

## **VEHICUL AUTONOM** **NXP CUP 2025**

**Candidat:** Alexandru-Cătălin GULYASY

**Coordonator științific:** S.I.dr.ing. Paul NEGÎRLĂ

Sesiunea: Iunie 2025



## REZUMAT

Această lucrare prezintă dezvoltarea unui vehicul autonom pentru competiția NXP Cup care are ca scop parcurgerea unei piste predefinite fără intervenție umană. Proiectul îmbină atât componentele hardware cât și algoritmi creați pentru a putea avea rezultate în regim de competiție.

Pentru detecția pistei a fost folosită o cameră amplasată în mijlocul mașinii care capturează imaginile în timp real cu ajutorul bibliotecii OpenCV. Toată procesarea imaginii se rulează pe placuța Debix Model B, iar Teensy 4.0 este cea care controlează elementele de execuție.

Un aspect important al acestei lucrări este tot procesul prelucrării imaginii regăsit în capitolul dedicat 4.3 în arhitectura software dar și tehnica Bird Eye View care ajută la percepția mai corectă a mediului înconjurător. Algoritmul principal de setare a traiectoriei mașinii se bazează pe o variantă îmbunătățită a tehnicii Pure Pursuit regăsită în capitolul 4.4 care se folosește de linia de mijloc a traseului, și de dimensiunile vehiculului. Îmbunătățirea constă în **ajustarea dinamică** a distanței lookahead în funcție de viteza vehiculului și de raza curbei detectate.

Într-un final testele efectuate au luat loc pe o pistă proprie cu o aderență mai bună decât cea de la concurs dar și pe cea de la concurs. Acest aspect a afectat performanțele sistemului deoarece a fost calibrat în condiții diferite decât cele de competiție. Cu toate acestea, datorită meniului interactiv setările au putut fi îmbunătățite în ziua de concurs iar rezultatele au fost satisfăcătoare.

## Cuprins

<b>1 Introducere</b>	<b>7</b>
1.1 Contextul proiectului . . . . .	7
1.2 Scopul și motivația . . . . .	8
1.3 Structura lucrării . . . . .	8
1.4 Arhitectura sistemului . . . . .	9
<b>2 Componenta hardware</b>	<b>10</b>
2.1 Microcontrolerul Teensy . . . . .	10
2.2 Modulul SBC Debix . . . . .	10
2.3 Servomotorul pentru direcție . . . . .	12
2.4 Camera USB și alimentarea . . . . .	13
2.5 Motoare și ESC . . . . .	14
2.6 EDF și stabilizare . . . . .	15
2.7 Senzorul LiDAR . . . . .	16
2.8 Modul step-down . . . . .	16
<b>3 Modelarea 3D a șasiului</b>	<b>17</b>
3.1 Mecanism de direcție Ackermann . . . . .	18
3.2 Detaliere EDF . . . . .	19
<b>4 Arhitectura software</b>	<b>20</b>
4.1 Organizarea firelor de execuție . . . . .	21
4.1.1 Structura Process Frames . . . . .	25
4.2 Progresul prelucrării imaginilor . . . . .	28
4.2.1 Simple vs Otsu vs Adaptive Gaussian Threshold . . . . .	29
4.2.2 Thinning/Skeletonizing . . . . .	33
4.2.3 Bird Eye View . . . . .	33
4.3 Prelucrarea imaginilor . . . . .	36
4.3.1 Cazul Following Line . . . . .	43
4.3.2 FINISH LINE DETECTION . . . . .	46
4.3.3 EXTEND LINE TO EDGES . . . . .	47
4.3.4 EVENLY SPACE POINTS . . . . .	47
4.3.5 GET LEFT RIGHT & FIND MIDDLE . . . . .	48
4.3.6 Cazul In Intersection . . . . .	51
4.3.7 Cazul Exiting Intersection . . . . .	52
4.4 Algoritmul Pure Pursuit și modelul de viteză . . . . .	54
4.4.1 Calculare look Ahead Distance . . . . .	55
4.4.2 shortestDistanceToCurve & longestDistanceOnCurveFromPoint . . . . .	56
4.4.3 Căutarea look Ahead Point-ului . . . . .	57
4.4.4 Calculare angle Heading Target . . . . .	59
4.4.5 Calculul vitezei mașinii în funcție de curbă . . . . .	60

4.4.6 Calculul valorii servomotorului pentru direcție . . . . .	61
4.4.7 Implementarea și eliminarea PID . . . . .	61
4.5 Funcționalitatea de oprire de urgență . . . . .	65
4.6 Scenarii de rulare . . . . .	66
4.7 Probleme întâmpinate, soluții și posibile îmbunătățiri . . . . .	66
4.8 Concluzii generale . . . . .	67
<b>Bibliografie</b>	<b>68</b>

## Listă de figuri

1 Pista NXP CUP . . . . .	7
2 Schematica architecturii . . . . .	9
3 Teensy 4.0 . . . . .	10
4 DEBIX Model B . . . . .	11
5 Servo SPT5613 . . . . .	12
6 Arducam B0385 . . . . .	13
7 ESC Blheli . . . . .	14
8 A2212 930KV BLDC . . . . .	14
9 Electric Ducted Fan (EDF) . . . . .	15
10 BT46B Fan car . . . . .	15
11 HBFS linefollower . . . . .	15
12 McMurtry Spéirling Fan car . . . . .	16
13 LIDAR Tf Mini Plus . . . . .	16
14 Modul Step-Down . . . . .	17
15 3D Assembly Design . . . . .	18
16 Ackermann setup . . . . .	18
17 Ackermann angles . . . . .	18
18 Bycicle Model . . . . .	19
19 Final Edf Setup . . . . .	19
20 First Edf Setup . . . . .	19
21 Procesul general . . . . .	20
22 Firele de executie . . . . .	21
23 Secventă din menu_rotary.py . . . . .	22
24 Meniu . . . . .	22
25 Secvență webisteTcpThread . . . . .	23
26 Shared Memory 1 . . . . .	24
27 Shared Memory 2 . . . . .	24
28 processFramesThread_1 . . . . .	25
29 processFramesThread_2 . . . . .	26
30 processFramesThread_3 . . . . .	27
31 Canny și BirdEyeView . . . . .	28

