă emisiile de carbon și să crească productivitatea economică datorită fluidizării traficului, rutării eficiente și serviciilor de tip ride-sharing [2].

Motivația pentru implementarea unei mașini autonome a început încă din anul doi de facultate, când am participat pentru prima oară la concursul iDEAS Engineering Competition[3]. Această experiență a fost punctul de plecare și a stârnit un interes profund pentru acest domeniu. Pe măsură ce am continuat să participe la competiții, motivația mea a crescut. Participarea la competiția NXP Cup[4], unde am avut ocazia să călătoresc la București pentru a participa în finală alături de multe alte echipe din toată Europa și să colaborez cu colegi, a consolidat și mai mult această pasiune. De asemenea, participarea din octombrie 2023 la concursul iDEAS a fost momentul în care am început să conturez mai concret proiectul de licență.

Pasiunea mea pentru mașini, combinată cu interesul pentru inteligență artificială, a jucat un rol crucial în alegerea acestui proiect de licență. Dezvoltarea unui sistem autonom de conducere nu este doar o provocare tehnică, ci și o oportunitate de a contribui la un domeniu aflat în plină expansiune.

Obiectivele acestei lucrări de licență sunt de a projecția și implementa o mașină autonomă capabilă să îndeplinească cerințele competiției iDEAS, navigând pe pista competiției și respectând toate constrângerile și regulile impuse. Pasiunea mea pentru acest domeniu și experiențele din participările anterioare la concursuri m-au motivat să dezvolt un sistem complet funcțional și eficient de conducere autonomă. Acest proiect urmărește să depășească limitele întâlnite în participările anterioare și să creeze un vehicul autonom capabil să funcționeze optim în cadrul competiției, demonstrând astfel progresele și cunoștințele acumulate.

Conducerea autonomă implică înțelegerea și interpretarea mediului înconjurător, sarcină simplă pentru o persoană datorită capabilității inerente ale omului de recunoaștere a formelor și de raționament, însă pentru un calculator replicarea acestor capabilități umane cu o acuratețe ridicată este o sarcină complexă.

Această complexitate face ca abordarea acestei probleme să fie posibilă numai prin intermediul sistemelor avansate de inteligență artificială și învățare automată. Aceste tehnologii oferă capacitatea de a procesa și interpreta cantități masive de date de la senzori vehiculului, precum camere și radar, permitând vehiculelor să navigheze în siguranță și să ia decizii în timp real într-un mediu în continuă schimbare.

În cadrul acestui proiect, s-au utilizat diverse metode și instrumente pentru a atinge obiectivele propuse. Două rețele neuronale au fost antrenate și rulate în paralel pe o placă NVIDIA Jetson Nano. Procesul de antrenare a implicat utilizarea soluțiilor online, cum ar fi Google Colaboratory[5], pentru accesul la GPU-uri puternice, precum și laptopul personal pentru antrenarea și validarea continuă a modelelor, utilizând diverse librării de învățare automată.

Pe lângă antrenarea rețelelor neuronale, proiectul a implicat utilizarea mediului de simulare Unity[6] pentru a recrea condițiile concursului iDEAS. Acest mediu de simulare a permis testarea și validarea sistemului autonom într-un cadru controlat, similar cu cel al competiției. Astfel, s-a putut analiza comportamentul vehiculului în diferite scenarii și condiții, optimizând algoritmii și strategiile folosite.

Tehnici avansate, utilizate și în industrie, au fost implementate pentru a îmbunătăți performanța sistemului. Una dintre aceste tehnici este perspectiva vederii de sus, *bird's eye view*, care facilitează planificarea traectoriei vehiculului. Această abordare permite o mai bună interpretare a mediului înconjurător și o planificare mai eficientă a traseului.

Sistemul autonom dezvoltat este modular, similar celor din industria actuală. Aceasta înseamnă că fiecare componentă a sistemului, de la percepție la planificare și control, funcționează independent și poate fi actualizată sau îmbunătățită fără a afecta restul sistemului. Modulul de percepție interpretează datele senzorilor, modulul de planificare generează traectoria optimă, iar modulul de control aplică comenziile necesare pentru a urmări această traectorie.

Pe partea de automatizare, a fost implementat un algoritm PID pentru controlul direcției vehiculului. Acesta permite un control precis și stabil al mișcării vehiculului, asigurând o navigare fluidă și sigură.

Această combinație de metode și instrumente, împreună cu abordarea modulară și utilizarea tehnicii avansate, a permis dezvoltarea unui sistem robust și performant de conducere autonomă, capabil să îndeplinească cerințele competiției iDEAS și să demonstreze progresele realizate în cadrul acestui proiect de licență.

În continuare, lucrarea este structurată astfel. Capitolul 1 acoperă fundamentarea teoretică și documentarea bibliografică, inclusiv istoria mașinilor autonome, detalii despre concursul iDEAS și tehnologiile moderne în domeniul conducerii autonome. Capitolul 2 detaliază proiectarea sistemului, prezintând arhitectura hardware și software, mediul de simulare și procesul de antrenare a rețelelor neuronale. Capitolul 3 discută implementarea practică a aplicației, iar Capitolul 4 prezintă metodele de testare aplicate, oferind o analiză a performanțelor sistemului. În final, lucrarea se încheie cu concluziile și perspectivele de viitor, evaluând realizările proiectului și lecțiile învățate.

## Capitolul 1. Fundamentarea teoretică și documentarea bibliografică

### 1.1. Tema

Tema propusă pentru această lucrare de licență este proiectarea și implementarea unei mașini autonome capabile să participe la competiția iDEAS Engineering Competition. Această temă implică dezvoltarea unui sistem autonom de conducere pentru o mașină în miniatură, capabil să navigheze în mod autonom pe o pistă, respectând regulile de trafic și interacționând cu diverse semne de circulație. Această competiție oferă un mediu excelent pentru experimentarea și punerea în practică a unor tehnici de ultimă oră utilizate în domeniul conducerii autonome.

Participarea la competiții precum iDEAS și NXP Cup a oferit o perspectivă valoroasă asupra dificultăților și oportunităților din acest domeniu, alimentând dorința de a contribui la avansarea tehnologiilor de conducere autonomă. Această temă oferă o oportunitate practică de a aplica și testa tehnologiile de conducere autonomă, contribuind la înțelegerea detaliată a modului în care funcționează o mașină autonomă la un nivel mai mic.

Implementarea acestui proiect de licență va oferi o experiență practică semnificativă, contribuind la dezvoltarea competențelor în domenii precum inteligența artificială, procesarea imaginilor și controlul automat. Acest proiect este deosebit de important, oferind o oportunitate de a aplica cunoștințele acumulate și de a explora un domeniu în plină dezvoltare.

### 1.2. Istoria mașinii fără șofer

Conceptul de mașină fără șofer își are originile în anii '20. Prima astfel de mașină, denumită *American Wonder*, era un autovehicul controlat prin unde radio, supranumit *mașina fantomă*. Aceasta era operată de o persoană aflată într-un alt vehicul în urma acesteia.

Abia după anii 2000, tehnologia a ajuns la un punct în care acest concept să poată fi pus în practică. În 2004, DARPA (Defense Advanced Research Projects Agency) a organizat prima competiție de distanță mare pentru vehicule autonome în desertul Mojave din Statele Unite ale Americii. Aceasta a propus un traseu de 240 km, din Barstow, California, până în Primm, Nevada. Din cele 15 echipe finaliste, niciuna nu a reușit să revendice premiul de un milion de dolari. Scopul acestei competiții a fost accelerarea dezvoltării tehnologiei vehiculelor autonome. Deși niciun competitor nu a reușit să finalizeze traseul, în ediția din anul următor, cinci echipe au completat cu succes cursa, care a avut loc în desertul din sud-vest, în apropierea graniței dintre California și Nevada.

În 2007, DARPA a organizat ultima competiție destinată vehiculelor autonome, numită *Urban Challenge*. Vehiculele au concursat pe un traseu urban de 96 km, respectând regulile de circulație și navigând alături de alte vehicule autonome și vehicule conduse de oameni. Șase echipe au reușit să completeze traseul, demonstrând astfel progresele semnificative făcute în domeniu.

În 2009, Sebastian Thrun, coordonatorul echipei The Stanford Racing Team, care a câștigat competiția *Grand Challenge* din 2005 și s-a clasat pe locul doi în competiția din 2007, a înființat proiectul *Google Self-Driving Car Project*, cunoscut astăzi sub numele de *Waymo LLC*. Astăzi, Waymo se află printre cele mai avansate vehicule autonome, oferind servicii de călătorie fără șofer în orașele Phoenix și San Francisco.

De la primele competiții organizate de DARPA, tehnologia vehiculelor autonome a avansat semnificativ, atingând noi niveluri de performanță și fiabilitate. Un moment notabil în evoluția acestei tehnologii a fost lansarea primei mașini de nivel 3 SAE de către Mercedes-Benz. Aceasta poate opera autonom fără intervenția șoferului, deși în anumite condiții de mediu și de infrastructură.

În paralel, diverse companii de tehnologie și automobile au făcut progrese remarcabile în

dezvoltarea vehiculelor autonome. Tesla, de exemplu, a introdus sistemul său de asistență avansată pentru șofer, cunoscut sub numele de Autopilot, care oferă capabilități de conducere semi-autonomă, nivel 2 SAE, pe drumuri publice. Această tehnologie a fost îmbunătățită constant prin actualizări software, aducând vehiculele Tesla mai aproape de nivelurile superioare de autonomie.

Waymo, derivată din proiectul *Google Self-Driving Car*, a continuat să inoveze în domeniu, oferind servicii de ride-hailing autonome în orașele Phoenix și San Francisco. Aceștia utilizează vehicule de nivel 4 SAE care sunt capabile să navigheze în mod autonom în zone urbane complexe, demonstrând capacitatea de a opera fără intervenție umană în medii variate.

În 2020, Nuro a devenit prima companie care a primit aprobarea de a opera vehicule complet autonome pentru livrări pe drumurile publice. Vehiculele Nuro sunt concepute pentru a livra produse alimentare și alte bunuri, operând în principal în suburbii și zone rezidențiale.

De asemenea Starship Technologies a lansat roboti de livrare autonomi, care operează pe trotuar pentru a livra alimente și alte produse în campusuri universitare și cartiere urbane. Acești roboti sunt capabili să navigheze autonom și să evite obstacole, oferind un serviciu de livrare eficient și sigur.

Aceste realizări subliniază progresul rapid în tehnologia vehiculelor autonome și arată cum diferite companii contribuie la dezvoltarea acestui domeniu.

Nivelurile SAE de autonomie a vehiculelor, stabilite de Society of Automotive Engineers (SAE), sunt o clasificare standardizată a capacitaților de conducere autonomă, variind de la nivelul 0, fără automatizare, până la nivelul 5 automatizare completă. Fiecare nivel reprezintă un grad diferit de automatizare, de la asistență simplă a șoferului până la operarea complet autonomă în toate condițiile de drum și de mediu. Aceste nivele sunt esențiale pentru înțelegerea progresului în tehnologia vehiculelor autonome și pentru evaluarea capacitaților diferitelor sisteme dezvoltate de industrie. Nivelurile SAE de automatizare pot fi văzute în figura 1.1.

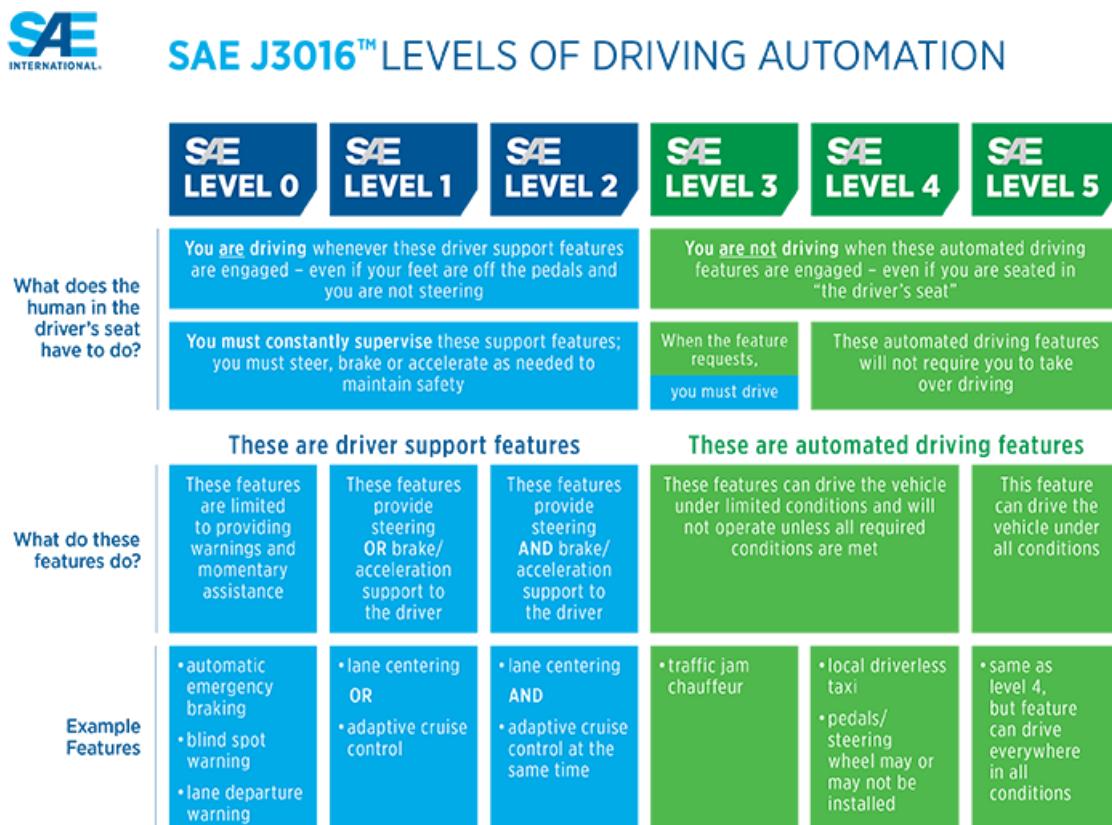


Figura 1.1. Niveluri SAE de automatizare.<sup>1</sup>

<sup>1</sup> imagine preluată de pe site-ul oficial SAE[7]

### 1.3. iIDEAS Engineering Competition

iIDEAS Engineering Competition[3] este un concurs adresat studenților ce presupune proiectarea unui prototip de mașină capabil să navigheze o pistă în mod autonom fără intervenție umană. Cunoscut în trecut sub numele de *Electro-Mobility* această competiție a suferit diverse modificări de-a lungul anilor, cum ar fi adăugarea semafoarelor sau a stației de încărcare unde mașina trebuie să se parcheze singură.

Competiția propune o pistă în forma de opt, de aproximativ 12 metri în lungime și 5 metri în lățime, aceasta conține două benzi de circulație în ambele sensuri. Pe parcursul acesteia sunt amplasate șapte semne de circulație, două semafoare și o stație de încărcare, poziția acestora poate fi vizualizată în figura numărul 1.3.

Mașinuța trebuie să fie capabilă să recunoască patru semne distincte de circulație, figura numărul 1.2, și toate cele trei stări ale unui semafor.



(a) PC1 - Pedestrian Crossing



(b) SS2 - Stop Sign



(c) PS3 - Parking Sign



(d) CS4 - Charging Station

Figura 1.2. Cele patru semne de circulație.

Traseul pe care mașina trebuie să îl parcurgă reprezintă o tura a pistei, linia de start este marcată pe pistă prin săgeata îndreptată spre dreapta, cea de lângă semnul PS3, iar linia de stop reprezintă zona de 0.95 metri după semnul PS3. Mașina trebuie să implementeze protocoale pentru fiecare dintre cele patru semne de circulație și cele trei stări ale semaforului.

În urma detecției semnelor PC1 și SS2, mașina se va opri timp de o secundă, respectiv trei secunde la o distanță mai mică de 0.95 metri față de semn.

În ceea ce privește semnul PS3, acesta desemnează finalul traseului, iar în urma detecției acestuia mașina va opri cu toate rotile la o distanță mai mică de 0.95 metri după acesta.

La momentul detecției semnului CS4 mașina va efectua manevra de parcare pe suportul de încărcare wireless, marcat cu verde, indicând acest lucru prin aprinderea unui led timp de cinci secunde.

În ceea ce privește semaforul mașina va trece doar când acesta este verde, iar în caz contrar va aștepta până când se face verde. Regulamentul prevede două astfel de semafoare însă doar unul din ele se află pe traiectoria ce trebuie urmată de mașină.

Pe lângă toate acestea, mașina trebuie să fie capabilă să detecteze un obstacol plasat aleator pe sensul de drum și să efectueze manevra de depășire. Deși regulamentul prevede poziționarea aleatorie a obstacolului, în realitate acesta nu poate fi plasat în locuri care ar împiedica detectarea corectă a semnelor de circulație, cum ar fi în fața unui semn.

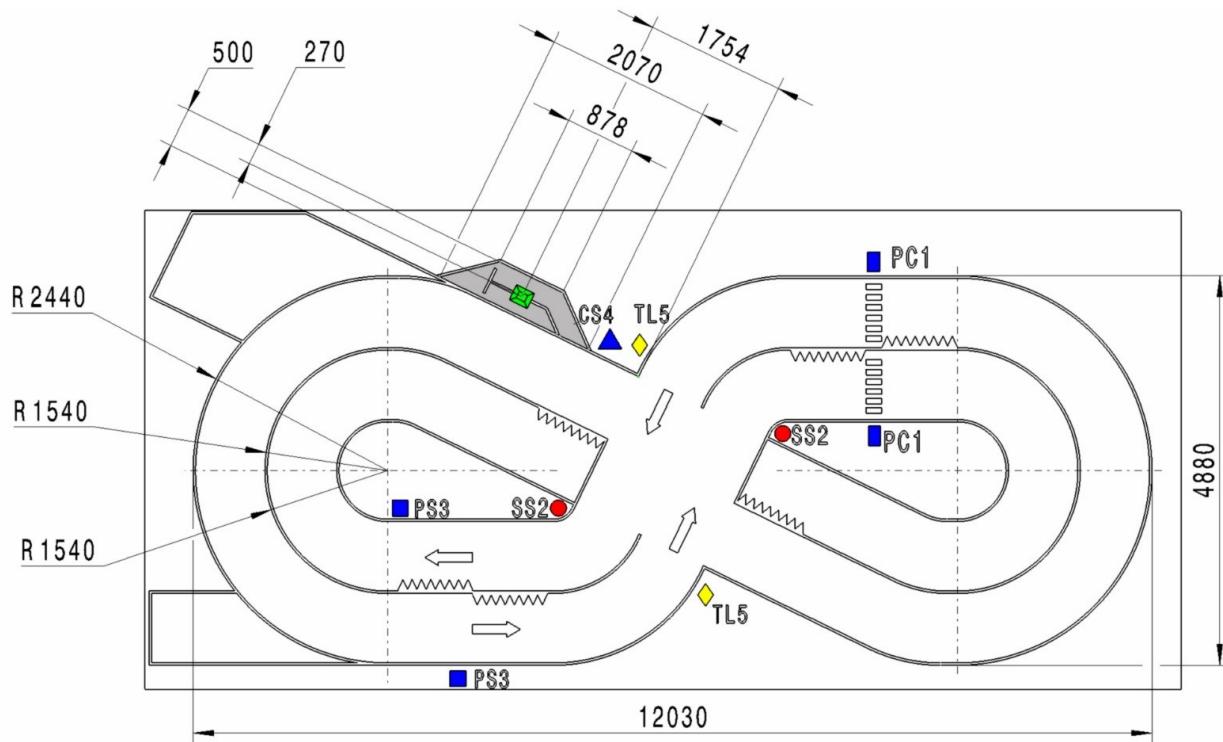


Figura 1.3. Dimensiuni pista și locația semnelor și semafoarelor de circulație.<sup>2</sup>

Astfel, se așteaptă ca mașinuță să fie capabilă să parcurgă o tură completă a traseului, respectând toate regulile de trafic. Aceasta trebuie să oprească la semnele care impun oprirea, să respecte semaforul, să realizeze corect manevra de parcare și să depășească obstacolul atunci când este prezent. Pentru a îndeplini aceste cerințe, mașinuță va utiliza o serie de algoritmi avansați de detectare a benzii, obstacolului, semaforului, semnelor și a stației de încărcare. Aceste funcționalități combinate vor permite vehiculului autonom să navigheze în mod eficient și sigur pe traseu, demonstrând capacitatele sale de conducere autonomă.

#### 1.4. Tehnologia de ultimă oră

În domeniul conducerii autonome, tehnologia a evoluat semnificativ în ultimii ani. Pe măsură ce progresul continuă, devine crucială înțelegerea tehnicilor și metodologiilor curente utilizate în acest domeniu. Această secțiune oferă o privire de ansamblu asupra celor mai recente progrese în conducerea autonomă, incluzând utilizarea algoritmilor de învățare automată, fuziunea senzorilor și sisteme avansate de control.

Una dintre cele mai semnificative progrese în conducerea autonomă este integrarea algoritmilor de învățare automată. Acești algoritmi sunt antrenați pe cantități mari de date pentru a recunoaște tipare și a face predicții precise despre mediu. Prin valorificarea învățării automate, vehiculele autonome se pot adapta condițiilor în schimbare, datorită unei înțelegeri mai profunde a mediului înconjurător.

##### 1.4.1. Sisteme avansate de asistență la conducere

În prezent, tehnologiile de asistență în conducerea vehiculelor au evoluat semnificativ, oferind diferite niveluri de automatizare care îmbunătățesc siguranța și confortul șoferilor. De la sisteme de asistență de conducere de bază, precum menținerea benzii de circulație (Lane Assist) și cruise control adaptiv, până la soluții avansate de conducere autonomă dezvoltate de companii precum Waymo și Tesla, aceste tehnologii reprezintă pași importanți către realizarea vehiculelor

<sup>2</sup>Imagine preluată din regulamentul tehnic de pe site-ul oficial al competiției [3]

complet autonome.

Sistemele avansate de asistență pentru șofer (ADAS) includ un ansamblu de tehnologii care îmbunătățesc siguranța și confortul la volan.

#### 1.4.1.1. Lane Assist

Lane Assist este un sistem de asistență care ajută șoferul să mențină vehiculul în interiorul benzii de circulație. Acest sistem utilizează camere frontale și senzori pentru a detecta marcajele rutiere. Atunci când vehiculul se apropiște de marginea benzii fără a semnaliza, sistemul poate oferi avertismente sonore, vizuale sau tactile pentru a alerta șoferul. Dacă șoferul nu reacționează, sistemul poate prelua controlul volanului pentru a aduce vehiculul înapoi pe bandă, ajustând direcția vehiculului pentru a preveni ieșirea de pe bandă.

Lane Assist funcționează eficient în anumite condiții. Este esențial ca marcajele rutiere să fie clar vizibile, în caz contrar, sistemul poate să nu detecteze corect banda de circulație. De asemenea, funcționează optim în condiții de luminozitate adecvată și la viteze de autostradă sau drumuri principale. Condițiile meteorologice nefavorabile, cum ar fi ploaia torențială, ceața densă sau ninsoarea, pot afecta performanța sistemului, deoarece senzorii și camerele nu pot capta clar marcajele rutiere.