32	cvtColor . . . . .	28
33	bitwise not . . . . .	29
34	Metode de Threshold . . . . .	29
35	Simple Threshold . . . . .	30
36	Otsu Threshold . . . . .	30
37	Adaptive Gaussian Threshold . . . . .	31
38	Canny vs Thinning . . . . .	33
39	Thinning . . . . .	33
40	Normal View vs Bird Eye View . . . . .	33
41	Bird Eye View Setup Points . . . . .	34
42	getPerspectiveTransform() . . . . .	35
43	perspectiveChange(Point/Line) . . . . .	35
44	procesul prelucrării . . . . .	36
45	linia de mijloc . . . . .	37
46	FindLines . . . . .	38
47	generateNeighborhood . . . . .	38
48	diagramă generateNeighborhood . . . . .	39
49	CustomConnectedComponents_1 . . . . .	39
50	Colț procesat de jos în sus . . . . .	40
51	Colț procesat de sus în jos . . . . .	40
52	explicație caz V . . . . .	40
53	CustomConnectedComponents_2 . . . . .	41
54	labelImage . . . . .	41
55	CustomConnectedComponents_3 . . . . .	42
56	CustomConnectedComponents_4 . . . . .	42
57	fitPolynomial inner function . . . . .	43
58	RDP Epsilon 0 . . . . .	43
59	RDP Epsilon 3.8 . . . . .	43
60	FOLLOWING LINE 1 . . . . .	44
61	FOLLOWING LINE 2 . . . . .	45
62	Finish Line View . . . . .	46
63	getLeftRightLines_1 . . . . .	49
64	getLeftRightLines_2 . . . . .	49
65	Mirroring Line 1 . . . . .	50
66	Mirroring Line 2 . . . . .	50
67	IN INTERSECTION . . . . .	51
68	EXITING INTERSECTION 1 . . . . .	52
69	EXITING INTERSECTION 2 . . . . .	53
70	computePurePursuit . . . . .	54
71	computeK . . . . .	55
72	shortestDistanceToCurve . . . . .	56
73	Calibrare inițială . . . . .	62

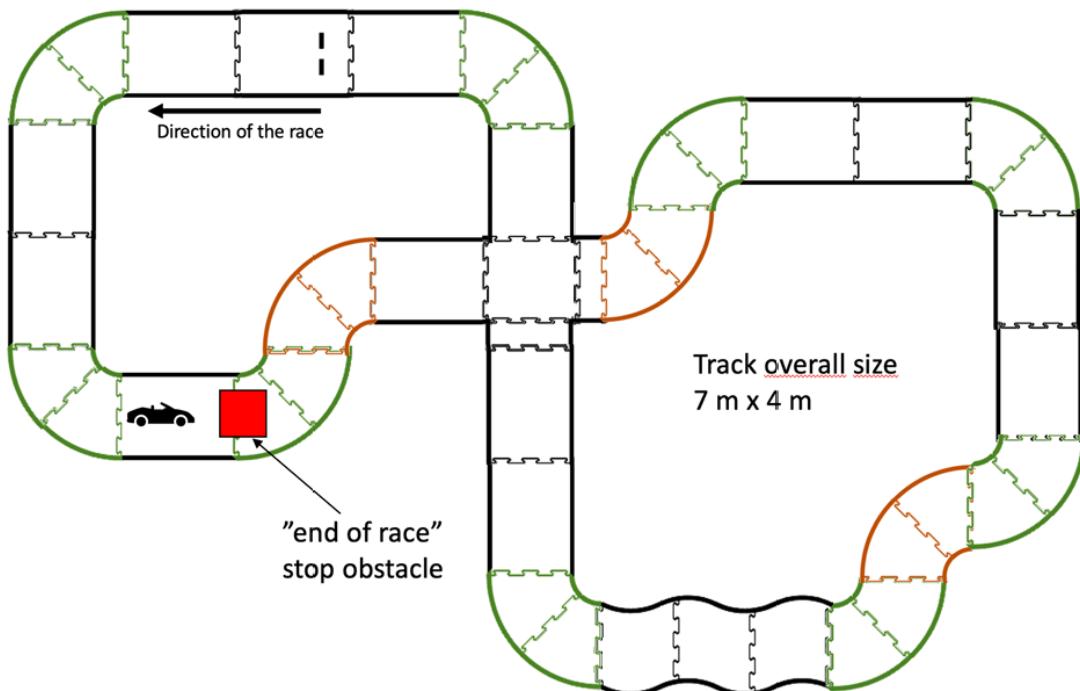
74	Calibrare finală . . . . .	62
75	Măsurători Viteză . . . . .	64
76	Cod de Oprire . . . . .	65
77	Piste rulate . . . . .	66

# 1 Introducere

## 1.1 Contextul proiectului

NXP CUP este un **concurs internațional**, dedicat vehiculelor autonome, care are la baza dezvoltarea robotilor mobili pentru a concura și a obține **cel mai bun timp** pe o pistă specifică desemnată de organizatori. Compania NXP este un producător de semiconductoare și dezvoltare atât hardware cât și software. Prin acest concurs se dorește promovarea produselor interne (folosite în mod obligatoriu la concurs), dar și **achiziția de talente din rândul studenților**.

Conform regulamentului un participant are dreptul la trei încercări consecutive, în cazul în care nu reușește o tură completă în jurul pistei, însă prima tură finalizată este cea care se va lua în calcul. De asemenea conceptul numit "Lap of Honor" este o probă adăugată în care vehiculul trebuie să se oprească la o distanță mai mică de 10cm înaintea obstacolului care aduce 150 de puncte in plus pentru participant [13].



Above: Example of event racetrack layout

Figura 1: Pista NXP CUP

## 1.2 Scopul și motivația

În anul 2024 am participat la concursul NXP CUP cu o echipă formată din 5 persoane și am reușit să securizăm **locul 3** și 4 în etapă internațională. Observând că echipa clasată pe primul loc a făcut segmentare de imagine, mi-am dorit la rândul meu să trec și eu la următorul nivel și să îmi îmbunătățesc cunoștințele de programare, dar și de inginerie aplicată. Astfel am ajuns să lucrez cu tehnologii precum C++, Python, OpenCV, Matlab și am ajuns să înțeleg mai bine sistemul Linux. **Scopul robotului** este să rămână pe pistă, între liniile negre și să termine cursa cât mai rapid posibil.

## 1.3 Structura lucrării

Lucrarea este împărțită pe mai multe capituloare fiecare abordând o etapă esențială a proiectului.

În prima parte a lucrării se prezintă **sistemul hardware**, componentele folosite, **justificarea** acestora precum și şasiul mașinii modelat în program 3D.

Mai apoi vom discuta despre **arhitectura software** și despre procesele din spate care fac legătura totală a sistemului. În subcapitolul **Progresul prelucrării imaginilor** abordăm întregul proces care a adus rezultatul final. Consider că este important pentru cititorii lucrării să înțeleagă modul de găndire abordat în dezvoltarea soluției de segmentare a imaginii pentru a aduce îmbunătățiri ulterioare întregului sistem.

În capitolul **Algoritmul Pure Pursuit și modelul de viteză** vom aborda anumite concepte matematice folosite în dezvoltarea robotului. De-a lungul fiecărui capitol vom justifica fiecare direcție aleasă în dezvoltarea software cu capturi de ecran care cuprind teste făcute pe parcurs. În final vom aborda posibilele îmbunătățiri care vor urma pentru acest proiect.

## 1.4 Arhitectura sistemului

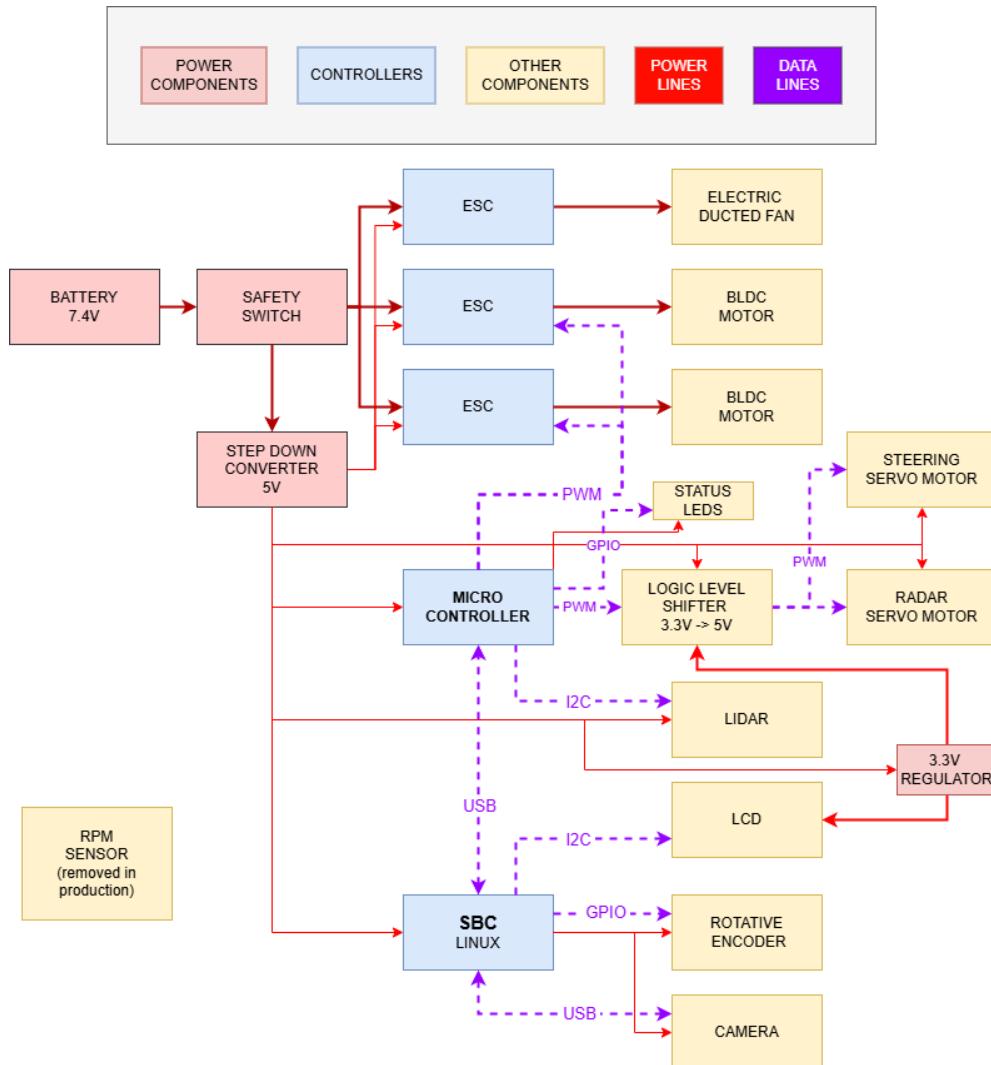


Figura 2: Schematică arhitecturii

SBC (Single Board Computer)-ul cu sistem de operare Linux este componenta principală care realizează procesarea imaginilor și luarea deciziilor pentru acționarea robotului. De asemenea este punctul de debug între robot și programator cu opțiuni precum vizualizarea cadrelor și schimbarea parametrilor în timp real de pe o interfață web. SBC-ul este conectat prin USB la camera, de unde preia informațiile mediului înconjurător, și la microcontroller. Microcontrollerul este cel care comandă în mod direct motoarele (prin semnal PWM), ledurile de stare și citește senzorul de distanță (prin UART), tuometrul(eliminat din cauza spațiului în produsul final).

## 2 Componenta hardware

### 2.1 Microcontrolerul Teensy

Microcontrollerul ales a fost Teensy 4.0 datorită faptului că se poate programa ca un Arduino Uno. Libraria "Arduino.h" face ca dezvoltarea să fie foarte rapidă.

Chipul **NXP i.MX RT1062** este un microcontroler considerat "crossover" dezvoltat pentru a fi folosit în sisteme "real-time" [12].

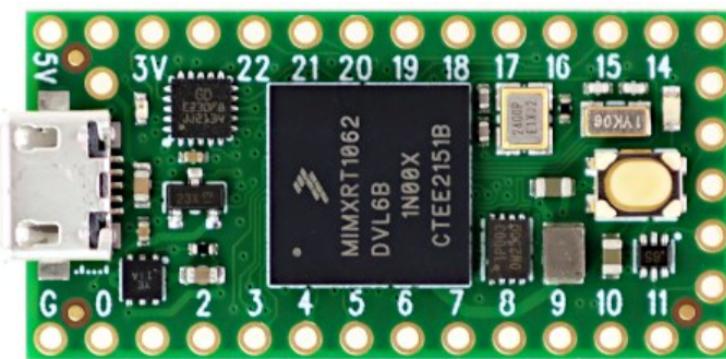


Figura 3: Teensy 4.0  
<https://www.pjrc.com/store/teensy40.html>

- **Procesor:** ARM Cortex-M7 @ 600 MHz

- **Memorie:**

- 1 MB RAM intern
- 8 MB Flash

- **Interfețe:**

- 40 de pini digitali, 33 de pini PWM
- UART, I2C, SPI, CAN, PWM, ADC

### 2.2 Modulul SBC Debix

Concursul a limitat componentele de control folosite la chipuri marca NXP. Debix Model B este o placă asemănătoare Raspberry Pi cu specificații industriale.