Lane Assist are diverse implementări, printre care Lane Departure Warning (LDW) și Lane Keeping Assist (LKA). LDW este un sistem pasiv care doar avertizează șoferul că vehiculul părăsește banda, în timp ce LKA este un sistem activ care corectează direcția vehiculului pentru a-l menține în bandă. Unele sisteme avansate, cum ar fi Lane Centering Assist (LCA), pot chiar să mențină vehiculul centrat pe bandă pe tot parcursul traseului, inclusiv în curbe ușoare.

Sistemele de asistență la păstrarea benzii joacă un rol esențial în îmbunătățirea siguranței rutiere, prevenind coliziunile laterale și ieșirile neintenționate de pe drum. Conform unui studiu realizat de Insurance Institute for Highway Safety (IIHS), sistemele de avertizare la părăsirea benzii (LDW) pot reduce incidența coliziunilor unilaterale, laterale și frontale cu 11%. De asemenea, aceste sisteme scad cu 21% rata accidentelor soldate cu răniri, în cadrul acelorași tipuri de coliziuni. Astfel, tehnologiile lane assist contribuie semnificativ la creșterea siguranței tuturor participanților la trafic, prin reducerea riscului de accidente și minimizarea consecințelor acestora.

#### 1.4.1.2. Adaptive Cruise Control

Adaptive Cruise Control (ACC) este un sistem avansat de control al vitezei care ajustează automat viteză vehiculului pentru a menține o distanță sigură față de vehiculul din față. Utilizând senzori radar și camere, ACC poate accelera și frâna vehiculul în funcție de condițiile de trafic, oferind o experiență de conducere mai relaxată și mai sigură.

ACC funcționează prin utilizarea senzorilor radar și a camerelor montate pe vehicul pentru a monitoriza traficul din față. Sistemul ajustează automat viteza vehiculului pentru a menține o distanță prestabilită față de vehiculul precedent. Dacă traficul încetinește, ACC va reduce viteza vehiculului, iar dacă traficul se degajează, sistemul va accelera pentru a reveni la viteza impusă de trafic. În condiții de trafic aglomerat, ACC poate opri complet vehiculul și reporni automat atunci când traficul se mișcă din nou.

Conform unui studiu realizat de National Highway Traffic Safety Administration (NHTSA), ACC poate reduce coliziunile din spate cu până la 27%. Sistemul este disponibil pe o gamă largă de vehicule moderne, inclusiv modele de la producători precum Audi, BMW, Mercedes-Benz, Tesla, și Toyota. De exemplu, Toyota Safety Sense integrează ACC în pachetele sale de siguranță pentru a oferi protecție suplimentară.

#### 1.4.1.3. Automated Emergency Braking

Automated Emergency Braking (AEB) este un sistem de siguranță care detectează obstacolele din fața vehiculului și aplică frânele automat pentru a preveni sau reduce impactul unei coliziuni. AEB poate detecta pietoni, vehicule și alte obstacole, reaționând mai rapid decât șoferul în situații de urgență.

AEB a demonstrat o eficiență semnificativă în reducerea accidentelor și salvarea de vieți, conform studiilor realizate de IIHS. Un studiu arată că prevenirea coliziunilor frontale reduce cu 50% accidentele din spate raportate la poliție, și cu 56% accidentele care implică răniți[8]. De asemenea, AEB a dovedit o reducere de 27% a accidentelor cu pietoni în timpul zilei[9].

#### 1.4.1.4. Parking Assistance

Sistemele de asistență la parcare utilizează senzori și camere pentru a ajuta șoferul să parcheze vehiculul în locuri înguste sau dificile. Aceste sisteme pot include funcții automate de parcare, unde vehiculul preia controlul vehiculului. În mod specific, vehiculul poate detecta spațiul de parcare disponibil și poate calcula manevrele necesare pentru a se parca. În timpul acestei manevre, sistemul poate controla volanul, acceleratia și frâna, asigurându-se că vehiculul se poziționează corect și în siguranță.

Pe lângă funcțiile de parcare automată, există și alte sisteme de asistență care pot simplifica procesul de parcare. De exemplu, unele vehicule sunt echipate cu sisteme de frânare automată în cazul în care detectează un obstacol în timpul manevrelor de parcare. Aceste sisteme folosesc senzori pentru a monitoriza mediul înconjurător și pot aplica frânele automat pentru a preveni coliziunile.

Alte caracteristici avansate includ afișajul cu vedere de sus, bird's eye view, care combină imagini de la mai multe camere pentru a oferi șoferului o vedere panoramică a vehiculului și a împrejurimilor sale. Aceste tehnologii contribuie la o parcare mai sigură și mai eficientă, reducând riscul de accidente și stresul asociat cu manevrele de parcare în spații restrânse.

### 1.4.2. Sisteme de conducere autonomă

Niciunul dintre sistemele avansate prezentate anterior nu se apropie cu adevărat de conducerea autonomă. Aceste sisteme sunt, după cum sugerează și numele, doar funcționalități de asistență pentru șofer. Diferența dintre aceste funcționalități și conducerea autonomă adevărată este semnificativă.

Conducerea autonomă completă presupune ca vehiculul să fie capabil să navigheze și să ia decizii în mod independent, fără intervenția umană. Acest lucru implică o combinație complexă de tehnologii avansate, inclusiv percepție, planificare și control. În continuare, vor fi explorate tehnicele și tehnologiile utilizate în conducerea autonomă de astăzi.

Primul pas către autonomia vehiculului este capacitatea de a percepse mediul înconjurător. Această percepție este esențială pentru poziționarea vehiculului și detectarea obstacolelor. Există o gamă variată de senzori care pot fi folosiți în acest scop. Aceștia pot fi categorizați în senzori proprioceptivi și senzori exteroceptivi[10].

Senzorii proprioceptivi măsoară stările interne ale vehiculului. De exemplu, unitățile de măsurare inertială (IMU) monitorizează acceleratiile și rotațiile vehiculului, iar codurile de odometrie măsoară distanțele parcuse și vitezele roților, oferind date precise despre poziționarea vehiculului în spațiu. GPS-ul (Global Positioning System) furnizează coordonate precise pentru a determina poziția vehiculului în orice moment.

Senzorii exteroceptivi percep mediul înconjurător. LIDAR (Light Detection and Ranging) utilizează laser pentru a măsura distanțele și a crea hărți 3D detaliante ale mediului. Camerele capturează imagini vizuale, utilizate pentru detectarea obiectelor și a marcapelor rutiere, în timp ce radarul folosește unde radio pentru a detecta obiecte și a măsura distanțele și vitezele acestora.

Unul dintre provocările dezvoltării unui vehicul autonom este capacitatea de a percepă cu exactitate mediul înconjurător. În acest context, fuziunea senzorilor joacă un rol esențial. Aceasta implică combinarea datelor de la mai mulți senzori, precum camere, LIDAR și radar, pentru a oferi o percepție mai precisă și mai fiabilă a mediului. Vehiculul este echipat cu o gamă variată de senzori care furnizează informații specifice despre împrejurimi. Prin fuzionarea acestor informații, se obține o ieșire de calitate superioară, permitând vehiculului să ia decizii mai precise și să navigheze în siguranță.

Fuziunea senzorilor nu doar recreează mediul înconjurător într-o reprezentare simplificată, ci oferă și multiple avantaje. În cazul defectării unui senzor, ceilalți senzori pot compensa, menținând o reprezentare precisă. De asemenea, această tehnică atenuează limitările fiecărui senzor individual, asigurând o percepție mai robustă și mai fiabilă. Totuși, fuziunea senzorilor nu este o soluție completă, deoarece senzorii nu sunt perfecti și introduc zgomot și incertitudine. Mai mult de atât, utilizarea mai multor senzori crește complexitatea algoritmilor și timpul de procesare, necesitând hardware mai scump pentru a combate latența crescută, esențială într-un mediu dinamic.

Cel mai avansat sistem autonom de astăzi este Waymo, aflat la a cincea generație. Acesta se bazează pe fuziunea datelor de la multiple camere LIDAR-uri, camere și radare. Sistemul include un LIDAR cu unghi de vizualizare de 360 de grade, o cameră, de asemenea cu un unghi de 360 de grade, capabil să detecteze obiecte la peste 500 de metri distanță, și mai multe LIDAR-uri pentru distanțe mici, destinate obiectelor apropiate. Vehiculul este echipat cu camere amplasate strategic pentru un câmp vizual suprapus și radare pentru măsurarea vitezei și detectarea obiectelor în mișcare[11].

Având înțelesе conceptele despre cum percepă un vehicul mediul înconjurător și cu ce tipuri de date operează, ne îndreptăm acum spre analiza mai detaliată a arhitecturii software a unei astfel de mașini și a modului de funcționare internă a acesteia.

Articolul [12] prezintă practici comune în dezvoltarea sistemelor de conducere autonomă. Yurtsever et al. clasifică aceste sisteme după două criterii, conectivitate și designul arhitecturii. Din punct de vedere al conectivității, sistemele pot fi ego sau conectate.

În cadrul unui sistem ego, toate funcțiile de conducere autonomă sunt integrate într-un singur vehicul autosuficient. Această abordare este cea mai comună datorită costurilor relativ scăzute și a simplității implementării. Vehiculul colectează, procesează și interpretează datele de la senzorii proprii pentru a lua decizii în timp real.

În cadrul unui sistem conectat, vehiculele autonome comunică între ele, cu infrastructura sau alte dispozitive inteligente. Acest tip de sistem folosește tehnologii precum V2X (Vehicle-to-Everything), V2V (Vehicle-to-Vehicle), V2I (Vehicle-to-Infrastructure) și V2P (Vehicle-to-Pedestrian). Aceste tehnologii permit vehiculelor să împărtășească informații despre trafic, obstacole și condițiile drumului, îmbunătățind siguranța și eficiența traficului. Deși promițătoare, această tehnologie este încă în stadiul de concept și necesită o infrastructură extinsă pentru a fi implementată pe scară largă. Mai multe detalii despre perspectivele viitoare și o privire de ansamblu asupra stadiului actual al tehnologiei pot fi găsite în articolul [13].

În ceea ce privește designul arhitecturii, sistemele autonome pot fi clasificate în două categorii principale, modulare și end to end. Sistemele modulare sunt împărțite în componente distincte, fiecare responsabilă pentru o funcționalitate specifică, precum percepția, planificarea traiectoriei și controlul vehiculului. În contrast, sistemele end to end produc acțiuni de comandă direct pe baza datelor de la senzori, învățând să percepă și să acționeze într-un mod asemănător cu cel al oamenilor. În continuare, vor fi explorate în detaliu aceste două tipuri de arhitectură.

#### *1.4.3. Sisteme end to end de conducere*

Fata de sistemele de conducere convenționale, care abordează o strategie de implementare modulară, sistemele end to end combină sarcinile de percepție, predicție și planificare într-un sin-

gur model de învățare automată, astfel sistemul este optimizat pentru a realiza o singura sarcină. Alte avantaje pe care sistemele end to end le oferă sunt eficiența computațională superioară datorită sarcinii unificate și eliminarea erorilor de propagare dintre module.

Acest tip de sisteme încă nu și-a făcut apariția pe străzile din lumea reală ci doar în simulări sau medii controlate, însă în ultimii ani interesul pentru aceste sisteme a crescut datorită simplității designului, comparativ cu sistemele conventionale modulare, și a disponibilității seturilor de date.

Cele mai utilizate abordări end to end de conducere autonomă se bazează pe tehnici care utilizează învățarea prin imitație sau învățarea prin întărire.

Învățarea prin imitație se bazează pe principiul învățării din demonstrațiile experților. Aceste demonstrații antrenează sistemul să imite comportamentul unui expert în diferite scenarii de conducere. Spre deosebire de învățarea prin întărire, învățarea prin imitație abordează paradigma învățării supervizate. În această paradigmă, algoritmul este antrenat pe un set de date etichetat, unde fiecare exemplu din setul de date conține o intrare și ieșirea dorită corespunzătoare. În contextul conducerii autonome, aceasta înseamnă că sistemul învăță să mapeze intrările de la senzori, cum ar fi imagini sau date de la LIDAR, la acțiunile de control, cum ar fi direcția, accelerarea sau frânarea, pe baza exemplelor oferite de comportamentul uman expert. Acest proces necesită o cantitate semnificativă de date etichetate pentru a asigura că sistemul poate generaliza corect și se poate adapta la diverse scenarii de conducere.

Învățarea prin întărire este un tip de tehnică de învățare automată care permite unui agent să învețe într-un mediu interactiv prin încercare și eroare folosind feedback din propriile acțiuni și experiențe. Spre deosebire de învățarea prin imitație, unde feedback-ul furnizat agentului este un set corect de acțiuni pentru îndeplinirea unei sarcini, învățarea prin întărire folosește recompense și pedepse ca semnale pentru comportamentul pozitiv și negativ. Învățarea prin întărire nu dispune de un set de date, antrenarea unei astfel de rețele se realizează online prin încercare și eroare, scopul fiind maximizarea funcției recompensă.

ALVINN[14] a fost primul sistem autonom care să utilizeze o rețea neuronală, dezvoltat de Universitatea Carnegie Mellon în anul 1988, această mașină era capabilă să navigheze autonom printr-o zonă împădurită la viteze mici de 0.5 metri pe secundă, rețeaua rulând pe un procesor alimentat de un generator de cinci kW.

Setul de date era format dintr-o corespondență directă între două imagini, una de la cameră și cealaltă de la senzorul de rază laser, și valoarea unghiului de viraj. Prin acest proces de antrenament, sistemul ALVINN a învățat să asocieze informația vizuală din imaginile atât de la cameră, cât și de la senzorul de rază laser, cu unghiul de viraj corespunzător, permitând astfel vehiculului să navigheze terenul în mod autonom.

Robotul DAVE (DARPA Autonomous Vehicle) este un vehicul autonom dezvoltat de Defense Advanced Research Projects Agency (DARPA) din SUA[15]. Proiectul DAVE face parte din eforturile mai ample ale DARPA de a avansa tehnologiile autonome, precum competițiile Grand Challenge și Urban Challenge.

Scopul principal al proiectului DAVE a fost de a explora beneficiile învățării automate end to end pentru controlul vehiculelor autonome. Proiectul a urmărit să abordeze provocările legate de navigarea pe teren necunoscut, evitând obstacolele precum pietre, copaci și sănțuri.

Mașina autonomă DAVE funcționa utilizând o rețea neuronală convoluțională pentru a prelua imaginile de la camerele stânga și dreapta și a prezice unghiul de direcție. Aceasta a fost antrenată să mapeze perechile de imagini brute în format YUV la unghiurile de direcție ale șoferului uman. Rețeaua convoluțională avea 3,15 milioane de conexiuni și 71.900 de parametri antrenabili, permitând vehiculului să navigheze autonom, adaptând unghiul de direcție pe baza datelor vizuale.

După succesul și lecțiile învățate din proiectul inițial DAVE, aproximativ 12 ani mai târziu, NVIDIA a dezvoltat sistemul DAVE-2 cu scopul de a construi pe baza robotului DAVE pentru a obține o soluție end to end capabilă să navigheze pe drumuri publice. DAVE-2 reprezintă o îmbunătățire semnificativă față de predecesorul său, fiind capabil să navigheze autonom pe autos-

trăzi, străzi locale și parcări, în diverse condiții meteorologice, cum ar fi ploaie, ceață și ninsoare. Această versiune utilizează, la fel ca predecesorul său, o rețea neuronală convoluțională pentru a procesa imaginile capturate de cele trei camere ale vehiculului și pentru a determina direcția corectă de conducere.

Rețeaua neuronală utilizată în DAVE-2, spre deosebire de predecesorul său, are aproximativ 250 de mii de parametri și 27 de milioane de conexiuni. Similar cu DAVE, rețeaua a fost antrenată să imite comportamentul uman, preluând controlul direcției vehiculului pe baza inputurilor vizuale.

Alte soluții end-to-end, precum cele prezentate în articolele [16] și [17], folosesc de asemenea rețele neuronale convoluționale antrenate pe imagini pentru a prezice unghiul și/sau viteza vehiculului. În articolul [16], Zihao Nie și Jian Qu antrenează o rețea ResNet [18] pentru a urmări axa drumului pe baza imaginilor de la camera. Khidhir et al. [17] implementează un sistem end to end care utilizează o rețea neuronală de convoluție pentru a produce comenzi de control lateral și longitudinal pe baza imaginilor achiziționate de la o cameră video. Diferențele între aceste abordări constau în etichetarea setului de date și arhitectura rețelei, însă ambele sisteme au ca rezultat final producerea unghiului de viraj pe baza imitării unui șofer uman.

#### *1.4.4. Sisteme modulare de conducere*

Funcțiile esențiale ale unui sistem de conducere autonomă modular includ localizarea, perceptia, planificarea și controlul vehiculului. Localizarea determină poziția precisă a vehiculului în mediu folosind date de la GPS și senzori IMU. Percepția interpretează datele de la senzori precum camere, LIDAR și radar, identificând obiectele din mediu și clasificându-le. Planificarea generează traiectoria optimă pe baza datelor de percepție și localizare, asigurând evitarea obstacolelor și respectarea regulilor de circulație. Controlul vehiculului execută traiectoria planificată prin comenzi precise către sistemele de direcție, accelerare și frânare.

Dezvoltarea modulelor individuale în cadrul unui sistem de conducere autonomă împarte problema complexă a conducerii autonome într-un set de probleme bine definite și specializate, ceea ce face stagiul de dezvoltare un proces bine organizat și intuitiv.

Modulul de percepție este un component de baza într-un sistem autonom, însă acesta reprezintă un punct critic de limitare. Performanța unui vehicul autonom depinde în mare măsură de eficiența componentelor din modulul de percepție cum ar fi algoritmi inadecvați, dar și de fiabilitatea și robustețea senzorilor.

Modulul de percepție este responsabil pentru procesarea și interpretarea datelor furnizate de senzori, construind astfel o reprezentare simplificată a mediului înconjurător. Acesta integrează diverse tehnologii, cum ar fi rețele neuronale de detecție și segmentare, algoritmi de urmărire și estimare a traiectoriilor vehiculelor. Modulul detectează și clasifică obiectele, marcajele rutiere, semnele de circulație și obstacolele. De asemenea, este responsabil pentru detectarea și urmărirea obiectelor în mișcare, detectarea benzii de drum, evaluarea distanțelor și vitezelor relative. Toate aceste informații sunt utilizate de modulele de planificare și control pentru a lua decizii corecte și sigure de navigare.

Fără îndoială cel mai discutat subiect în domeniul vederii artificiale îl constituie detectarea obiectelor, cercetarea vederii artificiale își are originile încă din anii 1960, însă abia După anii 2000 metodele de detectie au început să ofere rezultate promițătoare. Până în anul 2014 vederea artificială a fost predominată de tehnici tradiționale cum ar fi detectorii Viola Jones [19], [20] sau detectorul HOG [21], însă în 2012 arhitectura AlexNet [22] domină în competiția organizată de ImageNet [23] ceea ce aduce atenția către rețelele neuronale de convoluție. În prezent sarcina detectiei de obiecte este abordata de tehnici bazate pe învățare profunda.

Detectoarele de obiecte pot fi clasificate ca detectoare într-un stagiul sau detectoare în două stagii.

Detectoarele în două stagii oferă o precizie superioară de recunoaștere și localizare în ima-

gine, deși la prețul vitezei de inferență. Unul dintre cele mai populare detectoare în două stagii este familia de rețele R-CNN (Region-Based Convolutional Neural Network), care include Fast R-CNN, Faster R-CNN și Mask R-CNN. În prima etapă a unui detector în două etape, regiunile de interes sunt propuse folosind tehnici precum căutarea selectivă sau rețelele de propunere de regiune (RPN). Aceste regiuni propuse sunt apoi clasificate și rafinate în a doua etapă pentru a genera detecții finale. Acest proces în două etape permite o localizare mai precisă a obiectelor, în special a celor mai mici, ceea ce duce la o precizie mai mare, însă necesită o putere de procesare pe măsură.

Detectoarele într-un singur stagiu sunt algoritmi de detectare a obiectelor care își propun să detecteze și să clasifice obiectele printr-o singură procesare a rețelei, fără a fi nevoie de un pas separat de propunere a regiunilor. Aceste detectoare sunt proiectate pentru a fi eficiente și rapide, făcându-le potrivite pentru aplicații în timp real unde viteza este crucială.

Una dintre caracteristicile cheie ale detectoarelor într-un singur stagiu este simplitatea lor în design. Ele folosesc în mod tipic o singură rețea neuronală pentru a prezice direct casetele de delimitare și probabilitățile de clasă pentru obiectele dintr-o imagine. Detectoarele într-un singur stagiu folosesc adesea tehnici precum casetele ancoră pentru a detecta obiecte la diferite scări în cadrul unei imagini. Prin incorporarea acestor mecanisme, detectoarele într-un singur stagiu pot captura eficient obiecte de diferite dimensiuni și forme într-o singură trecere prin rețea.

Un alt avantaj al detectoarelor într-un singur stagiu este capacitatea lor de a gestiona eficient sarcini dense de detectare a obiectelor. Deoarece procesează întreaga imagine odată, acestea pot detecta multiple obiecte într-o scenă simultan fără a fi necesare calcule suplimentare sau pași de post-procesare.

Cu toate acestea, detectoarele într-un singur stagiu pot întâmpina dificultăți în localizarea precisă a obiectelor mici sau a obiectelor cu forme complexe. Aceste dificultăți apar din cauza hărților de caracteristici mai puțin detaliate utilizate în detectoarele într-un singur stagiu, care pot limita capacitatea lor de a identifica poziția exactă a acestor obiecte în cadrul unei imagini.

În domeniul detectării obiectelor în timp real, YOLO (You Only Look Once) [24] și SSD (Single Shot Multibox Detector) [25] sunt două arhitecturi foarte apreciate. O scurtă explorare a literaturii pe această temă ar dezvăluia cu ușurință proeminența lor. Aceste rețele au atras atenție semnificativă datorită performanței lor impresionante și abordărilor inovatoare.

Redmon et al. introduce arhitectura YOLO [24] în anul 2016 la conferința CVPR(Computer Vision and Pattern Recognition), acesta a introdus o abordare inovatoare pentru detecția de obiecte care a permis procesarea în timp real. Spre deosebire de metodele anterioare arhitectura YOLO combina într-o singură procesare de rețea, determinarea casetelor de delimitare și clasificarea obiectelor. Rețeaua atinge viteze superioare de inferență față de alți detectori în timp real totodată dublând acuratețea. De-a lungul timpului, arhitectura s-a dezvoltat producând o serie de cele mai bune detectoare de obiecte în timp real, deși Redmon și-a oprit cercetarea în domeniul viziunii computerizate odată cu publicarea arhitecturii YOLOv3[26], conceptul și vizuirea arhitecturii continuă sa fie dezvoltată de mai mulți autori, cea mai recentă adăugare la familia YOLO fiind YOLOv9 [27].