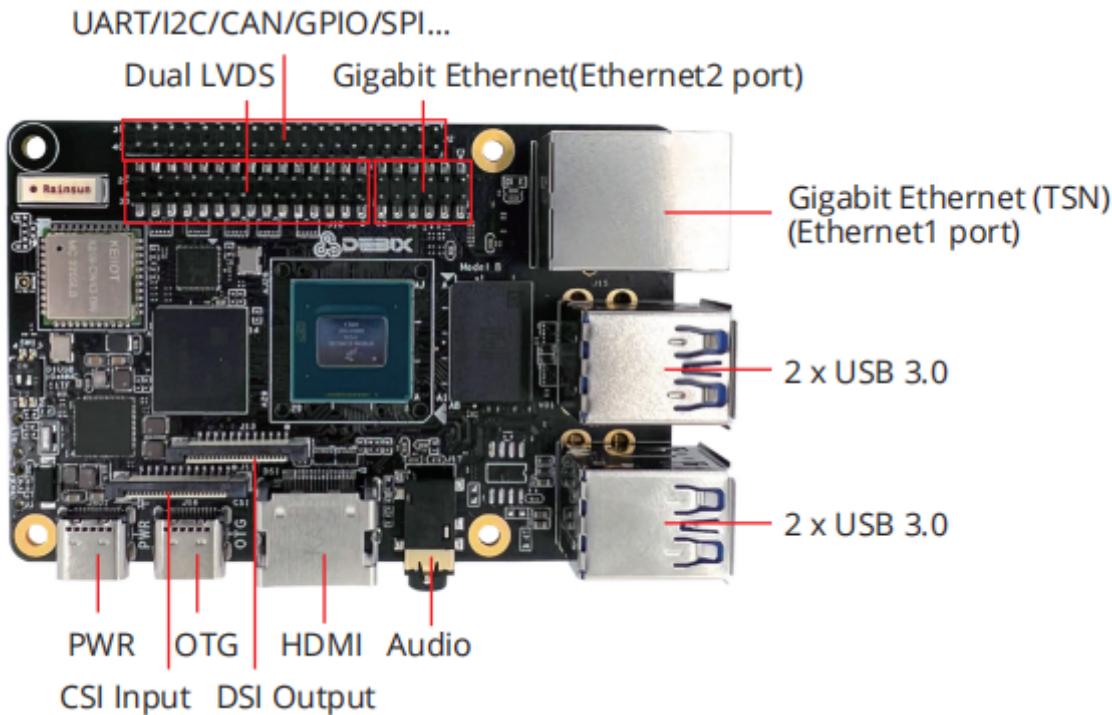


Figura 4: DEBIX Model B  
<https://debix.io/hardware/model-b.html>

• **Procesor:** NXP i.MX 8M Plus

- Quad-core ARM Cortex-A53 @ 1.6 GHz
- NPU integrat: până la 2.3 TOPS (important pentru viitoarele îmbunătățiri)

• **Memorie:**

- 4 GB RAM LPDDR4
- 32 GB stocare eMMC
- Slot pentru card microSD

• **Interfețe:**

- 2 x USB 3.0
- 2 x USB 2.0
- Conector GPIO 40 pini (compatibil Raspberry Pi)

• **Conecțivitate:**

- UART, I2C, SPI, CAN, PWM, ADC
- Wi-Fi și Bluetooth (optional, prin extensie)

- **Sistem de operare și software:**

- Suport Linux (Debian, Ubuntu)
- Compatibil cu OpenCV, TensorFlow Lite, ONNX, GStreamer

Am ales să prezint aceste informații ca să clarific motivul pentru care doar microcontrollerul Teensy nu era îndeajuns. **Teensy** este un microcontroller puternic, însă **nu are memorie destulă** ca să satisfacă cerințele de rulare a bibliotecii OpenCV. Desigur, existau opțiuni pentru a rescrie multe funcții de la zero, însă în prototipare timpul este mereu cel mai limitat. Initial am încercat să dezvolt totul pe placă Debix, dar am ajuns rapid la concluzia că nu este foarte intuitiv și este relativ greu de operat pe fișiere. Ulterior am aflat că **ADC-ul de pe Teensy este foarte bun** și am ales să rămân cu soluția în care am rămas cu ambele componente.

### 2.3 Servomotorul pentru direcție



Figura 5: Servo SPT5613  
<http://spt-servo.com/Product/0943621445.html>

Pentru direcția mașinii a fost utilizat un motor servo **SPT5613**. Acesta a fost ales datorită angrenajelor complet metalice. Conform specificațiilor tehnice, semnalul de control are un ciclu complet de **20 ms**, ceea ce corespunde unei frecvențe PWM standard de **50 Hz**.

## 2.4 Camera USB și alimentarea



Figura 6: Arducam B0385  
Arducam Official Website

Camera este una dintre cele mai importante componente ale acestui sistem. Arducam B0385 are trei atrbute foarte importante:

- **Plug and play** cu ajutorul V4L2 driver
  - Video for Linux 2 (V4L2) este o colecție de drivere disponibile pe platforma Linux care standardizează procesul de captură video în timp-real a camerelor și oferă opțiuni parametrizate de calibrare utilizatorului precum "exposure, contrast, etc." [15].
- **100+ fps Capture** la un preț accesibil
  - Camera acoperă două moduri de captură, una este de 100fps iar cealaltă este de 120fps. Acest aspect este foarte important deoarece luminile din jurul nostru sunt alimentate de o rețea de curent. Frecvența rețelei de curent este de 50Hz în Europa, 60Hz în Statele Unite [3]. Prin opțiunile oferite de cameră reușim să ne apropiem de eliminarea totală a fenomenului numit "flickering", însă mai trebuie să setăm exposure-ul corect.
- **Global Shutter Camera**
  - Diferența între Rolling și Global Shutter este faptul că **Rolling Shutter** preia informațiile rând cu rând de la pixeli iar **Global Shutter** preia informațiile de la toți pixelii deodata. Prin această opțiune ne asigurăm că nu apar artefacte în linia percepă de robot la viteze mari.

## 2.5 Motoare si ESC



Figura 7: ESC Blheli

<https://www.aliexpress.com/i/1005005557304417.html>

Am ales ESC compatibil BIHeli deoarece prin BLHeliSuite programarea se desfășoară foarte ușor. Se pot seta parametrii precum "Brake on Stop" sau "Startup Power".



Figura 8: A2212 930KV BLDC

<https://www.aliexpress.com/i/32868853905.html>

În etapa inițială am pornit cu un singur motor, însă am ajuns destul de rapid la capabilitățile maxime al acestui setup. Combinatia unei roți dințate mici oferea cuplu (torque) foarte bun, însă viteză scazută de 1.5m/s. Următorul setup a fost cu un singur motor cu o roată dințată mai mare. Acum aveam viteza necesară, însă nu mai pleca de pe loc fără să o împing cu mâna din cauza lipsei de cuplu. Ca să adaug al doilea motor a trebuit să renunț la turometru, spațiul disponibil fiind o problema. Noua viteză maximă posibila a ajuns la 3m/s.

## 2.6 EDF și stabilizare



Figura 9: Electric Ducted Fan (EDF)

<https://www.aliexpress.com/item/1005007969957183.html>

Odată cu noua viteză obținută avem nevoie și de o aderență mai bună. Aderența se poate spori în doi pași; în primul rand prin cauciucuri moi, cauciucuri specifice de viteză, iar în al doilea rând prin vidul creat de sub mașina cu un edf (electric ducted fan). Inspirația a venit din trei locuri importante:



Figura 10: BT46B Fan car

[https://en.wikipedia.org/wiki/Brabham\\_BT46](https://en.wikipedia.org/wiki/Brabham_BT46)

Brabham F1 debutează în 1978 cu o mașină care are o turbină masivă.

Figura 11: HBFS Fan linefollower

<https://hbfsrobotics.com/linefollower>

HBFS Robotics crează un linefollower care învinge gravitația.





Figura 12: McMurtry Spéirling Fan car

<https://mcmurtry.com/speirling-pure>

McMurtry Spéirling este primul vehicul de producție limitată care reușește să doboare recordul deținut de o mașină de Formula 1 pe pista TOP Gear.

## 2.7 Senzorul LiDAR



Figura 13: LiDAR Tf Mini Plus

<https://en.benewake.com/TFminiPlus/index.html>

În anul precedent am testat senzorul ultrasonic **HC-SR04** care are o rată de achiziție de **40Hz**. Ca să reușim să acoperim un unghi mare în fața mașinii am folosit trei senzori de același tip. Problemele pe care le-am întâlnit în mod repetat au fost detecțiile eronate (**false-positives**). Lidarul rotativ a fost următoarea idee, însă costurile ridicate de achiziție m-au împins rapid spre o variantă mai accesibilă. Astfel am ajuns la soluția unui stil de radar compus dintr-un **TF MINI Plus** și un servomotor care acoperea un unghi de 20 de grade. De menționat este faptul că Tf mini plus are o rată de achiziție între 1-1000 Hz.

## 2.8 Modul step-down

Modulul stepdown a fost ales ca să satisfacă cerințele minime de curent pentru întregul sistem. **Debix** folosește **maxim 3A**, Camera și Teensy ul sunt alimentate direct din SBC deci nu le luăm în considerare. Motoarele au fost conectate direct la baterie deci nu au trecut prin modulul

step-down. Lidarul folosește maxim 150mA. Servomotoarele pot folosi curenti pana la 2A in cazul unui blocaj dar se pot și arde. Având doua totalul de consum maxim posibil este de 7A. Așa am ales să folosesc un step down care duce maxim 12A.



Figura 14: Modul Step-Down

<https://sigmanortec.ro/>

Modul-sursa-coboratoare-reglabila-12A-0-8-30V-100W-p158720051

### 3 Modelarea 3D a șasiului

În anul precedent am avut un kit dedicat pentru robot. Am beneficiat de componente ușoare din plastic și șasiu fabricat din aluminiu. În ziua concursului cu câteva ore înainte am rupt un braț de direcție și am avut mari emoții. Cea mai mare problemă pe care am întâlnit-o a fost numarul redus de piese de schimb și libertatea inexistentă în designul vehiculului.

Anul acesta am decis sa modelez totul in 3D pentru a crea o soluție unică, personalizată nevoilor mele. Am reușit să integrez în design:

- Mecanism de direcție Ackermann, care ajută la virare
- Un diferențial mecanic
- Sistem de creare vid cu EDF

**Lista componentelor prefabricate:**

- 1 x Diferențial mecanic
- 4 x Roți RC car

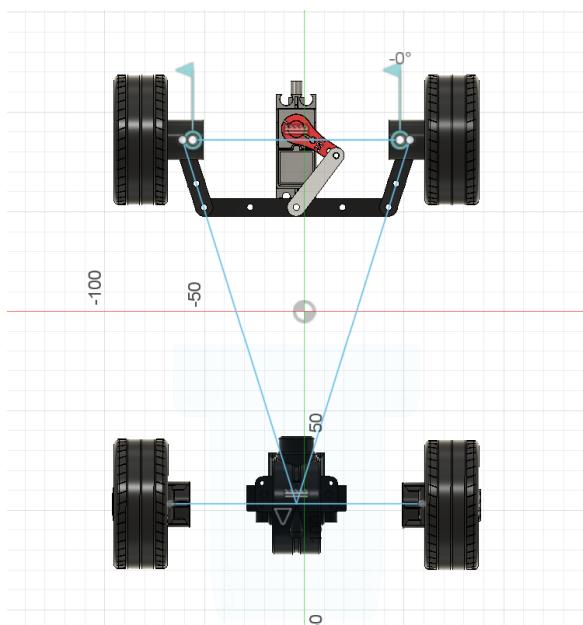
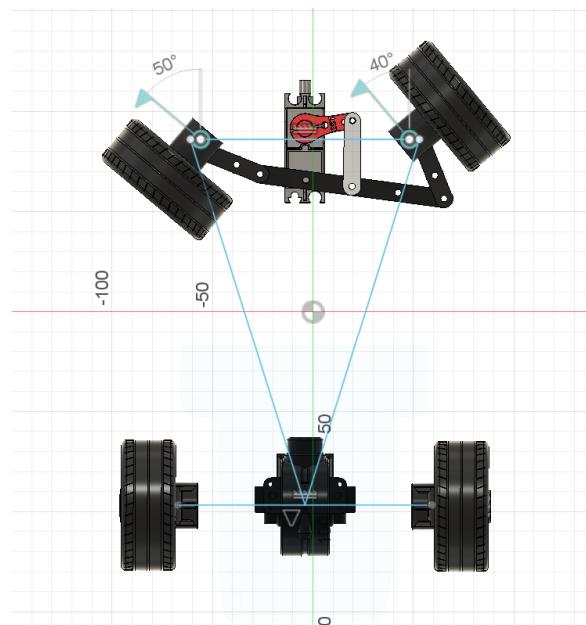
**Dimensiuni importante pe șasiu:**

- 104mm între pivotii față
- 183mm între axul față și axul spate



Figura 15: 3D Assembly Design

### 3.1 Mecanism de direcție Ackermann

Figura 16:  
Ackermann setupFigura 17:  
Ackermann angles

În intuitiv în dezvoltarea șasiului am considerat că roțile din față ale mașinii se învârt la același unghi, însă în testele facute, o roată mereu aluneca și am ajuns rapid la concluzia că am pornit cu o presupunere greșită. Astfel am gasit teoria de geometrie dezvoltată de Ackermann. Dupa relaționările găsite de acesta roata din interiorul curbei se deplasează pe un cerc cu o rază mai mică decât cea din exterior[1]. Având raze diferite, se observă automat și **unghiul roților**, acestea diferă (**50°** respectiv **40°**). Pentru a avea acest stil de geometrie trebuie poziționate punctele de pivotare pe o linie între pivot și centrul punctii spate.

Teoretic în modelul cinematic tip bicicletă, patru roți sunt reduse la doar două roți. Pentru a vedea exact arcul de cerc pe care se învârte vehiculul am folosit o rigla de măsurare a unghiului poziționată exact pe locația virtuală a roții din mijloc. Intervalul [30; 0; -30] reprezintă valorile maxime presele servomotor, iar unghiul maxim de viraj al roților este [50°; 0°; -50°]. Tot ce este pozitiv are direcție spre stanga, tot ce este negativ are direcție spre dreapta. Relația dintre prescrierea servomotorului și unghiul mașinii arată în felul următor:

$$\text{servoValue} = \frac{3}{5} \cdot \text{steeringAngleDegrees}$$

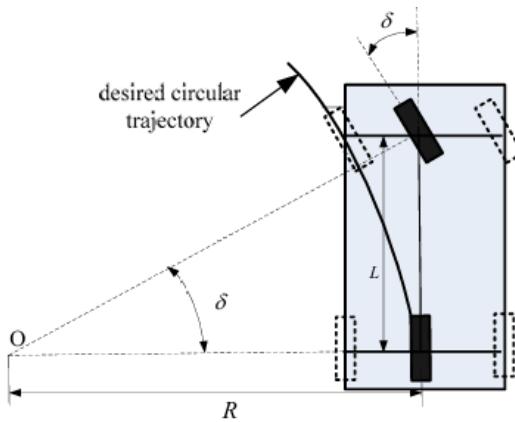
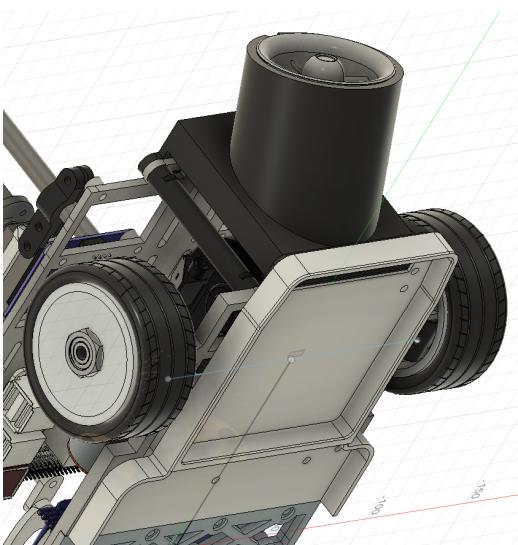