În general, arhitectura unui SSD constă în mod obișnuit dintr-o rețea de bază, cum ar fi VGG sau ResNet, care este pre-antrenată pe un set mare de date de clasificare a imaginilor, cum ar fi ImageNet. Această rețea de bază este apoi urmată de mai multe straturi suplimentare care reprezintă capul detectorului. Aceste straturi suplimentare sunt responsabile pentru detectarea obiectelor la diferite scări și sunt de obicei compuse din straturi convolutionale și straturi de grupare.

Deși utilizarea unei rețele de detecție reprezintă un punct de plecare excelent pentru detectarea semnelor de circulație, semafoarelor, obstacolelor și altor obiecte, abordările actuale tend să folosească metode mai complexe pentru a asigura o detecție fiabilă și precisă. Aceste abordări implică utilizarea unor detectoare 3D, care se bazează pe datele furnizate de LIDAR, camere ste-

reș sau chiar camere individuale. Mai mult de atât datele de la acești senzori sunt combinate prin tehnici de fuziune a senzorilor, îmbunătățind astfel precizia și reducând incertitudinea. Un exemplu notabil este proiectul Apollo[28], care se află printre liderii în domeniul conducerii autonome, integrând camere și LIDAR pentru o detecție 3D fiabilă a obiectelor.

Totuși, utilizarea unei rețele de detecție reprezintă un start solid și practic pentru majoritatea aplicațiilor de bază în detectarea obiectelor. Aceasta oferă o soluție accesibilă și eficientă pentru implementarea rapidă a sistemelor de detecție.

Gestionarea resurselor de procesare este o problemă majoră într-un sistem încorporat, aşadar alegerea modelului de detecție este un pas esențial pentru a atinge un balans între performanță și acuratețea acestuia, deseori aceste arhitecturi prezintă versiuni simplificate, cu un număr mai mic de parametri pentru a permite inferență pe sisteme încorporate cum ar fi YOLOv3 tiny sau alte arhitecturi sunt dezvoltate specific pentru a rula pe sisteme încorporate.

Pe de parte, cea mai populară rețea de detecție pentru sisteme încorporate este YOLOv3 tiny. Aceasta a devenit extrem de cunoscută datorită balansului între performanță și eficiență, fiind capabilă să ruleze în timp real pe dispozitive cu resurse limitate, precum placa Jetson Nano. Performanțele YOLOv3 tiny pe Jetson Nano sunt remarcabile[29], reușind să proceseze 25 de cadre pe secundă la o rezoluție de 416x416 pixeli, asigurând astfel un punct de plecare solid pentru dezvoltarea aplicațiilor autonome.

Detectarea drumului de mers este esențială pentru funcționarea vehiculelor autonome, asigurând că acestea pot naviga în siguranță pe diverse tipuri de drumuri. Tehnicile tradiționale, care nu utilizează învățarea automată, se bazează în principal pe detectarea marcajelor albe pentru a identifica suprafața drumului de condus. Aceste metode folosesc algoritmi de procesare a imaginilor pentru a recunoaște și urmări liniile albe de pe carosabil. Cu toate acestea, aceste tehnici au limitări semnificative, nu funcționează eficient în absența marcajelor pe drum, pe drumurile rurale, în zonele de construcție sau pe drumurile deteriorate.

Sistemele autonome de astăzi utilizează o combinație de tehnici avansate pentru a detecta banda de drum, inclusiv rețele de detecție a liniilor și segmentarea imaginii. Acestea integrează senzori diferenți, precum camere video și LIDAR, pentru a obține o reprezentare detaliată și precisă a drumului. O analiză detaliată a tehnicii utilizate în prezent este realizată de Zakaria, Noor Jannah et al. în articolul [30].

Un exemplu notabil este utilizarea segmentării semantice, care permite vehiculului să clasifice fiecare pixel al imaginii într-o anumită categorie, cum ar fi banda de drum, trotuar, vegetație sau obstacol. Acest proces asigură o detecție robustă a drumului, chiar și în absența marcajelor albe.

Articolul [31] explorează importanța segmentării semantice în conducerea autonomă, concentrându-se pe percepția mediului. Evaluează mai multe modele de segmentare, inclusiv FasterSeg, pentru acuratețe și eficiență în aplicații în timp real pe dispozitive încorporate cu putere redusă. Studiul utilizează o metodă de generare a datelor de antrenament sintetice cu ajutorul simulatorului Gazebo, combinând imagini simulate și reale. O comparație între perspectivele de vedere la persoana întâi și cele bird's eye view dezvăluie niveluri generale similare de precizie. În general, constatăriile subliniază importanța modelelor de segmentare în conducerea autonomă, evidențierind potențialul arhitecturii FasterSeg în atingerea unui echilibru între viteză și precizie în aplicațiile din lumea reală.

O altă soluție validă pentru segmentarea benzii este utilizarea rețelei FastSCNN. Performanța acesteia a fost evaluată pe platforma Jetson Nano, demonstrând o performanță acceptabilă pentru aplicații în timp real[32]. FastSCNN este apreciată nu doar pentru eficiența sa, dar și pentru suportul extins pe platforme precum PaddleSeg[33] și MMSegmentation[34], oferind flexibilitate în implementare.

Modulul de planificare este responsabil pentru calcularea traiectoriei vehiculului și realizarea diverselor manevre, cum ar fi parcare, depășirea obstacolelor. Planificarea rutei se efectuează

în două etape principale, planificarea globală și planificarea locală a comportamentului și traiecto-  
riei.

În primul rând, planificarea globală necesită integrarea cu modulul de localizare pentru a determina ruta optimă de la punctul de plecare la destinație pe o hartă globală. Algoritmi faimoși, precum Dijkstra și A\*, sunt adesea utilizati în acest context pentru a alege traseul cel mai bun.

Planificarea locală a comportamentului și a traectoriei funcționează împreună pentru a calcula o traectorie locală sigură, bazată pe ruta globală identificată. Deseori aceasta este realizată prin rezolvarea unei probleme de control optim, care minimizează o funcție de cost predefinită, respectând diverse constrângeri. Aceste constrângeri pot include, de exemplu, limitele de viteză, distanțele de siguranță față de alte vehicule și obstacole, sau curburile drumului. Mai multe informații despre tehniciile comune utilizate în planificarea locală și globală pot fi găsite în articolul realizat de Teng, Siyu et al.[35], care oferă o analiză detaliată a algoritmilor de planificare actuali.

Modulul de control al mașinilor autonome este responsabil pentru transformarea deciziilor de navigație în mișcări efective ale vehiculului. Acesta gestionează comenzi precise către sistemele de direcție, accelerare și frânare pentru a asigura mișcările dorite.

Controlul clasic, precum regulatoarele PID, oferă o soluție simplă și eficientă pentru generarea comenziilor de control atât pentru direcție, cât și pentru viteză. Regulatoarele PID funcționează prin ajustarea continuă a diferenței dintre valoarea dorită și cea măsurată, prin aplicarea unei combinații de acțiuni proporționale, integrale și derivate. Acestea sunt ușor de implementat și ajustat, oferind o soluție simplă și rapidă pentru controlul vehiculelor autonome.

Cu toate acestea, soluțiile mai complexe, cum ar fi utilizarea unui model de control predic-  
tiv (Model Predictive Control, MPC), sunt adesea preferate pentru controlul vehiculelor autonome datorită capacitatei lor de a anticipa și optimiza performanța pe un orizont de timp[36]. MPC funcționează prin rezolvarea unei probleme de optimizare în timp real, care ia în considerare un model al dinamicii vehiculului și limitele impuse de mediul înconjurător. Aceasta permite genera-  
rea unui set de comenzi optime care minimizează o funcție de cost predefinită, ținând cont de restricțiile și obiectivele specifice, cum ar fi evitarea obstacolelor, menținerea stabilității și confor-  
tului pasagerilor.

MPC este foarte eficient deoarece poate anticipa viitoarele stări ale vehiculului și poate ajusta comenziile în consecință, oferind astfel un control precis și adaptiv. Aceasta metodă este robustă la variațiile condițiilor de drum și la perturbări externe, ceea ce o face ideală pentru aplicarea în conducerea autonomă.

Aceasta secțiune a oferit o privire de ansamblu asupra componentelor esențiale, un vehi-  
cul autonom modular este mult mai complex decât ceea ce a fost prezentat până acum, acesta integrează și alte funcționalități cruciale, precum prediciția comportamentului altor participanți la trafic, coordonarea între vehicule și gestionarea interacțiunilor complexe cu infrastructura rutieră. În continuare, următorul capitol va explora tehnologiile specifice alese pentru proiectarea și im-  
plementarea proiectului, evidențiind instrumentele și metodele utilizate pentru a aduce la viață un sistem autonom funcțional. Aceasta va include detalii despre hardware-ul, software-ul și algoritmii care stau la baza proiectului.

## Capitolul 2. Proiectarea sistemului

Capitolul anterior a prezentat tehnologiile și tehniciile de ultimă oră în domeniul conducerii autonome. Acest capitol se concentrează pe proiectarea sistemului utilizând aceste tehnologii. Prin integrarea acestor tehnologii, se urmărește crearea unui sistem autonom, capabil să îndeplinească cerințele concursului IDEAS.

Majoritatea abordărilor în domeniul conducerii autonome recurg la un sistem modular, și există motive bine intemeiate pentru aceasta. Sistemele autonome end to end sunt lipsite de flexibilitate și control, modificările fiind dificil de implementat fără a afecta întregul sistem. În contrast o abordare modulară este mai intuitivă pentru dezvoltare. De asemenea, un sistem modular permite un control mai bun asupra funcționării mașinii și a deciziilor pe care aceasta le ia, facilitând în același timp implementarea unor soluții hardcodate pentru anumite necesități specifice cum ar fi oprirea pentru un anumit timp la diversele semne, sau inițierea depășirii în urma detectării unui obstacol la o anumita distanță. În plus, sistemele modulare oferă scalabilitate, permit dezvoltarea și menținerea eficientă, facilitează testarea individuală a componentelor, asigurând astfel un sistem mai robust și eficient.

În continuare, acest capitol se concentrează pe trei aspecte cheie, platforma hardware, simulatorul utilizat și arhitectura software. Prima parte este dedicată platformei hardware, examinând în detaliu capacitatele placii Jetson Nano de 4 GB, aleasă pentru implementare. Vor fi discutate caracteristicile hardware-ului, avantajele oferite de această platformă și modul în care acestea contribuie la funcționarea eficientă a aplicației.

Apoi va fi explorat simulatorul utilizat pentru testarea și validarea aplicației. Se va analiza modul în care simulatorul recreează scenariile conform competiției.

Ultima parte se concentrează pe arhitectura software a aplicației, prezentând modulele principale, relațiile funcționale dintre acestea și tehnologiile utilizate pentru implementarea lor. Se va explora cum aceste module interacționează pentru a forma un sistem coerent și eficient, subliniind importanța abordării modulare în dezvoltarea unei mașini autonome.

### 2.1. Hardware

Pentru implementarea și testarea aplicației, sistemul hardware utilizat este format din două componente principale, un laptop și o placă NVIDIA Jetson Nano. Acestea sunt interconectate prin rețea locală, care permite transmiterea fluxului video de la laptop către Jetson Nano și, în sens invers, trimitera comenziilor de control de la Jetson Nano către laptop. Alegerea acestei configurații simplificate este ideală pentru simulare, asigurând o comunicare eficientă și rapidă între cele două componente.

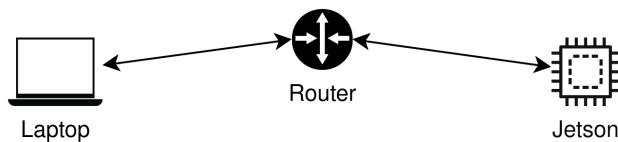


Figura 2.1. Comunicarea între Jetson și laptop.<sup>3</sup>

Familia NVIDIA Jetson prezintă o serie de plăci de dezvoltare special concepute pentru sarcini de inteligență artificială (AI) la scară mică și medie. În cadrul acestei game, placă NVIDIA Jetson Nano se evidențiază ca o soluție compactă, puternică și accesibilă. Din punct de vedere al memoriei RAM, modelul prezintă două variante, o variantă de 2GB și celalătă de 4GB. Din-

<sup>3</sup> imagine creata cu ajutorul aplicației web [37]

tre acestea, doar cea de 4GB este potrivită pentru aplicații AI mai complexe, oferind suficientă memorie pentru a rula retele neuronale și alți algoritmi avansati.

Avantajul acestei placi fata de alte sisteme cum ar fi Raspberry Pi este faptul ca dispune de un procesor grafic cu 128 de nuclei CUDA oferind un avantaj semnificativ în ceea ce privește performanța și capacitatele AI.

Jetson Nano dispune de un procesor quad-core ARM Cortex-A57, care, împreună cu procesorul grafic, asigură o putere de calcul suficientă pentru o varietate de aplicații AI. Placa suportă diverse interfețe de conectivitate, inclusiv HDMI, USB 3.0, Ethernet Gigabit și pini GPIO, facilitând integrarea într-o gamă largă de proiecte și dispozitive.

Jetson Nano vine cu suport software extins, inclusiv kitul de dezvoltare JetPack, care include un set complet de instrumente și biblioteci pentru dezvoltarea AI. JetPack oferă suport pentru TensorFlow, PyTorch, cuDNN, și multe alte cadre software, simplificând procesul de dezvoltare și implementare a aplicațiilor AI.

Această placă este ideală pentru o varietate de utilizări, de la vehicule autonome și roboți până la sisteme de supraveghere și recunoaștere facială. Flexibilitatea și puterea sa de procesare fac din Jetson Nano o alegere excelentă pentru dezvoltatorii care doresc să creeze soluții AI eficiente și scalabile.

Laptopul utilizat în cadrul sistemului joacă un rol crucial în preluarea fluxului video și transmiterea acestuia către placa Jetson Nano, precum și în recepționarea și interpretarea comenziilor de control trimise de Jetson.

Laptopul este echipat cu un procesor Intel Core i5-11300H, 16 GB de RAM și o placă grafică dedicată NVIDIA RTX 3050. Aceste caracteristici oferă puterea de calcul necesară pentru a rula aplicațiile de simulare și procesare a datelor eficient. Placa grafică NVIDIA RTX 3050 permite rularea aplicațiilor grafice intensive și a platformelor software pentru inteligență artificială, esențiale pentru antrenarea și testarea modelului de segmentare.

## 2.2. Simulatorul

Accesul la pistă este posibil doar în timpul concursului, aşadar testarea sistemului se realizează printr-o simulare în Unity, unde se recreează o pistă virtuală. Această abordare permite evaluarea funcționalității și performanței sistemului într-un mediu controlat și reproductibil. Unity oferă un mediu flexibil pentru dezvoltarea și testarea aplicațiilor 3D, furnizând numeroase facilități pentru crearea de medii realistice atât din punct de vedere al fizicii, cât și vizual.

Modelele pistei, semnelor de circulație și semaforului au fost create în Blender. Aceste modele au fost apoi importate într-o scenă predefinită din Unity, care include setări configurate pentru iluminare și alte aspecte vizuale. O vedere de sus a pistei în simulator, prezentând configurația acesteia și elementele de mediu, poate fi observată în figura de mai jos.

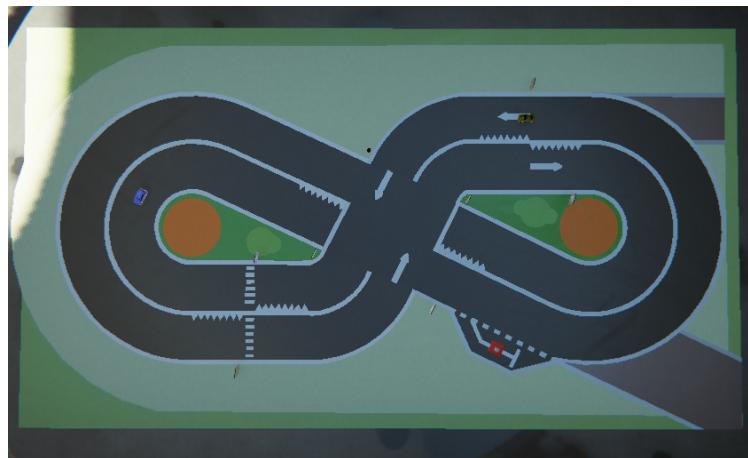


Figura 2.2. Vedere de sus a pistei în simulator.

Mașinuță este modelată printr-un corp rigid(Rigidbody) la care se aplică cate trei forțe la poziția fiecărei roți. Prințipiu de funcționare al acestui model constă în utilizarea razeelor de detecție (raycasts) pentru a simula interacțiunile fizice cu suprafața pistei și pentru a determina dinamica vehiculului. Modelul acestei mașini se bazează pe exemplul din depozitul[38].

Forța de suspensie împinge corpul mașinuței în sus, simulând efectul suspensiilor reale ale vehiculului. Razele de la poziția fiecărei roți sunt îndreptate către sol pentru a determina distanța până la suprafața pistei. În funcție de această distanță, se aplică o forță proporțională care împinge corpul mașinii în sus pentru a menține vehiculul la înălțimea corectă. Forța de accelerare este aplicată în direcția de orientare a roților și este responsabilă pentru mișcarea vehiculului înainte sau înapoi, direcția și magnitudinea acestei forțe fiind determinate de inputurile de accelerare și frânare. Forța laterală, pe de altă parte, simulează rezistența roților la derapare, contribuind la stabilitatea vehiculului în viraje și la menținerea tracțiunii pe suprafața pistei.

Aceste forțe sunt calculate și aplicate în mod continuu pentru fiecare roată, rezultând într-o simulare realistă a comportamentului vehiculului.

Scriptul responsabil pentru controlul mașinii în Unity primește comenzi de la placa Jetson, cum ar fi viteza și controlul lateral, și le aplică sub forma unor forțe asupra celor patru roți, după cum a fost menționat anterior. În cazul controlului lateral, acesta utilizează un controler PID care menține vehiculul pe banda de rulare. Parametrii controlerului PID au fost ajustați empiric.

Camera este plasată deasupra mașinii la înălțimea maxima prevăzută în regulament și este ușor înclinată în jos până la limita care permite o vedere completă a semnelor de circulație. Aceasta asigură o captare optimă a pistei și a semnelor, fără a compromite niciuna dintre acestea.

Pentru a imita condițiile reale ale camerei utilizate în timpul concursului, distorsiunile provocate de lentilă în realitate au fost simulate pentru camera din Unity. Deși aceste distorsiuni nu sunt identice cu cele reale și sunt mai atenuate pentru a facilita procesul de calibrare, includerea lor a fost esențială pentru a valida un algoritm de calibrare dezvoltat în timpul competiției, care va fi prezentat în detaliu ulterior.

Fluxul video al camerei este transmis în exterior prin intermediul unui plugin Unity dezvoltat de Keijiro, numit KlakNDI[39], care utilizează tehnologia Network Device Interface (NDI). Pentru a captura și transmite fluxul video din Unity către exterior, se utilizează OBS Studio, o aplicație de captură și streaming video. OBS Studio integrează un plugin numit DistroAV[40], care de asemenea prin tehnologia NDI extrage fluxul video din simulator și îl transmite către exterior. Fluxul video este transmis, utilizând protocolul RTSP, către un server media numit MediaMTX[41], care gestionează și distribuie acest flux către diverse destinații. În acest caz, MediaMTX asigură disponibilitatea continuă a fluxului video pentru aplicația rulată pe platforma Jetson.

Aplicația de pe Jetson interpretează fluxul video de pe serverul MediaMTX ca și cum ar proveni de la o cameră reală. În același timp, orice alte comenzi emise de către Jetson către simulator, inclusiv comenziile pentru controlul mașinii sau alte operațiuni, sunt gestionate și transmise prin intermediul unei conexiuni socket UDP, asigurând astfel o interacțiune simplă și rapidă între aplicația de control și simulator.

### 2.3. Arhitectura software

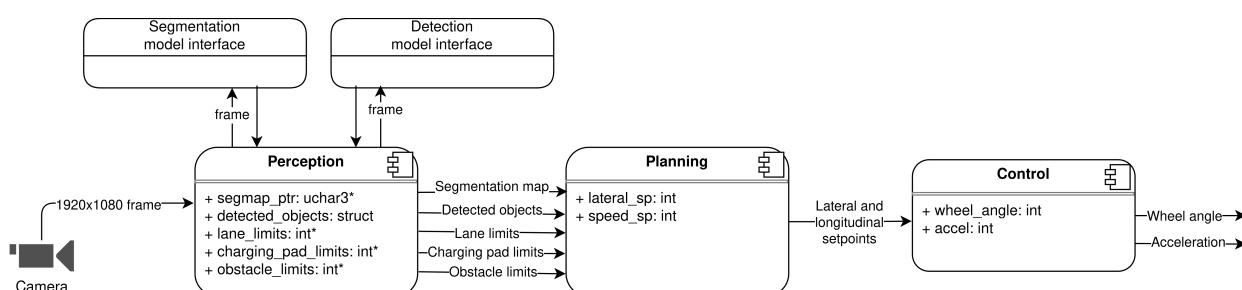
Proiectarea arhitecturii software implică stabilirea unor obiective clare pentru a defini aşteptările și pentru a fragmenta problema principală în subprobleme mai mici, facilitând astfel rezolvarea acestora.

Tabelul 2.1 prezintă cele trei module principale ale sistemului împreună cu funcționalitățile pe care acestea le vor oferi. Prima coloană enumeră cerințele importante pentru fiecare modul, derivează din regulamentul tehnic al concursului. A doua coloană descrie succint aşteptările legate de performanțele mașinii. Ultima coloană specifică modul în care fiecare funcționalitate sau cerință va fi implementată.

Tabelul 2.1. Clasificare cerințe în funcție de modulele sistemului

<b>Modulul de percepție</b>		
Cerință	Descriere	Soluție
1. Detectie semne, stări semafor	Sistemul trebuie să recunoască semnele și stările semaforului în mod fiabil, gestionând corect erorile de clasificare, astfel încât să nu le rateze sau să le interpreteze greșit.	Utilizarea unei rețele neuronale de detecție de obiecte
2. Detectie pista	Sistemul trebuie să identifice cu precizie banda de rulare, obstacolele și poziția acestora, precum și locația stației de încărcare.	Utilizarea unei rețele neuronale de segmentare semantică
<b>Modulul de planificare</b>		
Cerință	Descriere	Soluție
1. Calculare traекторie	Sistemul trebuie să calculeze mijlocul benzii de-a lungul traseului. Dacă este detectat un obstacol pe banda curentă, vehiculul va efectua manevra de depășire.	Utilizarea hărții de segmentare pentru a detecta obstacolul și pentru a identifica limitele benzii.
2. Interacțiunea cu semnele de circulație	Sistemul trebuie să urmeze comportamentul specific fiecarui semn și semafor în urma detecției.	Implementarea unei mașini cu stări pentru a gestiona comportamentul vehiculului
<b>Modulul de control</b>		
Cerință	Descriere	Soluție
1. Acționare controler mașină pe baza comenziilor modulu lui de planificare	Transmisie comenzi de referință pentru controlul longitudinal și lateral	Algoritm de generare comenzi de referință pentru mașină

Următoarea diagramă oferă o vedere de ansamblu asupra fluxului de date în cadrul aplicației. Imaginea capturată de cameră este preluată de modulul de percepție, unde este procesată de cele două rețele pentru a obține informațiile necesare. Modulul de planificare procesează apoi aceste date pentru a genera punctele de referință necesare modulului de control. Modulul de control utilizează aceste puncte de referință pentru a trimite comenziile de direcție și accelerare către vehicul.

Figura 2.3. Fluxul de date în cadrul aplicației.<sup>4</sup>

<sup>4</sup>Imagine creata cu ajutorul aplicației web [37]

### 2.3.1. Tehnologii utilizate și modulele principale

Aplicația este implementată în C++, utilizând librăria jetson-inference[42] pentru a facilita inferența rețelelor neuronale pe platformele Jetson. În implementarea practică, TensorRT este utilizat pentru a accelera și optimiza performanța rețelelor neuronale folosite în modulul de percepție. TensorRT este un cadru software dezvoltat de NVIDIA care permite rularea rapidă și eficientă a modelelor de învățare automată pe plăcile video din gama NVIDIA. Aceasta va asigura că rețelele neuronale pot procesa datele în timp real, o cerință esențială pentru sistemele autonome.

Acest depozit oferă exemple și instrucțiuni pentru achiziția de imagini de la camere, antrenarea rețelelor și inferența acestora, concentrându-se pe performanță. Utilizarea acestui depozit accelerează dezvoltarea și testarea modulului de percepție, oferind un punct de plecare robust pentru proiect. Aceasta permite ca rețelele neuronale să ruleze în mod eficient și să răspundă cerințelor de timp real ale aplicației.

#### 2.3.1.1. Modulul de percepție

Modulul de percepție este responsabil pentru colectarea și interpretarea datelor de mediu. Percepția în vehiculele autonome are un sens similar cu cel pentru oameni, referindu-se la capacitatea de a fi conștient de mediul înconjurător pentru a interacționa cu acesta. Acest modul este crucial, deoarece capacitatea mașinii de a percepe corect mediul înconjurător determină direct performanța sa.

Acest modul funcționează practic ca o interfață pentru cele două rețele neuronale, gestionând atât prelucrarea inițială a imaginilor cât și procesarea rezultatelor obținute. În primul rând, modulul pe percepție primește imagini brute de la camera și le preprocesează pentru a le aduce la formatul și dimensiunile necesare pentru inferența rețelelor neuronale. Aceste operațiuni de preprocesare includ redimensionarea, normalizarea și alte ajustări specifice fiecărei rețele. După ce imaginile sunt procesate de rețelele neuronale, modulul filtrează și interpretează ieșirile, extragând informațiile relevante și eliminând datele redundante sau zgomotoase.

Detectia semnelor și a semafoarelor este esențială pentru luarea deciziilor corecte de conducere, permitând mașinii autonome să identifice cele patru semne de circulație și cele trei stări ale unui semafor. Pentru această funcție, se utilizează o rețea neuronală de detectie a obiectelor.

Pentru implementarea eficientă a acestei funcționalități, s-a ales utilizarea rețelei YOLOv3 tiny. Această alegere este motivată de mai mulți factori importanți. YOLOv3 tiny este o rețea foarte populară, bine documentată și larg utilizată în domeniul detectiei de obiecte. Documentația bogată a acestei rețele facilitează înțelegerea și implementarea, reducând astfel timpul necesar pentru dezvoltare și depanare.

În al doilea rând, YOLOv3 tiny este optimizată pentru a rula eficient pe platforme înglobate, cum ar fi Jetson Nano atingând viteze de 27 de imagini procesate pe secunda la o rezoluție de 416x416[43]. Acest lucru face ca YOLOv3 tiny să fie o alegere ideală pentru detectia semnelor și a semafoarelor.

Datorită poziționării constante a semnelor și semaforului pe partea dreaptă a pistei, detectarea acestora poate fi îmbunătățită semnificativ prin extragerea unei zone de interes specifice pentru inferență. În acest context, rețeaua de detectie procesează doar o zonă de 540x540 pixeli din colțul drept sus al imaginii de 1920x1080 pixeli. Această abordare permite focalizarea pe o regiune specifică unde sunt plasate semnele, evitând necesitatea de a micșora întreaga imagine. Astfel, detectia este mai precisă, deoarece semnele și semaforele rămân la dimensiuni mai mari și mai clare în cadrul zonei de interes, ceea ce îmbunătățește acuratețea rețelei de detectie. Această zonă de interes este ulterior redimensionată la dimensiunile de intrare ale rețelei, adică 320x320 pixeli.

Detectia precisă a suprafeței de drum este crucială pentru înțelegerea scenariului în conducerea autonomă. În timp ce abordările tradiționale, precum modelarea marajelor benzii ca forme

geometrice, cum ar fi linii, polinoame sau spline, sunt alternative viabile, acestea au limitări în ceea ce privește timpul de procesare și acuratețea, care pot fi depășite prin valorificarea progreselor recente în algoritmii de învățare profundă. Mai exact, segmentarea semantică oferă o soluție mai eficientă și mai precisă. Retelele de ultimă generație, precum FasterSeg[44] și FastSCNN[45], au demonstrat performanțe în timp real pe platforme înglobate precum Jetson, făcându-le ideale pentru implementare în mașinile autonome. În acest context, s-a ales rețeaua FastSCNN datorită maturității sale, suportului extins din partea comunității și a faptului că performanța sa pe Jetson Nano a fost deja validată prin teste anterioare[32].

Aceasta procesează imagini RGB la rezoluția de 1024x512 pixeli, această dimensiune fiind aleasă pentru a oferi un echilibru optim între performanță și acuratețea segmentării. Rețeaua este capabilă să diferențieze între șase clase distincte, fundal, banda din stânga, banda din dreapta, marcajele albe, obstacol și stația de încărcare.

Adoptarea unei noi perspective, așa cum au propus Senay Cakir et al. în articolul [31], poate fi esențială în acest context. Tehnica utilizată de acestia implică perspectiva *bird's eye view*, care transformă imaginile capturate de camera într-o perspectivă de sus. Această abordare oferă o reprezentare bidimensională a mediului, facilitând detectarea și analiza drumului și a elementelor din împrejurimi.

Perspectiva *bird's eye view* facilitează procesul de planificare a traiectoriei, deoarece elementele esențiale, precum benzile de circulație, intersecțiile și obstacolele, sunt mai ușor de identificat și de procesat. Astfel, algoritmii de planificare sunt mai simpli și pot genera trasee mai precise și mai eficiente.

Depozitul IPM-Master[46] constituie o resursă valoroasă pentru implementarea algoritmului de *bird's-eye view* (BEV). Acest depozit furnizează un algoritm de schimbare de perspectivă în python. Totuși, este important de menționat că algoritmul furnizat nu este optimizat pentru utilizare în timp real. Prin urmare, algoritmul este rescris în CUDA C++ pentru a permite rularea acestuia pe procesorul grafic. Această abordare aduce beneficii semnificative în ceea ce privește timpul de preprocesare a imaginii.

Calibrarea imaginii nu este esențială pentru rețeaua de detecție, deoarece accentul nu se pune pe forma exactă a obiectelor, ci pe identificarea lor fiabilă în imagine. În schimb, pentru rețeaua de segmentare, distorsiunile în urma schimbării de perspectivă devin extrem de pronunțate. Aceste distorsiuni pot afecta precizia segmentării, ducând la obținerea unor imagini segmentate care nu reflectă corect structura reală a scenei. Astfel, pentru segmentare, calibrarea este crucială pentru a corecta perspectivele deformate și a produce rezultate care sunt cât mai fidele realității.

O simplă căutare pe Google a termenului *camera calibration* va aduce numeroase exemple utilizând OpenCV. Calibrarea camerei implică determinarea parametrilor intrinseci (distanța focală, centrul optic) și extrinseci ai camerei. Acești parametri sunt estimați cel mai des utilizând metoda Zhang, care presupune captarea a mai multor imagini ale unui model cunoscut, cum ar fi o tablă de șah, în diverse poziții. Datorită geometriei cunoscute a tablei de șah, coordonatele 3D ale colțurilor acesteia sunt deja cunoscute în propriul sistem de coordonate. Aceste coordonate, împreună cu pozițiile lor 2D în imagini, permit extragerea parametrilor intrinseci, a coeficientilor de distorsiune și, optional, a parametrilor extrinseci pentru fiecare imagine.

În continuare, obținerea imaginii fără distorsiuni în OpenCV se poate realiza fie prin apelarea funcției *undistort*, fie prin utilizarea funcției *remap*. De fapt, funcția *undistort* folosește intern *remap*, însă diferența este că apelează mereu *initUndistortRectifyMap* pentru a simplifica utilizarea. Într-un scenariu în care viteza contează, este mai eficientă apelarea a *initUndistortRectifyMap* o singură dată, iar apoi remaparea pixelilor imaginii prin apelarea funcției *remap*, ceea ce favorizează performanța. Astfel, problema calibrării camerei a fost redusă la o simplă problemă de remapare a pixelilor într-o imagine, o sarcină foarte eficientă de rezolvat utilizând puterea de calcul în paralel a procesorului grafic.

Datorită modului în care sunt montate camerele, imaginile capturate sunt afectate de efec-

tul de perspectivă, unde obiectele aflate la distanță par mai mici și liniile paralele tind să se intersecteze. Acest efect de perspectivă complică procesarea și analiza avansată a imaginilor, creând dificultăți pentru algoritmii de planificare a traectoriei. Pentru a soluționa această problemă, este necesară o transformare a perspectivei, astfel încât imaginile brute să fie convertite într-o vedere de sus (bird's eye view). Această transformare elimină efectul de perspectivă și oferă o reprezentare clară și precisă a mediului înconjurător, facilitând analiza și deciziile algoritmilor de planificare.

In teorie imaginea în vedere de sus se poate obține utilizând o matrice de omografie (2.1) care stabilește relația dintre pixelul  $(x', y')$  din imaginea bev și pixelul  $(x, y)$  din imaginea originală. Această transformare este cunoscută sub numele de inverse perspective mapping (IPM). Un algoritm IPM clasic primește imaginea frontală, aplică o omografie și creează o vedere de sus a scenei capturate prin maparea pixelilor într-un cadru 2D.

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} \quad (2.1)$$

Pentru a efectua transformarea, coordonatele pixelilor  $(x, y)$  sunt extinse la coordonate omogene adăugând o componentă 1, astfel încât vectorul pixelului devine  $(x, y, 1)$ . În final rezultatul va fi în coordonate omogene și va trebui adus în coordonate carteziene după ecuația (2.2).

$$x_{final} = \frac{x'}{w'} \quad y_{final} = \frac{y'}{w'} \quad (2.2)$$

După cum se observă operația de aplicare a transformării omografice asupra fiecărui pixel implică înmulțirea coordonatelor sale omogene cu matricea de omografie și aducerea rezultatului în coordonate carteziene. Deși această operație în sine nu este complexă, pentru o imagine de dimensiuni mari, cum ar fi una de 1920x1080 pixeli, numărul enorm de operații necesare poate deveni problematic. Chiar și cu procesarea în paralel oferită de procesorul grafic, latenta introdusă de un volum atât de mare de calcule poate fi semnificativă. Având în vedere că coordonatele pixelilor rămân constante în timpul procesării, este justificat să precalculez indicii de pixeli pentru a evita necesitatea recalculării lor în mod repetat. Prin urmare, problema se reduce din nou la o simplă remapare a pixelilor folosind vectori de indici precalculate.

Așadar în procesul de preprocesare a imaginilor pentru rețea de segmentare, este necesar să mapăm pixelii de două ori, o dată pentru a elimina distorsiunile cauzate de lentilă și apoi din nou pentru a ajusta perspectiva imaginii. Cu toate acestea, deoarece aceste două operații sunt similare, există posibilitatea de a le combina sau de a le optimiza. Acest lucru devine posibil prin agregarea celor două seturi de vectori de mapare într-un singur set. Astfel, utilizând indicii din vectorii de mapare pentru schimbarea de perspectivă, vectorii pentru calibrarea camerei la rândul lor pot fi mapati obținând o pereche de vectori de mapare care trebuie calculați doar o singură dată și apoi încărcăți în memorie. Acești vectori de mapare pot fi apoi utilizați repetativ, conducând la o optimizare mai bună a procesului. Această abordare reduce numărul total de operații de mapare necesare și, implicit, timpul și resursele necesare pentru preprocesarea imaginilor.

În ceea ce privește detectarea obstacolelor, aceasta este realizată cu ajutorul rețelei de segmentare semantică, așa cum a fost menționat anterior. Utilizarea perspectivei bird's eye view în acest context aduce beneficii suplimentare. Aceasta permite o evaluare mai ușoară a distanțelor relative dintre obstacole și alte elemente ale drumului. Prin maparea lungimii de pixeli din imagine la distanțele reale, este posibil să se extragă distanțele aproximative până la obstacole. Totuși, o astfel de mapare nu este necesară, deoarece lucrul direct în pixeli poate fi suficient pentru algoritmi și este o abordare validă în acest context.

### 2.3.1.2. Modulul de planificare

Modulul de planificare este, în esență, creierul vehiculului, responsabil pentru luarea deciziilor. Comportamentul vehiculului este dictat prin intermediul unei mașini de stări, așa cum se poate vedea în figura 2.4. Această mașină de stări gestionează tranzițiile între diferitele acțiuni ale vehiculului, cum ar fi oprirea, așteptarea, deplasarea și parcarea, în funcție de informațiile primite de la modulul de percepție.

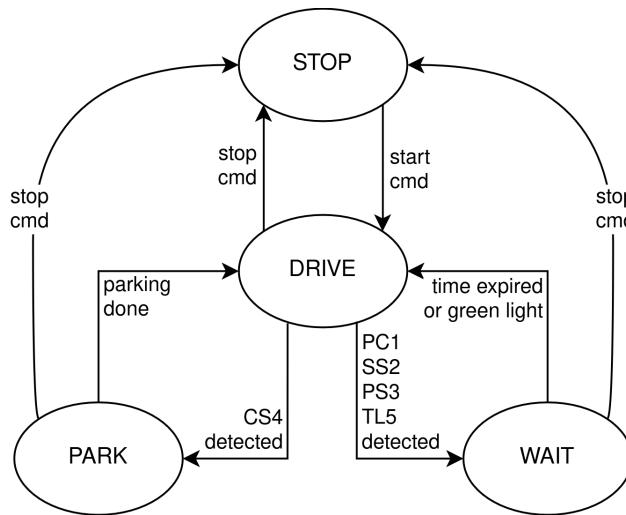


Figura 2.4. Mașina de stări decizionale.<sup>5</sup>

Rolul acestui modul este de a genera traectoria optimă a drumului de mers, asigurându-se că vehiculul urmează un traseu sigur și eficient. Traекторia este determinată în principal pe baza coordonatelor de delimitare a benzii, dar și influențată de alte informații furnizate de modulul de percepție, cum ar fi coordonatele obstacolului și coordonatele stației de încărcare.

Pe lângă generarea traectoriei drumului, modulul de planificare determină și comportamentul vehiculului în urma detectării semnelor de circulație și a semaforului după cum a fost menționat în secțiunea 1.3.

De asemenea, modulul de planificare este responsabil pentru generarea traectoriei necesare efectuării parcării în stația de încărcare. Aceasta implică un proces predefinit, care urmărește punctul cel mai din dreapta al măștii de segmentare ce corespunde stației de încărcare.

Modulul de planificare furnizează modulului de control punctele de referință necesare pentru controlul longitudinal și lateral al vehiculului. Controlul longitudinal implică gestionarea accelerării și frânării pentru a menține o viteza adecvată și constantă, în timp ce controlul lateral se ocupă de ajustarea direcției pentru a urma traectoria planificată.

### 2.3.1.3. Modulul de control

Modulul de control al vehiculului autonom preia datele de la modulul de planificare, incluzând comenzi de viteza și punctul de referință pentru controlul lateral. Aceste date sunt transmise printr-o conexiune socket UDP către controlerul mașinii din Unity. Inițial, modulul de control trebuia să implementeze două controlere PID pentru controlul longitudinal și lateral, însă din cauza constrângerilor de timp, s-a decis să se renunțe la această abordare, rămânând astfel doar cu transmiterea comenziilor de referință către Unity.

<sup>5</sup>Imagine creata cu ajutorul aplicației web [37]

### 2.3.2. Antrenarea rețelelor neuronale

Fiabilitatea rețelelor neuronale este crucială pentru funcționarea corectă a mașinii. Antrenarea adecvată a acestor rețele joacă un rol esențial, deoarece performanța lor influențează direct capacitatea vehiculului de a percep și interpreta corect mediul înconjurător. În această secțiune, se explorează procesul de antrenare a celor două rețele utilizate în modulul de percepție, abordând aspecte precum seturile de date folosite, tehnologiile utilizate și performanțele obținute în urma antrenării acestora.

Antrenarea celor două rețele neuronale, de detecție și segmentare, urmează principii similare, implicând achiziția setului de date și utilizarea unor librării și platforme pentru antrenare. Totuși, cele două rețele au fost antrenate în medii diferite, reflectând nevoile specifice și avantajele fiecărei platforme.

#### 2.3.2.1. YOLOv3

Pentru antrenarea rețelei YOLOv3-tiny, s-a optat pentru utilizarea platformei Google Colaboratory[5], un serviciu cloud oferit de Google care permite rularea notebook-urilor Jupyter în browser. Colab oferă acces gratuit la resurse GPU și TPU, ceea ce îl face extrem de convenabil pentru sarcinile de învățare profundă. Platforma suportă extensiv limbajul Python și librăriile de învățare profundă, precum TensorFlow și PyTorch, facilitând astfel procesul de antrenare și optimizare a rețelei. De asemenea, Google Colab beneficiază de o multitudine de ghiduri și tutoriale specifice, care simplifică implementarea și adaptarea codului pentru diverse proiecte. În acest context, caietul Colab utilizat pentru antrenarea rețelei YOLOv3-tiny este în mare parte bazat pe tutorialul oferit de DepthAI[47]. Acest tutorial furnizează toate resursele și pașii necesari pentru antrenarea rețelei pe un set de date personalizat. Deși au fost făcute unele modificări minore pentru a se adapta nevoilor specifice ale proiectului, esența și structura principală a notebook-ului rămân fidèle ghidului original.

Setul de date pentru rețea de detecție cuprinde șapte clase diferite, inclusiv patru semne de circulație și trei stări ale unui semafor. Imaginele sunt distribuite relativ egal între clase, cu aproximativ 550 de imagini pentru fiecare semn și 700 de imagini pentru fiecare stare a semaforului. Multe dintre aceste imagini conțin multiple clase, nu doar o singură clasă, pentru a îmbunătăți generalizarea modelului. Chiar dacă vehiculul va vedea în mod obișnuit un singur semn sau semafor la un moment dat, prezența mai multor clase în imagini ajută la antrenarea rețelei pentru a recunoaște că mai bine. Aceasta este compus din imagini din viață reală preluate de pe Google și alte surse online, imagini generate cu Blender în diverse condiții de iluminare și ipostaze, precum și imagini capturate direct din simulator, exact așa cum le vede vehiculul autonom. Proportia imaginilor din setul de date este distribuită astfel, aproximativ 30% dintre imagini pentru semnele de circulație provin din viață reală, în timp ce 70% sunt imagini artificiale, generate fie cu Blender, fie capturate din simulator. În cazul semafoarelor, aproximativ 55% dintre imagini provin din viață reală, iar restul de 45% sunt imagini simulate.

Etichetarea imaginilor a fost realizată folosind unealta LabelImg[48], figura 2.5, un instrument open source utilizat pentru a adăuga manual etichete obiectelor din imagini. LabelImg permite utilizatorilor să deseneze casete de delimitare (bounding boxes) în jurul obiectelor de interes și să le eticheteze corespunzător, specificând clasa fiecărui obiect.



Figura 2.5. Procesul de etichetare a imaginilor

Etichetele rezultate sunt fișiere text în care fiecare clasă din imagine este reprezentată pe o linie nouă. Fiecare linie conține mai multe coloane în următorul format: id clasă, centru x al casetei de delimitare, centru y al casetei de delimitare, lățime, înălțime. Valorile sunt normalizează între 0 și 1, raportate la dimensiunile imaginii, care în acest caz sunt de 320x320 pixeli. Eticheta rezultată pentru imaginea din figura 2.5 este prezentată mai jos.

```
5 0.211667 0.448333 0.250000 0.776667
6 0.678333 0.453333 0.263333 0.820000
```

Pentru antrenarea propriu-zisă, setul de date a fost împărțit în două subseturi alese aleatoriu: 80% dintre imagini au fost utilizate pentru antrenare, iar restul de 20% pentru validare. Nu a fost utilizat un subset de test separat în această etapă, deoarece testarea finală a modelului va fi realizată în simulare, unde performanța sa va putea fi evaluată în condiții reale de funcționare.

În graficul prezentat mai jos, se poate observa evoluția pierderii (loss) și a acurateței (accuracy) pe parcursul procesului de antrenare. Întregul proces a durat 50 de minute, iar parametrii rețelei au fost inițializați cu cei ai unei variante deja antrenate, ceea ce, fără îndoială, a contribuit la atingerea rapidă a unei acurateți de 90%, alături de puterea de procesare oferită de Google Colab.

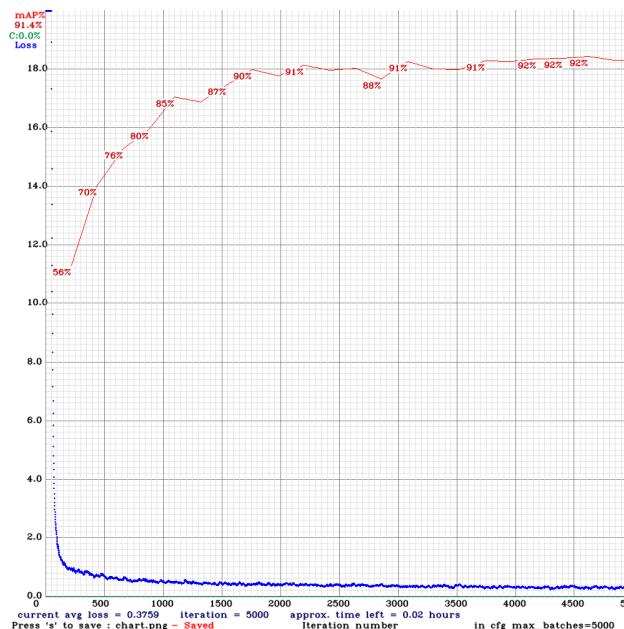


Figura 2.6. Grafic metrii de antrenare

În contextul învățării automate și al rețelelor neuronale, loss-ul sau funcția de pierdere este o metrică folosită pentru a evalua cât de bine funcționează un model. Aceasta măsoară diferența

dintre predicțiile modelului și valorile reale din datele de antrenament. În esență, pierderea reprezintă o măsură a erorii modelului. Când valoarea pierderii începe să se stabilizeze și să nu se mai schimbe semnificativ, procesul de antrenare este aproape de finalizare. Totuși, o valoare prea mică a pierderii poate indica overfitting, adică modelul s-a adaptat prea bine pe datele de antrenament și poate performa slab pe date noi.