Figura 18: Bycicle Model [4]

### 3.2 Detaliere EDF

Figura 19:  
Final Edf SetupFigura 20:  
First Edf Setup

Într-o primă iteratie am inceput cu o arie foarte mare dedesubtul mașinii. Am dorit să creez cât mai mult vid posibil. Din cauza unui singur ventilator a trebuit să cobor pragurile. Din cauza pragurilor coborâte uneori mașina rămânea suspendată, cu o roată în aer (când părăsea pistă în momentele de calcul eronate). În a doua iteratie am redus aria în care se poate crea vidul. Această zonă a fost specific pusă sub axul spate al mașinii deoarece fără edf mașina derapa cu roțile din spate. Dacă înainte era necesar ca pragurile mele să atingă solul, acum era destul loc cat să nu mai rămână suspendată iar performanțele au rămas la același nivel. Cu noua configurație am reușit să cresc **distanța dintre praguri și sol de la 0mm la 2mm**.

## 4 Arhitectura software

Codul de bază este scris în C++. Am ales astfel deoarece am dorit să îmi imbunătățesc cunoștiințele în acest limbaj. Deși OpenCV este bine optimizat și usor de folosit în Python, m-am adaptat foarte rapid la modul de folosire în C++. În cadrul dezvoltării m-am lovit de probleme de librărie pentru modulul LCD, astfel am decis să scriu codul dedicat meniului în Python. Ulterior, datorită simplicității am ales Python pentru vizualizarea datelor.

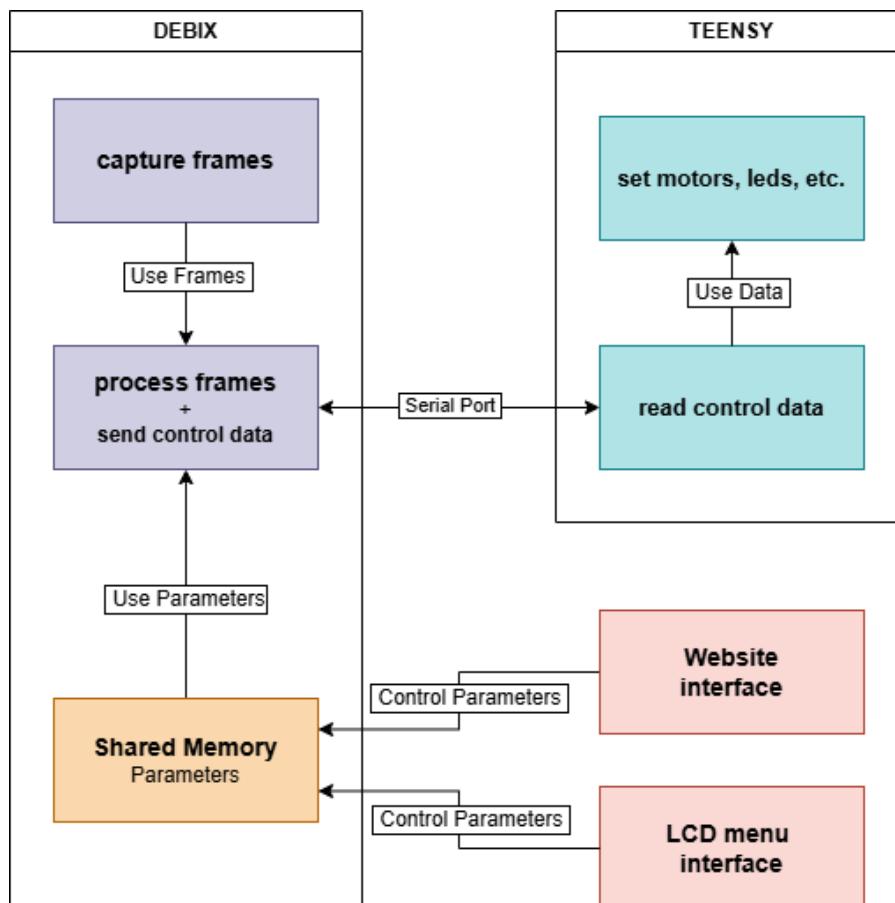


Figura 21: Procesul general

Figura 21 prezintă cel mai abstract mod de reprezentare a procesului întreg. În următoarea etapă vom discuta mai detaliat despre firele de execuție, iar mai apoi de procesul de prelucrare a imaginii.

#### 4.1 Organizarea firelor de execuție

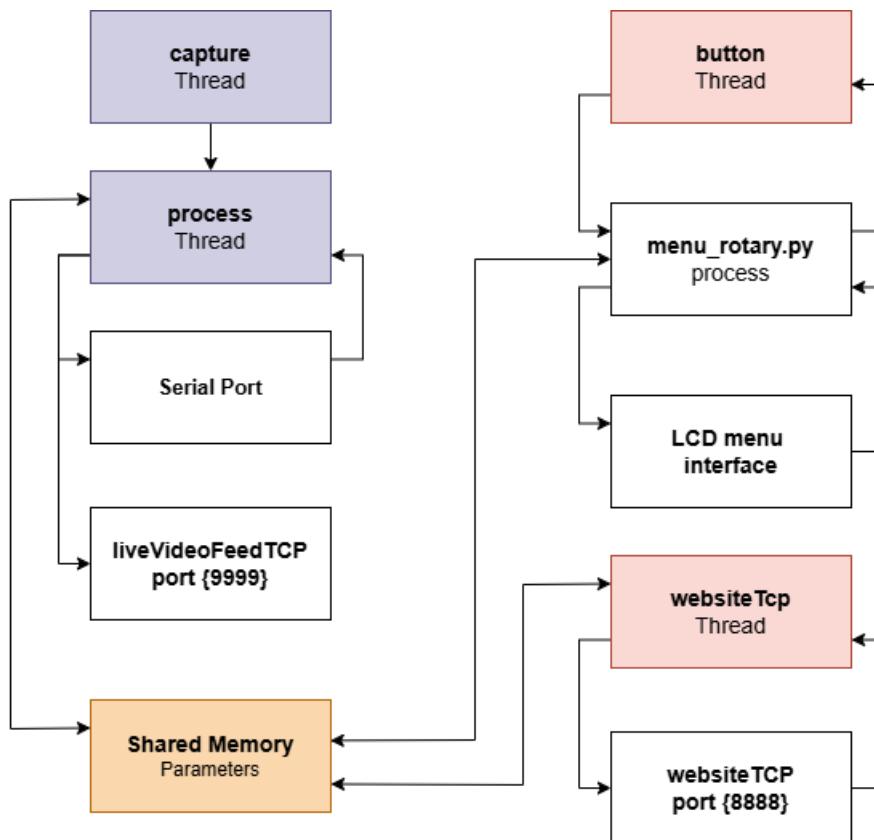


Figura 22: Firele de execuție

În tot procesul avem patru fire de execuție.