În cazul rețelei de detecție, graficul sugerează că modelul YOLOv3-tiny este bine antrenat, având o pierdere minimă și o precizie ridicată.

În final, performanța rețelei de detecție, măsurată prin acuratețea pe subsetul de test, poate fi văzută mai jos pentru fiecare clasă și în ansamblu. TP (True Positives) reprezintă numărul de detecții corecte ale unei clase, iar FP (False Positives) indică numărul de detecții incorecte, unde rețeaua a identificat un obiect inexistent sau a clasificat greșit un obiect.

```
class_id = 0, name = stop, ap = 99.52%          (TP = 94, FP = 4)
class_id = 1, name = parking, ap = 94.41%        (TP = 104, FP = 4)
class_id = 2, name = charging, ap = 99.17%       (TP = 120, FP = 0)
class_id = 3, name = cross, ap = 95.68%          (TP = 100, FP = 4)
class_id = 4, name = yellow light, ap = 89.75%   (TP = 168, FP = 22)
class_id = 5, name = red light, ap = 78.69%       (TP = 170, FP = 53)
class_id = 6, name = green light, ap = 82.71%     (TP = 160, FP = 25)
mean average precision (mAP@0.50) = 0.914178, or 91.42 %
```

Rețeaua de detecție a demonstrat performanțe excelente pentru majoritatea claselor, în special pentru clasele *stop* și *charging*. În cazul stârilor semaforului, rețeaua nu a avut rezultate la fel de bune. Aceasta se datorează în mare parte calității inferioare a setului de date pentru aceste clase, precum și similarităților vizuale între luminile roșii și galbene, care fac ca detecția să fie mai provocatoare.

### 2.3.2.2. FastSCNN

În procesul de antrenare a modelului de segmentare, au fost întâmpinate diverse provocări care au condus la învățarea unor lecții importante. Inițial, a fost utilizat depozitul SegmenTron pentru antrenare, dar segmentarea rezultată a fost nesatisfăcătoare, modelul nereușind să genereze confundând benzile stânga și dreapta. Aceasta a indicat o problemă potențială legată de arhitectura rețelei.

Pentru a îmbunătăți performanța, s-a decis schimbarea depozitului cu PaddleSeg, datorită varietății de rețele și facilităților oferite. Deși a fost testată și rețeaua pp-liteseg, problema confuziei între benzile stânga și dreapta a persistat. În urma unei analize mai atente a fișierului de configurare al antrenării, s-a descoperit că setarea *horizontal-flip* era activată. Acest lucru a generat o confuzie în antrenare, deoarece imaginile erau inversate aleatoriu, ceea ce ducea la interpretări eronate ale direcțiilor benzilor. După dezactivarea acestei setări, s-a observat o îmbunătățire semnificativă a performanței modelelor, care au început să segmenteze corect benzile de circulație.

În final, s-a optat pentru PaddleSeg, datorită ușurinței de utilizare pe care o oferea. Această descoperire simplă a fost crucială pentru îmbunătățirea performanței modelelor. Deși s-a pierdut mult timp din cauza unei probleme aparent minore, aceasta a demonstrat importanța înțelegerii detaliate a modului de funcționare a instrumentelor utilizate, nu doar a folosirii lor conform instrucțiunilor găsite pe internet. În continuare, va fi prezentat procesul de achiziționare a setului de date pentru rețeaua de segmentare.

Setul de date utilizat pentru antrenarea rețelei de segmentare semantică constă din imagini capturate din perspectiva bird's eye view. Aceste imagini sunt etichetate în şase clase distincte: fundal, banda din dreapta, banda din stânga, maraj alb, stație de încărcare și obstacol. Fiecare imagine are o rezoluție de 1024x512 pixeli, asigurând o calitate suficient de înaltă pentru a capta

detaliile relevante necesare unei segmentări precise, dar și nu prea mare, pentru a putea rula pe Jetson la o performanță acceptabilă.

Setul de date este compus în proporție de 40% din imagini din lumea reală, capturate în diverse medii și condiții de iluminare, și 60% din imagini sintetice generate digital în Blender și Unity, însumând un total de 3866 imagini din care 850 au fost generate în simulator. Imaginele reale au fost incluse în setul de date deoarece au fost segmentate în timpul concursului și s-au dovedit utile pentru îmbunătățirea generalizării modelului.

Procesul de etichetare în cazul segmentării este diferit și poate fi extrem de laborios. Etichetele sunt sub forma unor măști care reprezintă obiectele și clasele din imagine. Fiecare pixel din imagine trebuie asociat cu o anumită clasă, ceea ce implică trasarea manuală a contururilor exacte ale fiecărui obiect relevant. În funcție de tipul măștii, pixelii sunt etichetați fie printr-o valoare de gri, dacă masca este în format gri(grayscale), fie prin culori diferite, dacă masca este în format RGB. Deși multe librării de segmentare semantică suportă ambele formate, în timpul antrenării, măștile sunt adesea convertite în imagini gri, unde fiecare valoare de pixel reprezintă o clasă diferită. Această conversie simplifică procesul de antrenare și este suportată de majoritatea librăriilor de segmentare.

În comparație cu etichetarea pentru detectarea obiectelor, unde casetele de delimitare sunt suficiente, procesul de segmentare este mult mai detaliat și consumator de timp. Cu toate acestea, precizia etichetării este esențială pentru antrenarea eficientă a rețelelor de segmentare semantică, asigurând performanțe optime ale modelului.

Etichetarea imaginilor generate se realizează simultan cu generarea acestora. BlenderProc[49] este un pipeline modular pentru Blender, care permite crearea automată de perechi imagine-mască. Pentru imaginile generate în Unity, a fost utilizat pachetul Perception[50], care facilitează de asemenea generarea acestor perechi. Acest lucru este un avantaj major comparativ cu imaginile din lumea reală, care necesită etichetare manuală pentru segmentare.

Unealta software utilizat pentru etichetarea imaginilor din lumea reală este CVAT (Computer Vision Annotation Tool), dezvoltat de echipa OpenCV. CVAT este un instrument open-source versatil, proiectat pentru a facilita procesul de etichetare în diverse domenii ale vederii computaționale. Acesta oferă suport extins pentru segmentare, detecție de obiecte și alte sarcini de etichetare, permitând utilizatorilor să creeze cu ușurință măști precise și casete de delimitare pentru antrenarea rețelelor neuronale. CVAT vine cu o interfață intuitivă și numeroase facilități, cum ar fi suportul pentru colaborare în echipă, integrarea cu diverse platforme de stocare și posibilitatea de a gestiona proiecte mari de etichetare. Datorită acestor caracteristici, CVAT s-a dovedit a fi un instrument esențial în pregătirea setului de date pentru rețea de segmentare semantică.

Procesul de segmentare a imaginilor, aşa cum este ilustrat în figura 2.7, implică desenarea de poligoane pentru fiecare clasă, cu excepția clasei de fundal care este etichetată automat de CVAT. Fiecare poligon este desenat manual pentru a delimita cât mai bine contururile obiectelor și pentru a asigura o clasificare corectă a fiecărui pixel.

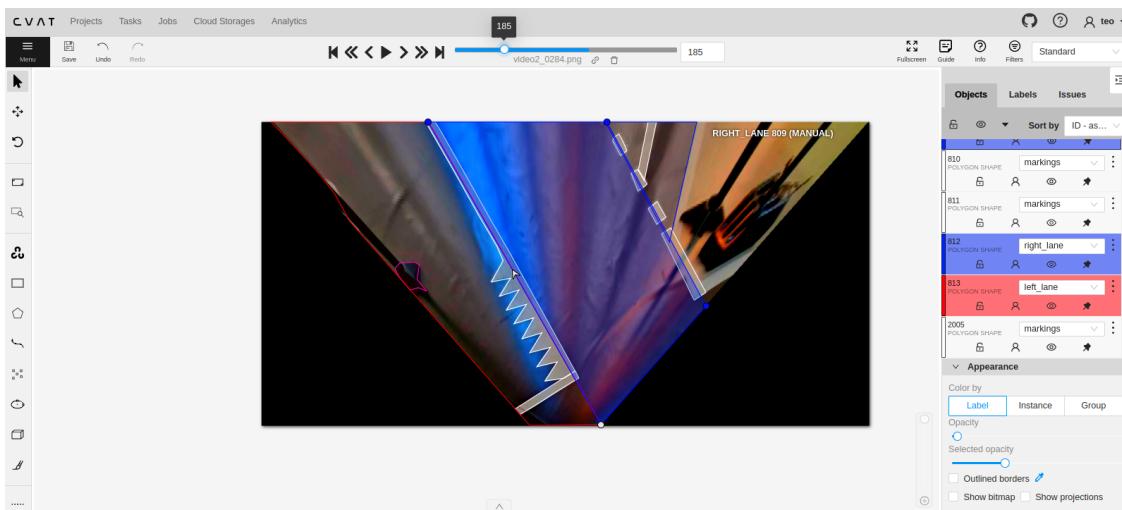


Figura 2.7. Procesul de segmentare a imaginilor

Un exemplu de pereche imagine-mască rezultată poate fi văzut în figura 2.8. În această figură, masca este reprezentată în format RGB, ceea ce permite o vizualizare clară a diferențelor dintre clase. Fiecare culoare din masca RGB corespunde unei clase specifice, facilitând astfel identificarea vizuală a diferitelor elemente din imagine. Acest format este deosebit de util în etapa de validare a setului de date, unde corectitudinea etichetării poate fi verificată rapid prin simple inspecții vizuale.



Figura 2.8. Pereche de imagine și masca sa

Graficele din timpul antrenării sunt prezentate mai jos și includ trei reprezentări esențiale pentru evaluarea și monitorizarea performanței modelului. Figura 2.9 ilustrează evoluția măsurii mIoU (mean Intersection over Union), care indică precizia generală a segmentării prin măsurarea suprapunerii dintre predicțiile modelului și etichetele reale pentru subsetul de validare. Graficul din figura 2.10 prezintă evoluția acurateței modelului pe parcursul antrenării, arătând procentul de pixeli corect clasificați. Ultimul grafic din figura 2.11, dedicat etapei de antrenare, arată evoluția pierderii (loss), evidențiind modul în care modelul se îmbunătățește în timp. Aceste grafice sunt cruciale pentru a înțelege performanța modelului și pentru a identifica posibilele probleme în procesul de antrenare.

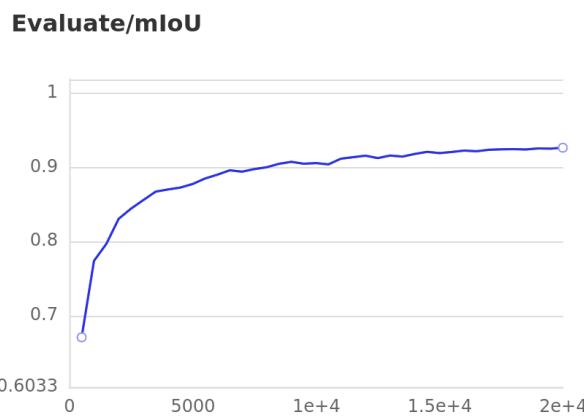


Figura 2.9. Evoluție mIoU segmentare

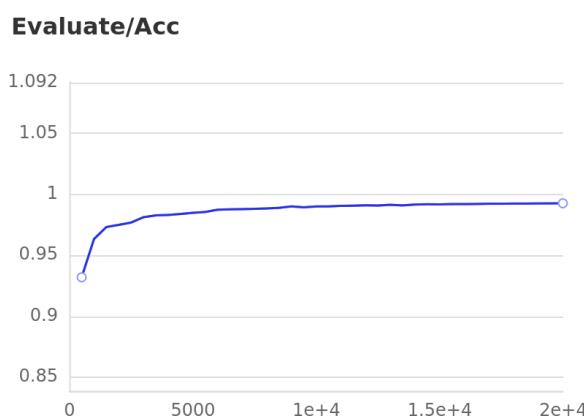


Figura 2.10. Evoluție acuratețe segmentare

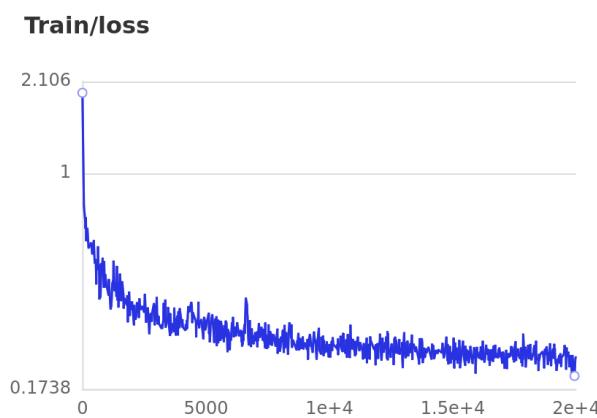


Figura 2.11. Evoluție pierdere

În cele din urmă, metricele de performanță obținute pe setul de date de validare sunt prezentate mai jos:

```
[EVAL] Class IoU:  
[0.9954 0.9755 0.9765 0.8658 0.8873 0.8563]  
[EVAL] Class Precision:  
[0.9979 0.9871 0.9879 0.9277 0.9302 0.9094]
```

```
[EVAL] Class Recall:  
[0.9975 0.9881 0.9883 0.9284 0.9505 0.9362]  
EVAL] The model with the best validation mIoU (0.9261) was saved at iter 20000.
```

Aceste rezultate indică faptul că rețeaua a atins un nivel înalt de acuratețe și fiabilitate.

Precizia indică numărul de predicții corecte ale clasei per pixel dintre toate predicțiile pozitive, măsurând procentajul pixelilor clasificați corect. Cu alte cuvinte, precizia poate fi văzută ca o metrică de măsurare a calității predicțiilor pozitive, reflectând acuratețea clasificării pentru fiecare pixel.

Rata de reamintire (recall), pe de altă parte, măsoară capacitatea modelului de a detecta toate instanțele pozitive din setul de date. Aceasta metrică indică proporția pixelilor corect clasificați din totalul pixelilor care ar trebui să fie clasificați într-o anumită clasă, evidențiind sensibilitatea modelului.

Concluzionând, se poate observa că acuratețea generală a modelului este foarte mare, demonstrând capacitatea sa de a segmenta cu succes fiecare clasă. Valorile ridicate ale IoU, preciziei și a reamintirii pentru fiecare clasă indică faptul că modelul reușește să identifice și să segmenteze corect elementele din imagini.



## Capitolul 3. Implementarea aplicației

Acest capitol prezintă în detaliu implementarea aplicației, structura proiectului, precum și componentele principale ale sistemului. Scopul este de a oferi o privire cuprinzătoare asupra întregului proces de dezvoltare, de la concepție până la implementare și optimizare.

Pentru a ilustra modul în care proiectul este organizat, va fi prezentat în continuare o diagramă tip arbore al directorului principal al proiectului. Aceasta include detalii despre subdirectorile și fișierele relevante, evidențiind rolul acestora în cadrul proiectului. Figura 3.1 prezintă structura directorului proiectului, subliniind principalele componente și organizarea acestora.

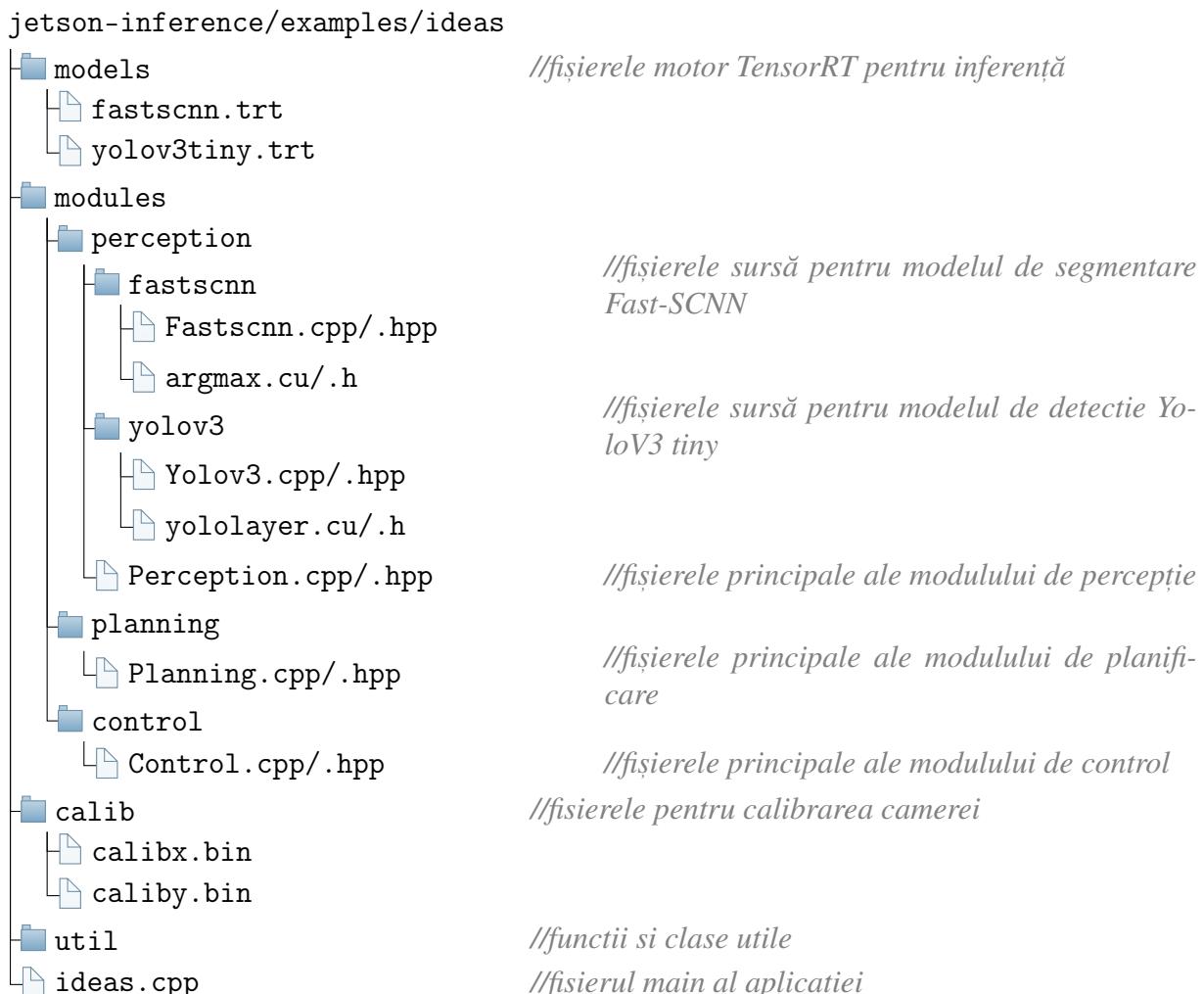


Figura 3.1. Structura director proiect

Proiectul urmează un model de programare orientată pe obiecte, în care fiecare modul este reprezentat printr-o clasă. În fișierul main, aceste clase sunt inițializate și procesate secvențial într-o singură buclă while.

Fișierul main este relativ simplu în structura sa, acesta se ocupă de inițializarea modulelor, capturarea și procesarea imaginilor, și gestionarea ciclului principal de execuție. Funcția sa principală este de a coordona funcționarea diferitelor module ale aplicației, asigurându-se că fiecare modul este corect inițializat și apelat secvențial într-o buclă while. În cele ce urmează, va fi prezentata pe scurt structura și funcționalitatea acestui fișier, subliniind etapele esențiale ale execuției aplicației.

La începutul fișierului, este definit un handler de semnal pentru a permite terminarea controlată a aplicației în cazul în care este primit semnalul SIGINT. Acest handler setează un flag *signal\_received* care va fi verificat în bucla principală de execuție.

Funcția main începe cu parsarea argumentelor de linie de comandă și alocarea memoriei pentru imaginea de vizualizare în zona zero-copy a Jetsonului, ceea ce permite accesarea directă a memoriei de către CPU și GPU fără a fi nevoie de operațiuni suplimentare de copiere. Această metodă îmbunătățește performanța aplicației atunci când vizualizarea este activată.

```
1  /* parsare linie de comanda */
2  commandLine cmdLine(argc, argv);
3
4 #if VISUALIZATION_ENABLED
5  /* allocare spatiu imagine de vizualizare */
6  if (!cudaAllocMapped(&imgOutput, make_int2(VIS_WINDOW_W,
7      → VIS_WINDOW_H)))
7  {
8      LogError("Ideas: Failed to allocate CUDA memory for out
    → image\n");
9  return 1;
10 }
11#endif
```

Listing 3.1. Parsare argumentelor de linie de comanda și alocarea memoriei pentru imaginea de vizualizare

După aceasta, se atașează handler-ul de semnal pentru SIGINT, permitând o terminare controlată a aplicației atunci când utilizatorul întrește execuția.

Urmează crearea stream-urilor de input și, optional, de output pentru capturarea și redarea imaginilor, respectiv. Aceste stream-uri sunt furnizate de Jetson Inference și sunt esențiale pentru interacțiunea cu aplicația. Stream-ul de input preia datele de intrare, care pot proveni de la diferite surse precum camerele video conectate la Jetson, sau adresele RTSP pentru fluxurile video de la distanță. Acest lucru conferă aplicației o mare modularitate și flexibilitate, deoarece utilizatorul poate specifica sursa de date printr-un protocol specificat de linia de comandă, fără a fi nevoie de modificări în codul sursă al aplicației.

```
1  /* creare flux intrare */
2  videoSource *input = videoSource::Create(cmdLine, ARG_POSITION(0));
```

Listing 3.2. Creare flux video de intrare

De asemenea, stream-ul de output este utilizat în stagiul de dezvoltare și permite vizualizarea rezultatelor procesării. Acesta poate reda imaginile procesate direct pe un ecran conectat la Jetson, pe un alt computer prin RTSP, WebRTC sau poate salva rezultatele într-un fișier video, în funcție de preferințele utilizatorului.

```
1 #if VISUALIZATION_ENABLED
2  /* creare flux iesire */
3  videoOutput *output = videoOutput::Create(cmdLine,
    → ARG_POSITION(1));
4#endif
```

Listing 3.3. Creare flux video de iesire

Această abordare modulară și ușor de configurat aduce beneficii semnificative în procesele de dezvoltare, testare și întreținere a aplicației. Utilizând facilitățile oferite de depozitul Jetson

Inference, cum ar fi captarea rapidă și eficientă a imaginilor, se permite o adaptabilitate sporită a surselor de date și a modurilor de redare. Astfel, schimbările sau actualizările ulterioare pot fi implementate fără dificultăți majore, contribuind la flexibilitatea și scalabilitatea aplicației. Comparativ cu dezvoltarea unei soluții personalizate bazate pe TensorRT de la zero, utilizarea Jetson Inference reduce semnificativ timpul și efortul necesar pentru implementarea funcționalităților de achiziționare a imaginilor, facilitând în același timp menținerea și îmbunătățirea continuă a aplicației.