**"CaptureThread"** se ocupă cu captarea imaginilor. Prin ajutorul unui mutex le salvează într-o variabilă globală. Un mutex este un mecanism de sincronizare care ajută ca firele de execuție să nu acceseze aceeași resursă în mod simultan. [6]. Pentru conveniență există și opțiunea de testare direct pe imagini deja stocate. Obiectul "global\_camera" este inițializat la inceput. Imaginea **capturată** este de  $320 \times 240$  de pixeli setată la **100fps**-uri (cadre pe secundă) stabilă. Pentru a ne asigura că ulterior procesul de **prelucrare** este unul rapid reducem cadrul transmis la  $200 \times 150$  de pixeli. Performanțele cresc notabil de la 40fps la **70fps** în urma **redimensionării**.

**"ProcessThread"** este cel care face prelucrarea imaginii (acest proces are dedicat un capitol

separat). Threadul trimite imagini către zona de debug, websiteul(\*), prin "liveVideoFeedTCP" și date de control către Teensy prin poarta serială.

**INFO(\*):** De notat este faptul că la concurs nu este permisă nici o conexiune externă cu vehiculul. El trebuie să opereze în mod independent, total autonom.

Formatul detaliilor trimise arată în felul următor:

1;1;15;220;0;50;1;1;0;200;1182

<enableCarEngine>;<enableCarSteering>;<steeringAngleServo>;<speed>;  
<isFinishLineDetected>;<distanceBeforeIssuesAppear>;<isRadarEnabled>;  
<isBlueStatusLedOn>;<isYellowStatusLedOn>;<currentEdffFanSpeed>;checksum

"**buttonThread**" așteaptă apăsarea butonului de pe encoderul rotativ citit pe liniile GPIO, mai apoi poernește procesul de python care prezintă un meniu interactiv pentru schimbarea parametrilor pe LCD. Codul de python accesează Shared Memory-ul pentru a prelua datele de config.

```
shared_mem.seek(0)
shared_mem_values = struct.unpack(SHM_FORMAT, shared_mem.read(SHM_SIZE))
shared_mem_values = list(shared_mem_values)
```

Figura 23: Secvență din menu\_rotary.py

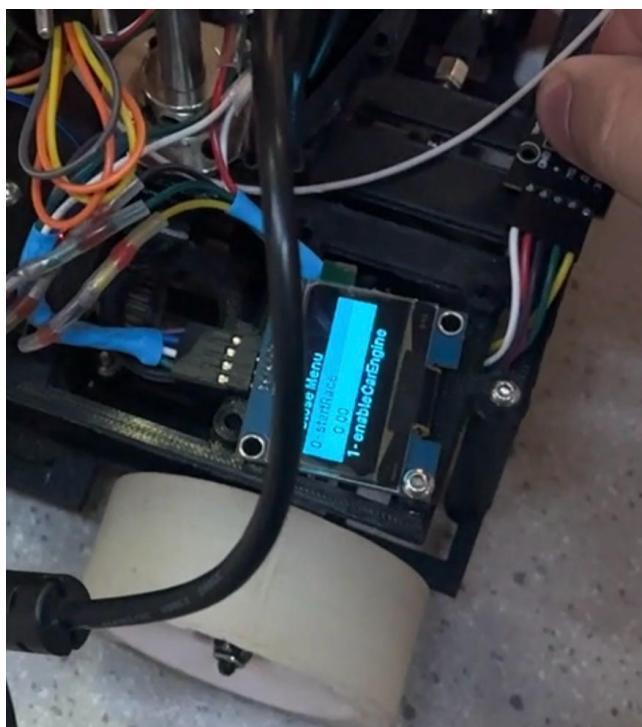


Figura 24: Meniu

"websiteTCPThread" poate primi trei comenzi prin TCP:

- STOP (folosit în caz de urgență ex. mașina ieșe de pe pistă)
- READ
- WRITE

Toate configurările se trimit și se citesc într-un format json. De asemenea, există și o verificare de siguranță în momentul scrierii pentru a ne asigura că nu scriem valori eronate.

```
while (true) {
    std::string command = connection.receiveStringData();

    if (command == "STOP")
    {
        global_config->enableCarEngine = 0;
        global_config->currentEdffFanSpeed = 0;
    }
    else if(command == "READ")
    {
        SharedConfig* raw_config = global_config.get();
        std::string jsonStr = json(*raw_config).dump();
        connection.sendStringData(jsonStr + "\n");
        std::cout << "[Website] Sent config JSON.\n";
    }
}
```

```
else if (command.rfind("WRITE:", 0) == 0) []
{
    std::string jsonStr = command.substr(6);
    try {
        SharedConfig newConfig = json::parse(jsonStr);
        *global_config = newConfig;
        std::cout << "[Website] Updated config from web.\n";
    } catch (const std::exception& e) {
        std::cerr << "[Website] Failed to parse config: " << e.what() << std::endl;
    }
}
else
{
    std::cerr << "[Website] Unknown command: " << command << std::endl;
}
```

Figura 25: Secvență websiteTcpThread

"**Shared Memory**" a fost folosit pentru a stabili o conexiune de date între procesul de C++ și procesul Python.

```

struct SharedConfig {
    int startRace;                                //Range: 0 - 1
    int enableCarEngine;                            //Range: 0 - 1
    int enableCarSteering;                          //Range: 0 - 1
    int enableCameraThresholdCheck;                //Range: 0 - 1
    int enableFinishLineDetection;                 //Range: 0 - 1
    int currentState;                             //Range: 0 - 3
    int thresholdValue;                           //Range: 0 - 255
    int distanceErrorFromChassis;                //Range: -240 - 240
    int lineMinPixelCount;                         //Range: 0 - 255
    int distanceSensorError;                      //Range: 0 - 30
    int stoppingDistanceBoxFrontEnd;              //Range: 1 - 30
    int interpolatedPointsSetup;                  //Range: 0 - 1
    char _padding[8];
    double calibrateTopLinePerc;                  //Range: 0 - 100
    double calibrateBottomLinePerc;                //Range: 0 - 100
    double trackLaneWidthOffset;                  //Range: -100 - 200
    double topImageCutPercentage;                 //Range: 0 - 1
    double topCutOffPercentageCustomConnected;    //Range: 0 - 1
    double lineStartPointY;                        //Range: 0 - 1
    double line90DegreeAngleRange;                 //Range: 0 - 90
    double finishLineAngleRange;                  //Range: 90 - 180
    double servoTurnAdjustmentCoefficient;        //Range: 0 - 5
    double corneringSpeedCoefficient;             //Range: 0 - 2
    double minSpeed;                             //Range: 0 - 350
    double maxSpeed;                            //Range: 0 - 350
}

```

Figura 26: Shared Memory 1

```

double minSpeedAfterFinish;                     //Range: 0 - 350
double maxSpeedAfterFinish;                    //Range: 0 - 350
double currentEdfFanSpeed;                   //Range: 0 - 350
double curvatureFactor;                       //Range: 0 - 200
double rdp_epsilon;                           //Range: 0 - 25
double minAngleLookAheadReference;            //Range: 0 - 180
double maxAngleLookAheadReference;            //Range: 0 - 180
double minLookAheadInCm;                      //Range: 0 - 100
double maxLookAheadInCm;                      //Range: 0 - 100
double waitBeforeStartSeconds;                //Range: 0 - 10
double straightWheelTimerSeconds;             //Range: 0 - 5
};


```

Figura 27: Shared Memory 2

În practică codul python citește fișierul config.h și cu o expresie regulată citește numele parametrilor care au **int** sau **double** și aria de funcționare (range-ul). Expresiile regulate sau regex este un şablon matematic de potrivire a textului[11].