Inițializarea modulelor se realizează odată cu instanțierea acestora prin intermediul constructorilor. În cadrul acestora fiecare modul își inițializează componentele și variabilele necesare.

```

1  /* Initializare prin constructori */
2  Perception PerceptionModule;
3  Planning PlanningModule;
4  Control ControlModule;
```

**Listing 3.4. Inițializarea modulelor prin apelarea constructorilor**

În bucla principală de execuție, imaginile sunt achiziționate de la fluxul de intrare video și procesate de modulul de percepție. Apoi, rezultatele procesării sunt preluate de către modulul de planificare, unde se calculează viteza dorita și poziția dorita pe banda de drum a mașinii. În continuare, rezultatele modulului de planificare sunt preluate de către modulul de control, care transmite comenzi de acționare a direcției și vitezei mașinii printr-o conexiune socket UDP către controlerul mașinii din simulator.

```

1  PerceptionModule.Process(imgInput, imgOutput); //Inferenta retele
2  PlanningModule.Process(&PerceptionModule);      //Procesare masina de
   → stari
3  ControlModule.Process(&PlanningModule);          //Generare comenzi de
   → control si transmisie
```

**Listing 3.5. Procesarea secvențială**

În continuare, se va explora în detaliu fiecare modul al proiectului, explicând funcționalitatea și interacțiunea dintre acestea.

Modulul de percepție, compus din fișierele *Perception.cpp*, *Perception.hpp* și fișierele sursă a celor două rețele, reprezintă o componentă esențială a aplicației. Aceasta este responsabil pentru gestionarea și procesarea imaginilor captureate prin inferența celor două rețele neuronale.

Clasa *Perception* este nucleul modulului de percepție și oferă o interfață pentru rularea celor două rețele neuronale. Listingul 3.6 oferă o privire generală asupra clasei *Perception* și funcțiile principale ale acesteia, codul complet poate fi găsit la anexa 1.

```

1  class Perception
2  {
3      public:
4      /* Constructor modul perceptie */
5      Perception() : seg_network(segModelPath), det_network(detModelPath)
6      {
7          ...
8          InitNetworks();
9      };
10     /* Functie procesare modul */
11     int Process(pixelType *imgInput, pixelType **imgOutput);
12     /* Functii de interfata */
```

```

13     ...
14
15     private:
16     /* Obiectele retelelor neuronale */
17     FastScnn seg_network;
18     YoloV3 det_network;
19     ...
20     /* Functie initializare retele */
21     int InitNetworks();
22 };

```

Listing 3.6. Clasa Perception

Constructorul clasei Perception apelează constructorii celor două rețele, `seg_network` și `det_network`, inițializându-le cu căile către fișierele motor, iar apoi apelează funcția `InitNetworks`.

Constructorii celor două rețele inițializează variabilele interne și deserializarea motorului TensorRT dintr-un fișier binar stocat pe disc. Aceasta implică citirea datelor de pe disc, crearea unui runtime TensorRT și folosirea acestuia pentru a de-serializa motorul de inferență. De asemenea acest constructor se ocupa cu inițializarea a diferite date despre care se vor discuta în curând.

Funcția `InitNetworks` este o funcție wrapper pentru funcțiile proprii de inițializare a celor două rețele.

```

1  /* Initializare retea de segmentare */
2  int status_seg = seg_network.InitEngine(); //creare context executie,
   ↳ alocare memorie input/output
3  ...
4  /* Initializare retea de detectie */
5  int status_det = det_network.InitEngine(); //creare context executie,
   ↳ alocare memorie input/output
6  ...

```

Listing 3.7. Prezentare generală a funcției `InitNetworks`

Cele două funcții de inițializare a rețelelor TensorRT, utilizate în modulul de percepție, urmează un flux de lucru standard pentru configurarea și pregătirea rețelelor neuronale pentru inferență. Acestea sunt responsabile de crearea și pregătirea contextului de execuție, precum și de configurarea și alocarea bufferelor pentru intrări și ieșiri în cadrul rețelei, codul complet pentru acestea poate fi văzut în anexa 4.

Funcția `Process` reprezintă componenta centrală a modulului de percepție, fiind responsabilă pentru procesarea imaginilor de intrare și generarea rezultatelor necesare altor module din sistem. Aceasta primește ca parametru o imagine de intrare și utilizează rețelele de segmentare și detectie pentru a analiza conținutul imaginii. După ce prelucrează imaginea, funcția generează datele de ieșire, obiectele detectate, harta de segmentare precum și alte date derivate din harta de segmentare.

```

1  /* Calibrare imagine, aplicare algoritm bev */
2  seg_network.PreProcess(imgInput);
3  /* Inferenta model */
4  seg_network.Process();
5  /* Aplicare algoritm argmax */
6  seg_network.PostProcess(segmap_ptr, left_lane_x_limits,
   ↳ right_lane_x_limits, charging_pad_limits, obstacle_limits);
7  ...

```

```

8 // Vizualizare harta de segmentare
9 ...
10 /* Calibrare imagine, extragere regiune de interes */
11 det_network.PreProcess(imgInput);
12 /* Inferenta model */
13 det_network.Process();
14 /* Aplicare algoritm non-maximum suppression */
15 det_network.PostProcess(&detctions);
16 ...
17 // Vizualizare casete de delimitare obiecte detectate
18 ...

```

Listing 3.8. Prezentare generală a funcției Process

Funcțiile de preprocessare a celor două modele au rolul de a procesa imaginea pentru a o pregăti pentru inferență, ajustând dimensiunile, normalizând valorile pixelilor sau eliminând distorsiunile cauzate de lentila în funcție de cerințele specifice fiecărei rețele.

În cazul rețelei de segmentare funcția de preprocessare transformă imaginea în vedere de sus, normalizând în același timp valorile pixelilor. Metoda utilizată pentru generarea vectorilor de mapare nu se bazează pe metoda clasica a utilizării unei matrice de omografie prezentată în secțiunea 2.3.1.1, ci pe o abordare mai complexă care ia în considerare parametrii intrinseci și extrinseci ai camerei. Această metodă oferă mai multă flexibilitate în configurarea imaginii Bird's Eye View (BEV) și se potrivește mai bine cerințelor proiectului.

Procesul de generare a vectorilor de mapare a fost realizat cu ajutorul depozitului *IP-MandCamCalib*[51], inspirat inițial din lucrarea lui Liao, James disponibilă în depozitul său *IPM-Master*[46]. Acest depozit nu se limitează doar la generarea vectorilor de mapare pentru IPM, ci oferă și unelte pentru calibrarea camerei și agregarea acestora pentru a obține un singur set de vectori de mapare.

Algoritmul de mapare, aşa cum a fost menționat anterior, utilizează facilitățile CUDA și poate fi găsit în Anexa 3.

Funcția de preprocessare a rețelei de detectie se ocupa cu normalizarea pixelilor și extragerea zonei de interes din imaginea mare după cum a fost menționat în secțiunea 2.3.1.1.

Funcțiile *Process* ale celor două rețele realizează inferență, utilizând datele de intrare preprocesate pentru a genera date brute. În cazul rețelei de detectie, aceste date brute sunt reprezentate de casetele de delimitare, iar în cazul rețelei de segmentare, ieșirea constă în scorurile de încredere pentru fiecare clasă, aplicate fiecărui pixel.

Funcțiile de postprocesare preiau datele brute generate de cele două rețele și le transformă în formatul utilizabil necesar. În cazul rețelei de segmentare, ieșirea inferenței are dimensiunile *nr\_clase x dim\_h x dim\_w*, adică în acest caz  $6 \times 512 \times 1024$ . Fiecare strat de dimensiune *id\_clasă x 512 x 1024* conține scorurile de încredere pentru care rețeaua consideră că pixelii respectivi aparțin clasei *id\_clasă*. Pentru a obține harta de segmentare finală, se compară aceste scoruri de încredere pentru fiecare pixel și se alege valoarea cea mai mare, proces cunoscut sub numele de *argmax*.

Pe lângă aplicarea algoritmului *argmax*, funcția de postprocesare calculează punctele de delimitare pentru banda din stânga, din dreapta, stație și obstacol. Aceste puncte de delimitare în cazul stației și a obstacolului reprezintă patru puncte care deservesc marginile acestora, punctul cel mai din stânga, cel mai din dreapta, cel mai de sus și cel mai de jos, obținându-se o idee generală asupra dimensiunii și a poziției în imagine a acestora. În cazul celor două benzi punctele de delimitare reprezintă cele mai din stânga și cele mai din dreapta puncte care aparțin benzii la diferite înălțimi, adică de-a lungul benzii ca în figura. Aceste date sunt utilizate mai departe de modulul de planificare pentru a usura procesul de planificare.

Pasul de post-procesare în cazul rețelei de detectie constă în aplicarea algoritmului de suprimare a non-maximelor (NMS). Datorită naturii arhitecturii rețelei de detectie, care utilizează

ancore pentru a acoperi diferite regiuni ale imaginii, rezultatul inferenței poate include sute de detectări pentru un singur obiect din imagine. Aceste detectări multiple generează suprapunerile și necesită o filtrare eficientă pentru a identifica și să păstreze doar detectările cele mai relevante. Algoritmul NMS ajută la eliminarea suprapunerilor și la păstrarea casetelor de delimitare cu cele mai mari scoruri de încredere, asigurând astfel o detecție precisă și clară a obiectelor.

Algoritmul NMS funcționează prin parcurgerea tuturor casetelor de delimitare generate de rețea și eliminarea celor care se suprapun într-o mare măsură cu altele care au scoruri de încredere mai mari. Practic, NMS păstrează caseta cu cel mai mare scor de încredere și elimină toate celelalte casete care au o suprapunere (calculată prin coeficientul IoU - Intersection over Union) mai mare decât un prag predefinit. Acest proces se repetă până când toate casetele relevante au fost evaluate, rezultând o listă filtrată de detectări precise.

În final se obțin datele procesate și filtrate care pot fi utilizate de către modulul de planificare.

Modulul de planificare este alcătuit din fișierele Planning.cpp și Planning.hpp. După cum a fost menționat și anterior acest modul se ocupa cu comportamentul vehiculului care este gestionat de o mașină de stări. O scurta privire la structura acestui modul poate fi văzută în listingul 3.9.

```

1  class Planning
2  {
3      public:
4          /* Initializare variabile */
5          Planning();
6          /* Functie procesare modul */
7          void Process(Perception *perception_module);
8          ...
9      private:
10         ...
11         /* Calculeaza punctele de centru a celor doua benzi */
12         void GetLaneCenterPoints();
13         /* Calculeaza punctul de referinta pentru controlul lateral */
14         void GetLaneCenter(int lane_idx);
15         /* Proceseaza masina de stare */
16         void RunStateHandler();
17     };

```

Listing 3.9. Clasa Planning<sup>6</sup>

Logica de luare a deciziilor este implementată sub forma unei mașini de stări cu patru stări principale, aşa cum este ilustrat în figura 2.4 STOP, WAIT, DRIVE și PARK. În starea STOP, modulul de planificare menține vehiculul oprit, așteptând comanda de start pentru a trece în starea DRIVE. În starea DRIVE, modulul verifică constant prezența obstacolelor pe bandă. Dacă se detectează un obstacol, traiectoria se modifică temporar pentru a evita obstacolul, trecând între cele două benzi. De asemenea, în această stare, vehiculul monitorizează semnele de circulație și semafoarele din apropiere. În funcție de semnul detectat, vehiculul trece în starea WAIT pentru o perioadă determinată, în cazul indicatoarelor, sau până la schimbarea semaforului la verde.

În cazul detectării semnului de încărcare CS4, vehiculul trece în starea PARK, unde se deplasează încet către stația de încărcare, efectuând parcarea doar din mers înainte. După ce vehiculul este poziționat corect în stație, acesta rămâne în starea WAIT pentru cinci secunde înainte de a reveni în starea DRIVE.

Distanța până la un semn este estimată într-un mod simplu, dar eficient. În principiu, cu cât este mai mare dimensiunea cutiei de delimitare, cu atât semnul este mai aproape de mașină.

<sup>6</sup>codul complet se află în anexa 2

Totuși, dimensiunea pe axa x nu este utilizată pentru această estimare, deoarece semnul ar putea fi vizibil doar parțial, ceea ce ar putea indica, în mod eronat, că semnul este aproape de mașină. În schimb, dimensiunea pe axa y nu suferă de această problemă. Astfel, utilizarea dimensiunii pe axa y oferă o estimare suficient de precisă a distanței până la semn, deoarece dimensiunea aparentă a semnului în imagine crește pe măsură ce mașina se apropie de acesta.

Banda de drum este modelată printr-o serie de puncte care reprezintă mijlocul acesteia la diverse poziții de-a lungul drumului. Aceste puncte sunt calculate de modulul de planificare, utilizând punctele de delimitare a benzii. Funcția responsabilă pentru calcularea și filtrarea punctelor de centru ale benzii se numește GetLaneCenterPoints. Această funcție calculează punctele de mijloc ale benzii și, în cazurile în care distanța dintre punctele de extremitate este prea mică sau prea mare față de distanța așteptată, centrul este estimat pe baza cunoașterii lățimii tipice a benzii.

Următoarea secvență de cod face parte din funcția GetLaneCenterPoints și are rolul de a calcula punctele de centru pentru cazurile în care lățimea benzii pare să fie corectă și pregătește celelalte extremități pentru estimarea centrului lor. Calcularea acestor puncte pentru banda din stânga se realizează într-o manieră similară.

```

1  for (int i = 0; i < IMG_HEIGHT / SAMPLING_RES; i++)
2  {
3      right_lane_x[i] = NO_CENTER_POINT;
4      /* Daca avem extremitati(nu sunt valorile de init) */
5      if (right_lane_x_limits[2 * i] < right_lane_x_limits[2 * i + 1])
6      {
7          int lane_width = right_lane_x_limits[2 * i + 1] -
8              right_lane_x_limits[2 * i];
9          /* Daca largimea benzii are o valoare admisibila */
10         if (lane_width > LANE_WIDTH_BETWEEN_MIN && lane_width <
11             LANE_WIDTH_BETWEEN_MAX)
12         {
13             /* Calculare centru banda la y = i * SAMPLING_RES */
14             right_lane_x[i] = (right_lane_x_limits[2 * i + 1] +
15                 right_lane_x_limits[2 * i]) / 2;
16         }
17         /* Largime banda prea mare */
18         else if (lane_width > LANE_WIDTH_BETWEEN_MAX)
19         {
20             /* Va fi estimat */
21             right_lane_x[i] = LANE_TOO_WIDE;
22         }
23         /* Largime banda prea mica */
24         else if (lane_width < LANE_WIDTH_BETWEEN_MIN)
25         {
26             /* Va fi estimat */
27             right_lane_x[i] = LANE_TOO_NARROW;
28         }
29     }
30 }
```

Listing 3.10. Calculare puncte centru pentru banda din dreapta

În cazul în care lățimea benzii este prea mică față de valoarea așteptată, acest lucru se datorează adesea câmpului de vizualizare în formă de trapez rezultat în urma algoritmului IPM. În această situație, doar unul dintre cele două puncte de delimitare a benzii este corect, iar celălalt este estimat pe baza punctului corect, ajustat cu lățimea benzii, în funcție de poziția benzii, dreapta

sau stânga mașinii. De fapt, această estimare nu este absolut necesară, deoarece mașina ar tinde oricum să se centreze pe bandă, însă ar face-o într-un timp mai îndelungat.

Teoretic, este posibil ca rețeaua de segmentare să clasifice greșit anumite pixeli, ceea ce ar putea duce la calcularea incorectă a centrului benzii. Totuși, această problemă este mai puțin probabilă în partea inferioară a imaginii, unde segmentarea este mai precisă datorită detaliilor superioare. Este important ca segmentarea să fie mai exactă în această zonă, deoarece partea inferioară a imaginii reprezintă secțiunea drumului imediat din fața vehiculului, care este crucială pentru calculele de traiectorie și deciziile de direcție. Această situație nu reprezintă o problemă majoră, deoarece erorile sunt rare, iar pentru a afecta semnificativ traiectoria, rețeaua ar trebui să greșească clasificarea pe mai multe cadre consecutive, ceea ce este foarte puțin probabil. Rețeaua segmentează majoritatea cadrelor în mod fiabil. Următoarea secvență de cod realizează estimarea propriu-zisă.

```

1  for (int i = 0; i < IMG_HEIGHT / SAMPLING_RES; i++)
2  {
3      /* cel mai des doar unul din punctele de delimitare e eronat */
4      if (right_lane_x[i] == LANE_TOO_WIDE)
5      {
6          /* Se intampla foarte rar :) */
7      }
8      /* cel mai des este din cauza campului de vizualizare in forma de
9      ↳ trapez */
10     else if (right_lane_x[i] == LANE_TOO_NARROW)
11     {
12         int left_most_x = right_lane_x_limits[2 * i];
13         int right_most_x = right_lane_x_limits[2 * i + 1];
14
15         /* Daca banda se afla in jumatatea stanga */
16         if ((left_most_x + right_most_x) / 2 < IMG_WIDTH / 2)
17         {
18             /* estimare centru pe baza latimii tipice */
19             right_lane_x[i] = (right_most_x + (right_most_x -
20                 LANE_WIDTH)) / 2;
21         }
22         /* Daca banda se afla in jumatatea dreapta */
23         else
24         {
25             /* estimare centru pe baza latimii tipice */
26             right_lane_x[i] = (left_most_x + (left_most_x +
27                 LANE_WIDTH)) / 2;
28         }
29     }
30 }
```

Listing 3.11. Estimare puncte centru pentru banda din dreapta

Mai departe, dintre punctele care modeleză banda, este ales cel pe care mașina trebuie să îl urmărească. Această selecție este realizată de funcția *GetLaneCenter*, care, în funcție de parametru, determină banda pe care vehiculul o va urma. Dintre punctele de mijloc ale benzii respective, se selectează cel mai apropiat punct valid de mașină.

Modulul de control este alcătuit din fișierele *Control.hpp* și *Control.cpp*. Spre deosebire de celelalte două module, acesta este cel mai simplu, ocupându-se în principal de transmiterea comenziilor. Structura acestuia poate fi vizualizată în listingul 3.12.

```

1  class Control
2  {
3      public:
4          Control() : comm(SOCKET_COMM_TYPE)
5          {
6              perf_profiler_ptr = PerfProfiler::getInstance();
7          };
8          void Process(Planning* planning_module);
9      private:
10         int speed_sp;
11         int lateral_sp;
12
13         Comm comm;
14         PerfProfiler* perf_profiler_ptr;
15
16         /* Extragere comenzi */
17         void GetPlanningData(Planning* planning_module);
18         /* Transmisie comenzi */
19         void SendSetpoints();
20     };

```

Listing 3.12. Clasa Control

Constructorul acestei clase inițiază un obiect de tip Comm, care încorporează toate funcțiile necesare pentru transmisia datelor. Funcția Process apelează GetPlanningData pentru a extrage comenzi din modulul de planificare, după care SendSetpoints transmite aceste comenzi către simulator utilizând funcțiile de interfață ale obiectului comm.



## Capitolul 4. Testarea aplicației și rezultate experimentale

În acest capitol, se detaliază procesul de testare a aplicației și se prezintă rezultatele experimentale obținute. Testarea are scopul de a evalua performanțele sistemului dezvoltat, atât din punct de vedere hardware, cât și software. De asemenea se investighează modul de funcționare al aplicației, identificând cazurile în care sistemul se dovedește fiabil sau întâmpină dificultăți.

În primul rând, se descrie procesul de punere în funcțiune și lansare a aplicației, inclusiv detalii despre configurarea și instalarea uneltelor necesare pentru funcționarea acesteia. Ulterior, se analizează metodologia de testare a sistemului, evidențiind aspectele legate de performanța hardware și software, încărcarea procesorului, utilizarea memoriei și limitările în ceea ce privește transmisia video.

Capitolul continuă prin prezentarea datelor de test utilizate, metricilor și benchmark-urilor relevante pentru evaluarea sistemului. Sunt discutate aspectele legate de fiabilitate comparând performanțele sistemului cu obiectivele inițiale.

### 4.1. Elemente de configurare și instalare

#### 4.1.1. Laptop

Pentru testarea aplicației, laptopul pe care rulează simularea necesită diverse aplicații software esențiale pentru a asigura funcționarea corectă. Acestea includ serverul media MediaMTX, care permite publicarea și consumarea fluxurilor video în rețea locală, OBS (Open Broadcaster Software), care permite extragerea fluxului video din simulator și transmisarea acestuia către serverul media și desigur platforma Unity în cadrul căreia rulează simularea. Este important de menționat că laptopul rulează pe sistemul de operare Ubuntu, ales pentru stabilitatea și suportul extins pentru dezvoltare și aplicații de simulare.

##### 4.1.1.1. MediaMTX

Pentru a configura serverul media MediaMTX, primul pas este descărcarea binarului stand-alone de pe GitHub[41], care să corespundă sistemului de operare utilizat. Procesul de instalare este simplu și direct, necesitând doar descărcarea și rularea binarului specific platformei Ubuntu. Odată descărcat, binarul poate fi rulat direct pentru a inițializa serverul media, permitând astfel publicarea și consumarea fluxurilor video în rețea locală.

```

1 $ wget
  ↳ https://github.com/bluenviron/mediamtx/releases/download/v1.6.0/mediamtx_v1.6.0_amd64.tar.gz
  ↳ -O mediamtx.tar.gz
2 $ tar -xf mediamtx.tar.gz

```

Listing 4.1. Instalare MediaMTX

##### 4.1.1.2. OBS

Configurarea OBS pentru a publica fluxul video către serverul media implică mai mulți pași esențiali. Aplicația OBS a fost instalată de pe pagina oficială, optând pentru versiunea 29.1.3 pentru a asigura compatibilitatea cu pluginul DistroAV, versiunea 4.8.0. Instalarea pluginului implică mai mulți pași, care se găsesc detaliați pe pagina depozitului GitHub[40].

Pentru configurarea OBS, se navighează în meniul *Settings*, apoi la rubrica *Output* și se selectează fila *Recording*. Aici, se configurează FFmpeg ca metodă de ieșire și se setează adresa serverului media pentru transmisie. În plus, este necesară adăugarea sursei NDI Source, care

permite preluarea fluxului video. Aceasta se realizează din caseta *Sources* prin apăsarea butonului + și selectarea opțiunii *NDI Source*.

Configurațiile specifice, inclusiv adăugarea sursei și setările pentru transmisie, sunt ilustrate în figurile corespunzătoare figura 4.1 și figura 4.2, oferind o imagine clară a pașilor de urmat.

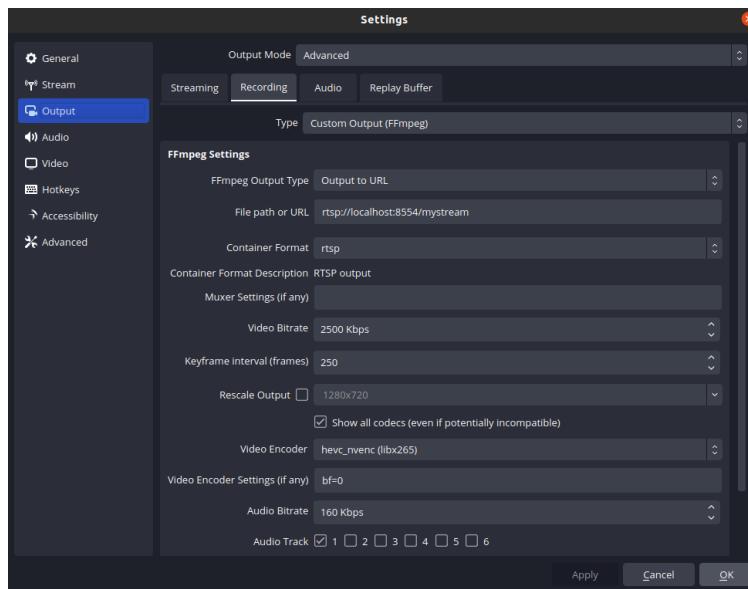


Figura 4.1. Configurare OBS

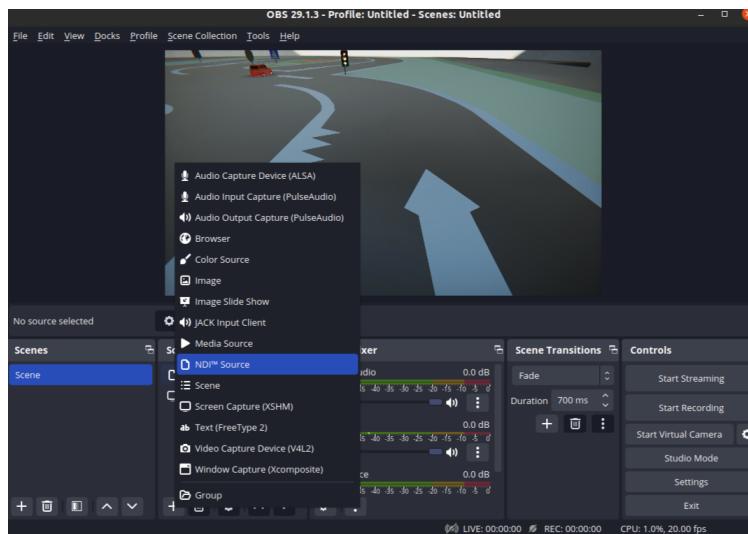


Figura 4.2. Adăugare sursa

#### 4.1.1.3. Unity

Instalarea platformei Unity se realizează de pe site-ul oficial, urmând pașii standard de descărcare și instalare. Pentru a permite extragerea fluxului video din Unity, este necesară instalarea pluginului KlakNDI. Acest plugin se poate adăuga prin intermediul Unity Package Manager, conform instrucțiunilor de pe pagina GitHub a pluginului[39].

#### 4.1.2. Jetson

Pentru a putea rula aplicația pe platforma Jetson, primul pas implică instalarea JetPack, care furnizează toate componentele de bază necesare, inclusiv driverele și SDK-urile pentru a lucra cu hardware-ul NVIDIA. Instrucțiunile detaliate pentru instalarea JetPack pot fi găsite pe pagina 42

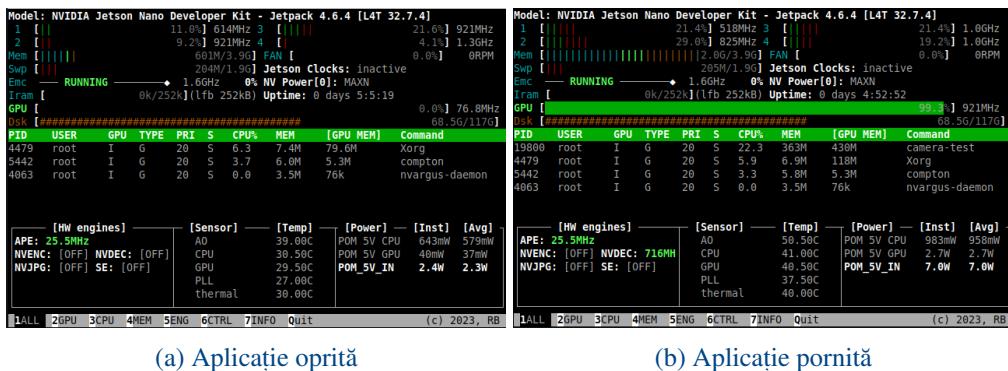


Figura 4.3. Prezentare generală utilizare resurse

oficială NVIDIA[52]. După instalarea JetPack, singura librărie suplimentară care trebuie instalată este jetson-inference, necesară pentru funcționarea specifică a aplicației. Instrucțiunile detaliate pentru instalarea librăriei sunt disponibile pe pagina GitHub a proiectului[42].

## 4.2. Performanța hardware și utilizarea resurselor sistemului

Pentru a evalua performanța hardware și utilizarea resurselor sistemului în cadrul aplicației autonome, se vor analiza mai mulți metrii cheie. Aceste metrii includ cadrele pe secundă (FPS), utilizarea procesorului a plăcii video și utilizarea memoriei. Vom prezenta datele colectate în diverse scenarii de operare pentru a oferi o imagine completă asupra performanței sistemului.

### 4.2.1. Utilizare resurse

În figura 4.3, se poate observa o prezentare generală asupra utilizării resurselor sistemului. Aplicația utilizată pentru a obține aceste metrice este jetson-stats, care este destinată entuziaștilor platformei Jetson pentru monitorizarea și configurarea diverselor aspecte ale sistemului.

Pentru a evalua performanța sistemului, au fost realizate capturi de ecran în două scenarii distincte: unul în care aplicația este oprită, imediat după pornirea Jetsonului, și altul în care aplicația rulează continuu timp de mai mult de 5 minute.

În capturile de ecran din figura 4.3, se poate observa utilizarea generală a resurselor, inclusiv consumul de CPU, GPU, RAM și energie. Este de menționat că în cadrul Jetson Nano, memoria este împărțită între GPU și CPU, ceea ce înseamnă că ambele componente utilizează aceeași memorie fizică, limitând astfel cantitatea disponibilă pentru fiecare în mod individual.

Memoria reală ocupată de aplicație este în jur de 2.9 GB, acest lucru poate fi observat analizând în detaliu memoria în figura 4.4. Memoria afișată în figura 4.3b, care indică în jur de 800 MB, nu reflectă întregul consum. În realitate, memoria ocupată de aplicație include crearea celor două contexte CUDA, inițializarea librăriilor cuDNN și cuBLAS de către TensorRT și alte buffere necesare pentru funcționarea corectă a aplicației.

În ceea ce privește utilizarea procesorului și a plăcii video, procesorul este utilizat în proporție de aproximativ 20%, în timp ce placa video este utilizată la capacitate maximă. Acest lucru subliniază faptul că aplicația este intens orientată spre GPU, datorită sarcinilor complexe de procesare a rețelelor neuronale, care sunt mult mai eficiente atunci când sunt rulate pe GPU. Capturile de ecran care oferă o analiză mai detaliată a utilizării CPU și GPU pot fi consultate în anexa 5.

### 4.2.2. Analiza latentei

Pe parcursul proiectului, s-a pus un accent deosebit pe optimizare pentru a minimiza latența, în special prin optimizarea pipeline-ului de procesare a rețelelor neuronale. Acest lucru a implicat utilizarea intensivă a facilităților CUDA pentru operațiile ce presupun lucrul cu imagini. Modulul



(a) Aplicație oprită

(b) Aplicație pornită

Figura 4.4. Utilizare memorie RAM

de perceptie, conform așteptărilor, prezintă cea mai mare latență, fiind semnificativ mai mare decât celelalte module. Tabelul 4.1 ilustrează durata de efectuare a operațiilor pentru cele două rețele din cadrul modulului de perceptie, atât pe procesor, cât și pe GPU, în funcție de diferitele etape ale procesului: pre-procesare, procesare rețea, post-procesare, vizualizare și timpul total.

Tabelul 4.1. Latenta modul perceptie(ms)

Modulul de perceptie		FastSCNN		YOLOv3-tiny	
		CPU	CUDA	CPU	CUDA
Pre-Process	0.2	4.1		0.1	0.8
Network	5.4		53.2	21.2	20.5
Post-Process	0.15		5.65	0.06	0.1
Visualize	0.1		3.3	0.28	4.6
<b>Total</b>		5.85	65.2	21.64	26

În ceea ce privește latența totală a aplicației, aceasta poate fi observată în figura 4.2. Se observă ca modulele de planificare și control nu introduc o latență semnificativă. Achiziția de imagini, însă, introduce o latență de 12 ms, care nu poate fi redusă, deși s-a încercat acest lucru. În ansamblu, sistemul atinge o latență de calcul de aproximativ 100 ms, ceea ce se traduce într-o capacitate de procesare de aproximativ 10 cadre pe secundă. Această performanță este adekvată pentru funcționarea vehiculului, asigurând o procesare suficient de rapidă.

Tabelul 4.2. Latenta totala(ms)

Aplicația main			
	CPU	CUDA	Durata totală
Perception	27.49	91.2	91.2
Planning	0.12	0	0.12
Control	0.17	0	0.17
Achiziție imagini	-	-	12.1
Bucla while	-	-	103.59

Pe lângă latența inherentă a aplicației, există și o latență suplimentară introdusă de limitările rețelei de transmisie video. Transmisia video de la Unity la Jetson nu se realizează instantaneu, având o întârziere de aproximativ 350 ms. Practic, aceasta înseamnă că Jetsonul primește imagini ușor întârziate. Din păcate, această latență nu poate fi eliminată, deși pare a fi o limitare substanțială. Totuși, aceasta nu afectează în mod semnificativ performanța și funcționarea mașinii.

### 4.3. Testarea sistemului

Testarea sistemului are ca obiectiv principal verificarea funcționalității și performanței aplicației autonome în condiții similare celor din competiție. Aceasta implică evaluarea corectitudinii manevrelor efectuare, reacția sistemului la diferite scenarii de trafic și evaluarea generală a stabilității și fiabilității acestuia.

#### 4.3.1. Metodologie

Testarea mașinii a implicat multiple ture pe pistă pentru a evalua corectitudinea manevrelor necesare conform criteriilor concursului. Aceasta include oprirea la semafor la roșu și galben, oprirea la semnele de stop, efectuarea corectă a parcării și depășirea obstacolelor. În Unity, a fost implementată o logică care verifică dacă mașina se află în apropierea unui semn sau semafor și dacă procedează corect. Astfel, au fost numărate de câte ori mașina eșuează în fiecare manevră specifică. Datele obținute din aceste teste sunt prezentate în figura x, evidențiind frecvența și tipul erorilor comise.

În cazul semnului PARK conform regulamentului se așteaptă oprirea definitiva în zona de finalizare a pistei, însă pentru a facilita testarea prin ture repetate aceasta a fost schimbată cu oprirea pentru 5 secunde.

#### 4.3.2. Rezultate

Pentru a evalua performanțele mașinii autonome, s-au realizat 132 de ture de pistă, iar rezultatele obținute sunt prezentate mai jos.

Tabelul 4.3. Rezultatele testării mașinii autonome

Metrică	Succese	Eșecuri
Număr de tururi complete	131	1
Număr de tururi fără coliziuni	132	0
Număr de opriri la semnul STOP (>3 sec)	132	0
Număr de opriri la semnul CROSS (>1 sec)	132	0
Număr de opriri la semnul PARK (>5 sec)	132	0
Număr de manevre de parcare efectuate corect	125	7
Număr de traversări regulamentare a intersecției	115	17

Se observă că mașina greșește rar, din cele 132 de ture, doar una nu a fost finalizată deoarece vehiculul a rămas blocat în intersecție după traversarea pe starea galben-roșu, ceea ce a dus la oprirea mașinii după semafor pentru a aștepta starea verde. În ceea ce privește depășirea obstacolului, mașina a reușit de fiecare dată să îl evite. Oprirea la semnele de PC1 1.2a, SS2 1.2b și PS3 1.2c s-a realizat corect în fiecare tură, vehiculul fiind capabil să le detecteze fără greș. Sapte dintre manevrele de parcare nu au reușit deoarece mașina nu a detectat semnul CS4 1.2d. La semafor, mașina a trecut pe verde de 115 ori, de 16 ori a trecut pe tranziția verde-galben și o singură dată a trecut pe tranziția galben-roșu, ceea ce a dus la blocarea ei în intersecție.

Rezultatele obținute arată că mașina s-a comportat foarte bine în majoritatea scenariilor de testare, demonstrând o rată de succes ridicată. Cele mai notabile limitări sunt legate de detectarea mai fidelă a semnului CS4 pentru inițierea parcării și de faptul că depășirea funcționează eficient doar atunci când obstacolul este poziționat în locuri mai simple, fiind nefuncțională în intersecții complexe. De asemenea, traversarea atunci când semaforul trece de la verde la galben poate fi interpretată ca un comportament acceptabil și nu neapărat ca o eroare, dat fiind că în multe scenarii de trafic real, vehiculele continuă să se deplaseze în această situație pentru a evita oprirea bruscă.



## Concluzii

Dintre toate cerințele propuse pentru mașinuță, modulul de control este singurul care nu respectă pe deplin viziunea inițială. La început, planul era ca acest modul să implementeze algoritmi PID pentru controlul lateral și longitudinal, calculând comenzi de accelerare și viraj direct pe placa Jetson. Totuși, din cauza constrângerilor de timp, s-a decis ca algoritmii PID să rămână în mediul Unity, conform proiectului inițial. Astfel, modulul de control se ocupă doar de transmiterea comenziilor de referință către Unity.

Înțial, nu mă așteptam ca mașina să ajungă să se comporte atât de bine, chiar și în mediul simulat, depășind cu mult așteptările stabilite încă din anul 2. Am avut mari îndoieri legate de performanțele plăcii Jetson Nano, mai ales având în vedere plăcile folosite de alți participanți care erau de cel puțin 10 ori mai rapide. Cu toate acestea, Jetson Nano s-a descurcat remarcabil de bine, fiind capabil să ruleze simultan două rețele neuronale și să mențină o rată de procesare de 10 cadre pe secundă. Această performanță a fost surprinzătoare și a depășit așteptările inițiale, arătând că, deși Jetson Nano este o placă mai puțin performantă în comparație cu altele, prin optimizare și un pic de optimism, poate susține totuși sarcini complexe de procesare în timp real.

### *Contribuții personale*

În cadrul contribuțiilor personale, am dezvoltat și optimizat un proces pentru eliminarea distorsiunilor camerei simultan cu transformarea perspectivei. Explorarea literaturii de specialitate pe această temă nu a relevat o soluție similară, deoarece algoritmii IPM utilizati sunt mai complecși și se bazează pe soluții de învățare automată. Procesul de agregare a vectorilor de mapare, deși simplu în esență, reprezintă o realizare semnificativă pentru mine. Această metodă poate fi extinsă și la alte procese ce implică aplicarea unor operații de transformare pentru a îmbunătăți timpul de procesare.

A fost creat un depozit denumit IPManCamCalib[51] pentru a facilita cele două transformări. Acesta include două aplicații, una pentru calibrarea manuală a camerei, utilizând slider pentru estimarea coeficienților de distorsiune, și alta pentru schimbarea perspectivei și agregarea seturilor de vectori de mapare. Acest proces simplifică preprocesarea imaginilor și contribuie la îmbunătățirea performanței generale a sistemului.

### *Direcții viitoare de dezvoltare*

Un pas esențial în dezvoltarea ulterioară a proiectului ar fi adaptarea sistemului pentru testarea pe pista de competiție reală. Aceasta ar permite verificarea funcționalității și performanței în condiții reale, aducând contribuții semnificative în cadrul competiției. De menționat este că în ultimele două ediții, niciuna dintre echipe nu a reușit să îndeplinească toate cerințele, astfel că atingerea acestui obiectiv ar reprezenta o realizare notabilă, evidențiind robustețea și eficiența sistemului dezvoltat.

Un alt aspect important al dezvoltării viitoare ar fi experimentarea și utilizarea altor platforme Jetson, mai performante decât Jetson Nano. Platformele avansate, cum ar fi Jetson Xavier NX sau Jetson AGX Xavier, oferă resurse computaționale superioare, permitând implementarea unor tehnici mai complexe și avansate. Acestea includ algoritmi SLAM pentru maparea și localizarea simultană, metode avansate de planificare a traiectoriei și control, cum ar fi controlul predictiv modelat (MPC), și utilizarea de rețele neuronale mai performante și mai fiabile. De asemenea, procesarea simultană a inputurilor de la mai multe camere ar oferi un unghi de vedere mai larg și o percepție mai detaliată a mediului înconjurător.

Dezvoltarea unui sistem autonom capabil să opereze în spații publice, cum ar fi parcări și drumuri interne ale campusurilor universitare, ar reprezenta un alt obiectiv ambicios. Acest lucru

ar demonstra capacitatea sistemului de a naviga în medii mai complexe decât cele de competiție. Implementarea unui astfel de sistem ar necesita integrarea avansată a tehnologiilor de percepție, planificare și control, împreună cu asigurarea unui nivel ridicat de fiabilitate și siguranță. Aceste dezvoltări ar extinde considerabil aplicabilitatea și utilitatea practică a vehiculului autonom.

Experimentarea cu sisteme end-to-end nu a fost explorată în totalitate, în special cele bazate pe învățarea prin întărire. Aceste sisteme pot beneficia de simulatoare pentru antrenarea lor. O astfel de implementare ar putea fi abordată, comparând performanțele cu abordarea modulară, pentru a determina avantajele și limitările fiecărei metode.

#### *Lecții învățate pe parcursul dezvoltării lucrării de diplomă*

Un aspect esențial a fost importanța documentării și cercetării amănuntele înainte de implementarea efectivă a soluțiilor. O simplă ignoranță în înțelegerea platformelor de antrenare a rețelei de segmentare a condus la multe bătăi de cap și timp pierdut care ar fi putut fi evită. Analiza detaliată a fișierelor de configurare și înțelegerea profundă a procesului de antrenare s-au dovedit a fi esențiale pana la urma. Această experiență mi-a arătat că pregătirea minuțioasă și înțelegerea profundă a fiecărui aspect tehnic sunt fundamentale pentru a evita problemele pe termen lung și pentru a asigura succesul proiectului.

## Bibliografie

- [1] S. Singh, “Critical reasons for crashes investigated in the national motor vehicle crash causation survey,” <https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812506>, 2018, Ultima accesare: 20.02.2024.
- [2] M. Massar, I. Reza, S. M. Rahman, S. M. H. Abdullah, A. Jamal, and F. S. Al-Ismail, “Impacts of autonomous vehicles on greenhouse gas Emissions-Positive or negative?” *Int J Environ Res Public Health*, vol. 18, no. 11, May 2021.
- [3] “ideas engineering competition,” <https://ideas-competition.ro/>, Ultima accesare: 21.02.2024.
- [4] “Nxp cup,” <https://nxpcup.nxp.com/>, Ultima accesare: 24.06.2024.
- [5] Google, “Google colaboratory,” <https://colab.research.google.com/>, Ultima accesare: 26.06.2024.
- [6] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar, and D. Lange, “Unity: A general platform for intelligent agents,” 2020. [Online]. Available: <https://arxiv.org/abs/1809.02627>
- [7] “Levels of driving automation,” <https://www.sae.org/news/press-room/2018/12/sae-international-releases-updated-visual-chart-for-its-9Levels-of-driving-automation>, Ultima accesare: 27.06.2024.
- [8] “Front crash prevention slashes police-reported rear-end crashes,” <https://www.iihs.org/news/detail/front-crash-prevention-slashes-police-reported-rear-end-crashes>, Ultima accesare: 28.06.2024.
- [9] “Pedestrian crash avoidance systems cut crashes — but not in the dark,” <https://www.iihs.org/news/detail/pedestrian-crash-avoidance-systems-cut-crashes--but-not-in-the-dark>, Ultima accesare: 28.06.2024.
- [10] S. Campbell, N. O’Mahony, L. Krpalcova, D. Riordan, J. Walsh, A. Murphy, and C. Ryan, “Sensor technology in autonomous vehicles : A review,” in *2018 29th Irish Signals and Systems Conference (ISSC)*, 2018, pp. 1–4.
- [11] “Introducing the 5th-generation waymo driver,” <https://waymo.com/blog/2020/03/introducing-5th-generation-waymo-driver/>, Ultima accesare: 28.06.2024.
- [12] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, “A survey of autonomous driving: Common practices and emerging technologies,” *IEEE Access*, vol. 8, pp. 58 443–58 469, 2020.
- [13] M. A. Rehman, M. Numan, H. Tahir, U. Rahman, M. W. Khan, and M. Z. Iftikhar, “A comprehensive overview of vehicle to everything (v2x) technology for sustainable ev adoption,” *Journal of Energy Storage*, vol. 74, p. 109304, 2023. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2352152X23027020>
- [14] D. A. Pomerleau, “Alvinn: An autonomous land vehicle in a neural network,” *Advances in neural information processing systems*, vol. 1, 1988.

- [15] U. Muller, J. Ben, E. Cosatto, B. Fleep, and Y. LeCun, “Autonomous off-road vehicle control using end-to-end learning,” *Courant Institute/CBLL, Arlington, VA, USA, Tech. Rep. DARPA-IPTO Final technical Report*, vol. 458, 2004.
- [16] Z. Nie and J. Qu, “Autonomous driving smart car based on deep learning,” *INTERNATIONAL SCIENTIFIC JOURNAL OF ENGINEERING AND TECHNOLOGY (ISJET)*, vol. 7, no. 1, p. 1–24, Jun. 2023. [Online]. Available: <https://ph02.tci-thaijo.org/index.php/isjet/article/view/245767>
- [17] Y. G. Khidhir and A. H. Morad, “Real-time end-to-end self-driving car navigation,” *International Journal of Intelligent Systems and Applications in Engineering*, vol. 11, no. 2s, pp. 366–372, 2023.
- [18] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” 2015.
- [19] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, 2001, pp. I–I.
- [20] P. Viola, M. Jones *et al.*, “Robust real-time object detection,” *International journal of computer vision*, vol. 4, no. 34–47, p. 4, 2001.
- [21] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR’05)*, vol. 1. Ieee, 2005, pp. 886–893.
- [22] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: [https://proceedings.neurips.cc/paper\\_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf)
- [23] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, “ImageNet Large Scale Visual Recognition Challenge,” *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.
- [24] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” 2016.
- [25] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, *SSD: Single Shot MultiBox Detector*. Springer International Publishing, 2016, p. 21–37. [Online]. Available: [http://dx.doi.org/10.1007/978-3-319-46448-0\\_2](http://dx.doi.org/10.1007/978-3-319-46448-0_2)
- [26] J. Redmon and A. Farhadi, “Yolov3: An incremental improvement,” 2018. [Online]. Available: <https://arxiv.org/abs/1804.02767>
- [27] C.-Y. Wang, I.-H. Yeh, and H.-Y. M. Liao, “Yolov9: Learning what you want to learn using programmable gradient information,” 2024.
- [28] “Baidu apollo team (2017), apollo: Open source autonomous driving,” <https://github.com/ApolloAuto/apollo>, Ultima accesare: 24.06.2024.
- [29] “Jetson nano brings ai computing to everyone,” <https://developer.nvidia.com/blog/jetson-nano-ai-computing>, Ultima accesare: 24.06.2024.

- [30] N. J. Zakaria, M. I. Shapiai, R. Abd Ghani, M. N. M. Yassin, M. Z. Ibrahim, and N. Wahid, “Lane detection in autonomous vehicles: A systematic review,” *IEEE access*, vol. 11, pp. 3729–3765, 2023.
- [31] S. Cakir, M. Gauß, K. Häppeler, Y. Ounajjar, F. Heinle, and R. Marchthaler, “Semantic segmentation for autonomous driving: Model evaluation, dataset generation, perspective comparison, and real-time capability,” *arXiv preprint arXiv:2207.12939*, 2022.
- [32] “Jetson nano tensorrt fast-scnn 576x576 fp16,” [https://www.youtube.com/watch?v=Qy4S4\\_5JCAM](https://www.youtube.com/watch?v=Qy4S4_5JCAM), Ultima accesare: 24.06.2024.
- [33] Y. Liu, L. Chu, G. Chen, Z. Wu, Z. Chen, B. Lai, and Y. Hao, “Paddleseg: A high-efficient development toolkit for image segmentation,” 2021.
- [34] M. Contributors, “MMSegmentation: Openmmlab semantic segmentation toolbox and benchmark,” <https://github.com/open-mmlab/mmsegmentation>, 2020.
- [35] S. Teng, X. Hu, P. Deng, B. Li, Y. Li, Y. Ai, D. Yang, L. Li, Z. Xuanyuan, F. Zhu *et al.*, “Motion planning for autonomous driving: The state of the art and future perspectives,” *IEEE Transactions on Intelligent Vehicles*, vol. 8, no. 6, pp. 3692–3711, 2023.
- [36] D. Hrovat, S. Di Cairano, H. E. Tseng, and I. V. Kolmanovsky, “The development of model predictive control in automotive industry: A survey,” in *2012 IEEE International Conference on Control Applications*. IEEE, 2012, pp. 295–302.
- [37] “diagrams.net,” <https://app.diagrams.net/>, Ultima accesare: 22.06.2024.
- [38] “Simpleraycastvehicle,” <https://github.com/unity-car-tutorials/SimpleRaycastVehicle-Unity>, Ultima accesare: 24.06.2024.
- [39] K. Takahashi, “Klakndi,” <https://github.com/keijiro/KlakNDI>, Ultima accesare: 07.06.2024.
- [40] “Distroav,” <https://github.com/DistroAV/DistroAV>, Ultima accesare: 01.07.2024.
- [41] “Mediamtx,” <https://github.com/bluenviron/mediamtx>, Ultima accesare: 07.06.2024.
- [42] “jetson-inference,” <https://github.com/dusty-nv/jetson-inference>, Ultima accesare: 24.06.2024.
- [43] P. Ganesh, Y. Chen, Y. Yang, D. Chen, and M. Winslett, “Yolo-ret: Towards high accuracy real-time object detection on edge gpus,” in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2022, pp. 3267–3277.
- [44] W. Chen, X. Gong, X. Liu, Q. Zhang, Y. Li, and Z. Wang, “Fasterseg: Searching for faster real-time semantic segmentation,” 2020. [Online]. Available: <https://arxiv.org/abs/1912.10917>
- [45] R. P. K. Poudel, S. Liwicki, and R. Cipolla, “Fast-scnn: Fast semantic segmentation network,” 2019. [Online]. Available: <https://arxiv.org/abs/1902.04502>
- [46] “Ipm-master,” <https://github.com/JamesLiao714/IPM-master>, Ultima accesare: 24.06.2024.
- [47] Luxonis, “Yolov3-tiny training tutorial,” [https://github.com/luxonis/depthai-ml-training/blob/master/colab-notebooks/YoloV3\\_V4\\_tiny\\_training.ipynb](https://github.com/luxonis/depthai-ml-training/blob/master/colab-notebooks/YoloV3_V4_tiny_training.ipynb), Ultima accesare: 26.06.2024.

- [48] “labelImg,” <https://github.com/HumanSignal/labelImg>, Ultima accesare: 27.06.2024.
- [49] M. Denninger, M. Sundermeyer, D. Winkelbauer, Y. Zidan, D. Olefir, M. Elbadrawy, A. Lodhi, and H. Katam, “Blenderproc,” 2019. [Online]. Available: <https://arxiv.org/abs/1911.01911>
- [50] Unity Technologies, “Unity Perception package,” <https://github.com/Unity-Technologies/com.unity.perception>, 2020.
- [51] “Ipmandcamcalib,” <https://github.com/butnaruteodor/IPManCamCalib>, Ultima accesare: 24.06.2024.
- [52] “Get started with jetson nano developer kit,” <https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit>, Ultima accesare: 24.06.2024.

## Anexe

### Anexa 1. Clasa Perception

```

1  class Perception
2  {
3      public:
4          /* Perception module constructor */
5          Perception() : seg_network(segModelPath),
6                          → det_network(detModelPath)
6      {
7          if (!cudaAllocMapped(&det_vis_image,
8                          → make_int2(DETECTION_ROI_W, DETECTION_ROI_H)))
9          {
10             LogError("Perception: Failed to allocate CUDA memory for
11                 → detection vis image\n");
12         }
13         if (!cudaAllocMapped(&left_lane_x_limits, 2 * IMG_HEIGHT *
14                         → CONTOUR_RES * sizeof(int)))
15         {
16             LogError("Perception: Failed to allocate CUDA memory for
17                 → left_lane_x_limits\n");
18         }
19         if (!cudaAllocMapped(&right_lane_x_limits, 2 * IMG_HEIGHT *
20                         → CONTOUR_RES * sizeof(int)))
21         {
22             LogError("Perception: Failed to allocate CUDA memory for
23                 → right_lane_x_limits\n");
24         }
25         if (!cudaAllocMapped(&charging_pad_center, 4 * sizeof(int)))
26         {
27             LogError("Perception: Failed to allocate CUDA memory for
28                 → charging_pad_center\n");
29         }
30         if (!cudaAllocMapped(&obstacle_limits, 4 * sizeof(int)))
31         {
32             LogError("Perception: Failed to allocate CUDA memory for
33                 → obstacle_limits\n");
34         }
35         perf_profiler_ptr = PerfProfiler::getInstance();
36         InitNetworks();
37     };
38
39     /* Perception module destructor */
40     ~Perception();
41     /* Main function that processes the module */
42     int Process(pixelType *img_input, pixelType *img_output);
43     /* Get detected sign interface function */
44     int GetDetection(Yolo::Detection* det);
45     /* Get segmap pointer */
46     uint8_t* GetSegmapPtr();

```

```

39     /* Get the array that contains the x limits coordinates of left
40     ↵ lane */
41     int* GetLeftLaneXLimits();
42     /* Get the array that contains the x limits coordinates of right
43     ↵ lane */
44     int* GetRightLaneXLimits();
45     /*Get the charging pad center array pointer */
46     int* GetChargingPadCenter();
47     /* Get the obstacle limits array */
48     int* GetObstacleLimitsArray();

49 private:
50     /* Performance profiler instance pointer */
51     PerfProfiler* perf_profiler_ptr;
52     /* Network TensorRT objects */
53     FastScnn seg_network;
54     YoloV3 det_network;

55     uint8_t* classmap_ptr;
56     int* left_lane_x_limits;
57     int* right_lane_x_limits;
58     /* Charging pad limits array */
59     int *charging_pad_center;
60     /* Obstacle limits array */
61     int *obstacle_limits;
62     /* Pointer to the detection overlay image */
63     pixelType *det_vis_image;
64     /* Structure that contains info about the detected sign */
65     DetectedSign detected_sign;
66     /* Initialize the networks*/
67     int InitNetworks();
68     /* Get the image thats fed to the detection network before
69     ↵ preprocessing */
70     void GetDetImage(pixelType *input_img);
71     /* Overlay detection image on the visualization image */
72     void OverlayDetImage(pixelType *vis_img);
73     /* Overlay bboxes on det image */
74     void OverlayBBoxesOnVisImage(uchar3 *out_image, int img_width,
75     ↵ int img_height);
76     /* Filter the detections to get a reliable reading of the
77     ↵ detected sign */
78     void FilterDetections(std::vector<Yolo::Detection> detections);
79     /* Overlay the segmap*/
80     void OverlaySegImage(pixelType *img, int middle_lane_x);
81     /* Initialize all the limits arrays */
82     void InitializeLimitsArrays();
83 }
```

## Anexa 2. Clasa Planning

```

1  class Planning
2  {
3
4  public:
5      Planning();
6      ~Planning();
7      /* Overlay lane center on image */
8      void OverlayLanePoints(uchar3 *out_img);
9      /* Get the lateral setpoint */
10     int GetLateralSetpoint();
11     /* Get the longitudinal setpoint */
12     int GetLongitudinalSetpoint();
13     /* Process the module, get perception data and run state handler
14     ↵ */
15     void Process(Perception *perception_module);
16
17 private:
18     enum State
19     {
20         WAIT,
21         DRIVE,
22         STOP,
23         PARK
24     };
25     struct WaitStruct
26     {
27         int last_time_sec;
28         int time_to_stop_sec;
29         bool first_time_wait;
30     };
31     State state;
32     State last_state;
33     int speed_sp;
34     int lateral_sp;
35
36     Yolo:::Detection detected_sign;
37     WaitStruct wait_struct;
38
39     PerfProfiler *perf_profiler_ptr;
40
41     /* Lane extremities x values, taken raw from perception module */
42     int *left_lane_x_limits;
43     int *right_lane_x_limits;
44     /* Charging pad center*/
45     int *charging_pad_center;
46     /* Obstacle limits array */
47     int *obstacle_limits;
48     /* Right lane center x values */
49     int right_lane_x[OUT_IMG_H / CONTOUR_RES];
50     /* Left lane center x values */
51     int left_lane_x[OUT_IMG_H / CONTOUR_RES];
52

```

```
53     /* Pre processes the data from the perception module, gives the
      ↵ lane center points */
54     void GetLaneCenterPoints();
55     /* Get lane lateral setpoint by lane id */
56     int GetLaneCenter(int lane_idx);
57     /* Choose the free lane */
58     bool IsObstacleOnLane();
59     /* Run the brain */
60     void RunStateHandler();
61     /* Get the detected sign and the pointer to the segmap */
62     void GetPerceptionData(Perception *perception_module);
63 }
```

### Anexa 3. Algoritm mapare

```

1  __global__ void rgbToIpm(uchar3 *input, float *output, int *uGrid,
2    ↵  int *vGrid, int width, int height)
3  {
4    int x = blockIdx.x * blockDim.x + threadIdx.x;
5    int y = blockIdx.y * blockDim.y + threadIdx.y;
6
7    if (x >= width || y >= height - 1)
8      return;
9
10   int uvIndex = y * width + x;
11
12   int ui = uGrid[uvIndex];
13   int vi = vGrid[uvIndex];
14
15   int inIndex = vi * 1920 + ui;
16   int outIndex = y * width + x;
17
18   if (ui >= 0 && ui < 1920 && vi >= 0 && vi < 1080)
19   {
20     output[outIndex] = (input[inIndex].x / 255.0f - 0.485f) /
21       ↵ 0.229f;
22     output[outIndex + 524288] = (input[inIndex].y / 255.0f -
23       ↵ 0.456f) / 0.224f;
24     output[outIndex + 1048576] = (input[inIndex].z / 255.0f -
25       ↵ 0.406f) / 0.225f;
26   }
27   else
28   {
29     output[outIndex] = -2.11790393f;
30     output[outIndex + 524288] = -2.035714286f;
31     output[outIndex + 1048576] = -1.804444444f;
32   }
33 }
```

#### Anexa 4. Functii initializare retele

```

1   int FastScnn::InitEngine()
2 {
3     // Context
4     if (!mEngine)
5       return false;
6     mContext = mEngine->createExecutionContext();
7     if (!mContext)
8     {
9       sample::gLogger.log(nvinfer1::ILogger::Severity::kERROR, "Failed
10      to create execution context");
11     }
12   }
13
14   const int numBindings = mEngine->getNbBindings();
15
16   std::vector<std::string> output_blobs;
17   std::vector<std::string> input_blobs;
18   output_blobs.push_back("save_infer_model/scale_0.tmp_0");
19   output_blobs.push_back("save_infer_model/scale_1.tmp_0");
20   input_blobs.push_back("x");
21
22   const int numInputs = input_blobs.size();
23   int mMaxBatchSize = 1;
24   for (int n = 0; n < numInputs; n++)
25   {
26     const int inputIndex =
27       mEngine->getBindingIndex(input_blobs[n].c_str());
28
29     if (inputIndex < 0)
30     {
31       LogError("failed to find requested input layer %s in
32         network\n", input_blobs[n].c_str());
33       return false;
34     }
35
36     LogVerbose("binding to input %i %s binding index: %i\n", n,
37     input_blobs[n].c_str(), inputIndex);
38     nvinfer1::Dims inputDims =
39       validateDims(mEngine->getBindingDimensions(inputIndex));
40     inputDims = shiftDims(inputDims);
41     const size_t inputSize = mMaxBatchSize * sizeDims(inputDims, 1) *
42       sizeof(float);
43     LogVerbose("binding to input %i %s dims (b=%u c=%u h=%u w=%u)
44       size=%zu\n", n, input_blobs[n].c_str(), mMaxBatchSize,
45       DIMS_C(inputDims), DIMS_H(inputDims), DIMS_W(inputDims),
46       inputSize);
47
48   // allocate memory to hold the input buffer
49   void *inputCPU = NULL;
50   void *inputCUDA = NULL;
51
52   if (!cudaAllocMapped((void **) &inputCPU, (void **) &inputCUDA,
53     inputSize))
54 
```

```

44     {
45         LogError("failed to alloc CUDA mapped memory for tensor input,
46             ↳ %zu bytes\n", inputSize);
47         return false;
48     }
49
50     layerInfo l;
51
52     l.CPU = (float *)inputCPU;
53     l.CUDA = (float *)inputCUDA;
54     l.size = inputSize;
55     l.name = input_blobs[n];
56     l.binding = inputIndex;
57
58     copyDims(&l.dims, &inputDims);
59     mInputs.push_back(l);
60 }
61 /** setup network output buffers
62 */
63 const int numOutputs = output_blobs.size();
64
65 for (int n = 0; n < numOutputs; n++)
66 {
67     const int outputIndex =
68         ↳ mEngine->getBindingIndex(output_blobs[n].c_str());
69
70     if (outputIndex < 0)
71     {
72         LogError("failed to find requested output layer %s in
73             ↳ network\n", output_blobs[n].c_str());
74         return false;
75     }
76
77     LogVerbose("binding to output %i %s binding index: %i\n", n,
78             ↳ output_blobs[n].c_str(), outputIndex);
79
80     nvinfer1::Dims outputDims =
81         ↳ validateDims(mEngine->getBindingDimensions(outputIndex));
82     outputDims = shiftDims(outputDims); // change NCHW to CHW if
83         ↳ EXPLICIT_BATCH set
84     const size_t outputSize = mMaxBatchSize * sizeDims(outputDims, 1)
85         ↳ * sizeof(float);
86     LogVerbose("binding to output %i %s dims (b=%u c=%u h=%u w=%u)
87             ↳ size=%zu\n", n, output_blobs[n].c_str(), mMaxBatchSize,
88             ↳ DIMS_C(outputDims), DIMS_H(outputDims), DIMS_W(outputDims),
89             ↳ outputSize);
90
91     // allocate output memory
92     void *outputCPU = NULL;
93     void *outputCUDA = NULL;
94
95     // if( CUDA_FAILED(cudaMalloc((void**)&outputCUDA, outputSize)) )

```

```

87     if (!cudaAllocMapped((void **)&outputCPU, (void **)&outputCUDA,
88         ↵ outputSize))
89     {
90         LogError("failed to alloc CUDA mapped memory for tensor output,
91             ↵ %zu bytes\n", outputSize);
92         return false;
93     }
94
95     layerInfo l;
96
97     l.CPU = (float *)outputCPU;
98     l.CUDA = (float *)outputCUDA;
99     l.size = outputSize;
100    l.name = output_blobs[n];
101    l.binding = outputIndex;
102
103    copyDims(&l.dims, &outputDims);
104    mOutputs.push_back(l);
105 }
106
107 /*  

108 * create list of binding buffers  

109 */
110 const int bindingSize = numBindings * sizeof(void *);
111
112 if (!mBindings)
113 {
114     LogError("failed to allocate %u bytes for bindings list\n",
115             ↵ bindingSize);
116     return false;
117 }
118
119 memset(mBindings, 0, bindingSize);
120
121 for (uint32_t n = 0; n < GetInputLayers(); n++)
122     mBindings[mInputs[n].binding] = mInputs[n].CUDA;
123
124 for (uint32_t n = 0; n < GetOutputLayers(); n++)
125     mBindings[mOutputs[n].binding] = mOutputs[n].CUDA;
126
127 // find unassigned bindings and allocate them
128 printf("numBindings: %d", numBindings);
129 for (uint32_t n = 0; n < numBindings; n++)
130 {
131     if (mBindings[n] != NULL)
132         continue;
133
134     const size_t bindingSize =
135         ↵ sizeDims(validateDims(mEngine->getBindingDimensions(n)), 1) *
136         ↵ mMaxBatchSize * sizeof(float);
137
138     if (CUDA_FAILED(cudaMalloc(&mBindings[n], bindingSize)))

```

```

136     {
137         LogError("failed to allocate %zu bytes for unused binding
138             ↳ %u\n", bindingSize, n);
139         return false;
140     }
141
142     LogVerbose("allocated %zu bytes for unused binding %u\n",
143             ↳ bindingSize, n);
144 }
145
146
147 if (!cudaAllocMapped((void **)&mClassMap, 1024 * 512 *
148     ↳ sizeof(uint8_t)))
149     return false;
150
151
152
153 // Allocate the mapping arrays for undistortion/ipm
154 cudaError_t u_malloc_err = cudaMalloc((void **)&uGrid, UV_GRID_COLS
155     ↳ * sizeof(int));
156 cudaError_t v_malloc_err = cudaMalloc((void **)&vGrid, UV_GRID_COLS
157     ↳ * sizeof(int));
158
159 if (u_malloc_err != cudaSuccess || v_malloc_err != cudaSuccess)
160 {
161     LogError("Could not allocate uGrid or vGrid\n");
162     return false;
163 }
164
165
166 int status = LoadGrid();
167 if (!status)
168 {
169     LogError("FastScnn: Failed to load grid\n");
170     return 1;
171 }
172
173 return true;
174 }
```

Listing 2. Initializare retea de segmentare

```

1 bool YoloV3::InitEngine()
2 {
3     // Context
4     if (!mEngine)
5         return false;
6
7     mContext = mEngine->createExecutionContext();
8     if (!mContext)
9     {
10         LogError("Failed to create execution context\n");
11         return 0;
12 }
```

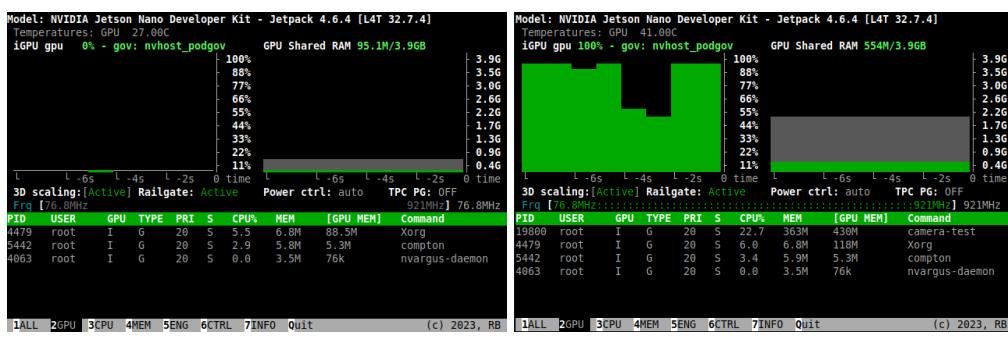
```

12     }
13     auto input_idx = mEngine->getBindingIndex("data");
14
15     if (input_idx == -1)
16     {
17         return false;
18     }
19     assert(mEngine->getBindingDataType(input_idx) ==
20         nvinfer1::DataType::kFLOAT);
21     auto input_dims = mContext->getBindingDimensions(input_idx);
22     mContext->setBindingDimensions(input_idx, input_dims);
23     auto input_size = sizeDims(input_dims, 1) * sizeof(float);
24
25     auto output_idx = mEngine->getBindingIndex("prob");
26     if (output_idx == -1)
27     {
28         return false;
29     }
30     assert(mEngine->getBindingDataType(output_idx) ==
31         nvinfer1::DataType::kFLOAT);
32     auto output_dims = mContext->getBindingDimensions(output_idx);
33     const size_t output_size = sizeDims(output_dims, 1) *
34         sizeof(float);
35
36     void *outputCUDA = NULL;
37     void *outputCPU = NULL;
38     if (!cudaAllocMapped((void **)&outputCPU, (void **)&outputCUDA,
39         input_size))
40     {
41         LogError("Could not allocate output CUDA\n");
42         return false;
43     }
44     void *inputCUDA = NULL;
45     void *inputCPU = NULL;
46     if (!cudaAllocMapped((void **)&inputCPU, (void **)&inputCUDA,
47         input_size))
48     {
49         LogError("Could not allocate input CUDA\n");
50         return false;
51     }
52     LogVerbose("Input size: %d\n", input_size);
53     LogVerbose("Output size: %d\n", output_size);
54
55     const int bindingSize = 2 * sizeof(void *);
56     mBindings = (void **)malloc(bindingSize);
57     mBindings[0] = (float *)inputCUDA;
58     mBindings[1] = (float *)outputCUDA;
59
60     if (cudaStreamCreate(&mStream) != cudaSuccess)
61     {
62         LogError("ERROR: cuda stream creation failed.\n");
63         return false;
64     }

```

```
61
62     LogVerbose("Successfully initialized yolo network\n");
63     return true;
64 }
```

Listing 3. Initializare retea de detectie



(a) App off

(b) App on

Figura A.1. Utilizare GPU



(a) App off

(b) App on

Figura A.2. Utilizare CPU

### Anexa 5. Utilizare GPU si CPU

În Figura A.1 se poate observa o comparatie intre utilizarea resurselor placii video cand aplicatia este oprită si cand este pornită. Fluctuatile utilizarii placii video nu au o influenta reala asupra performantei aplicatiei si sunt datorate mecanismului de protectie a placii pentru supratensiune si subtensiune.