#### 4.1.1 Structura Process Frames

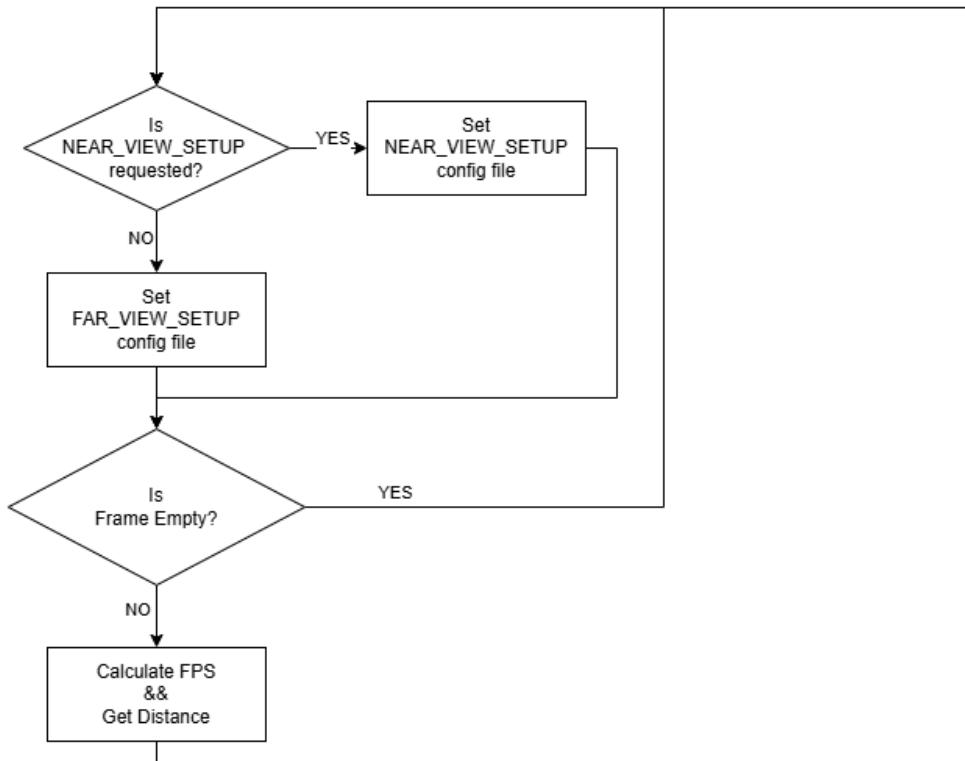


Figura 28: processFramesThread\_1

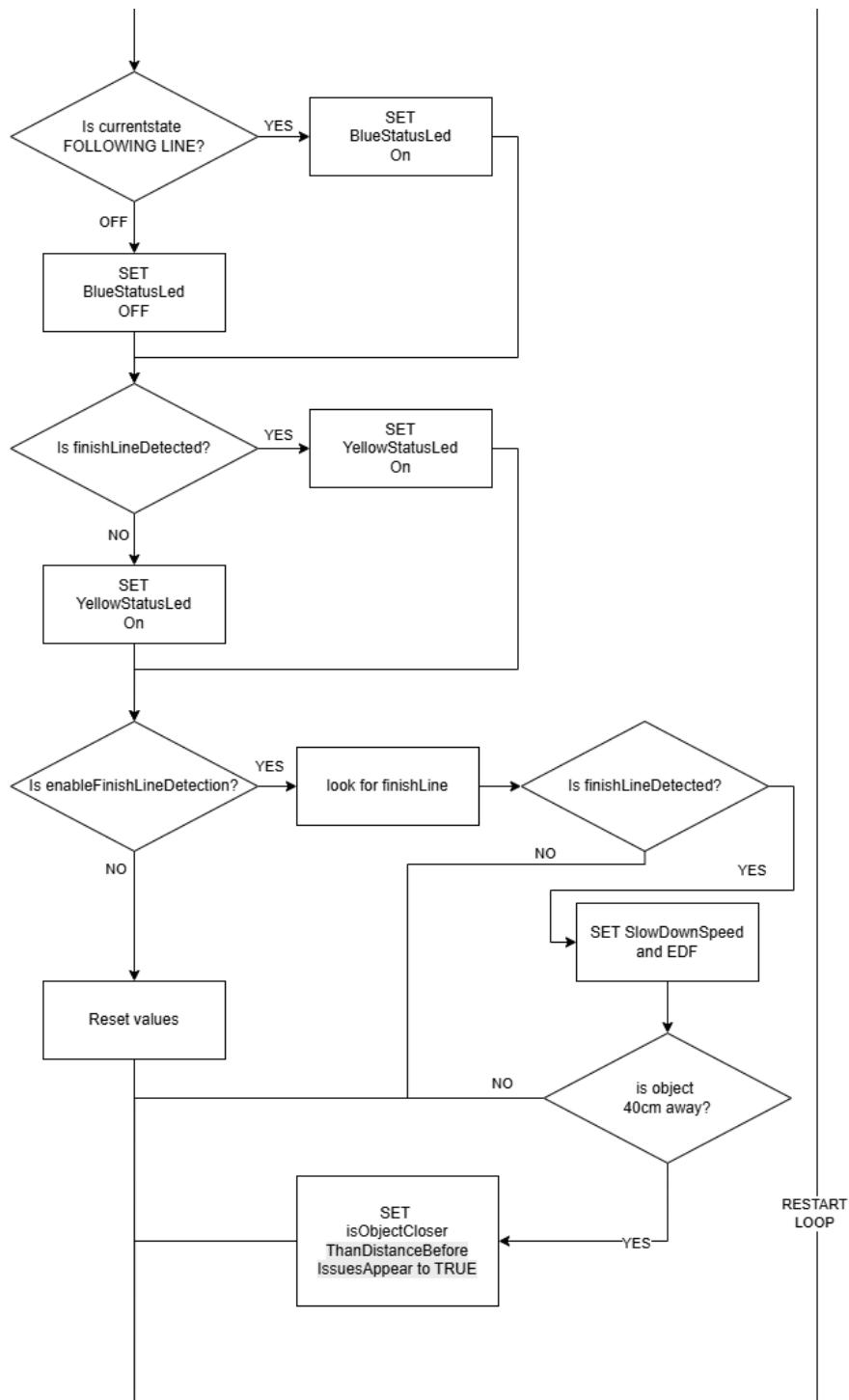


Figura 29: processFramesThread\_2

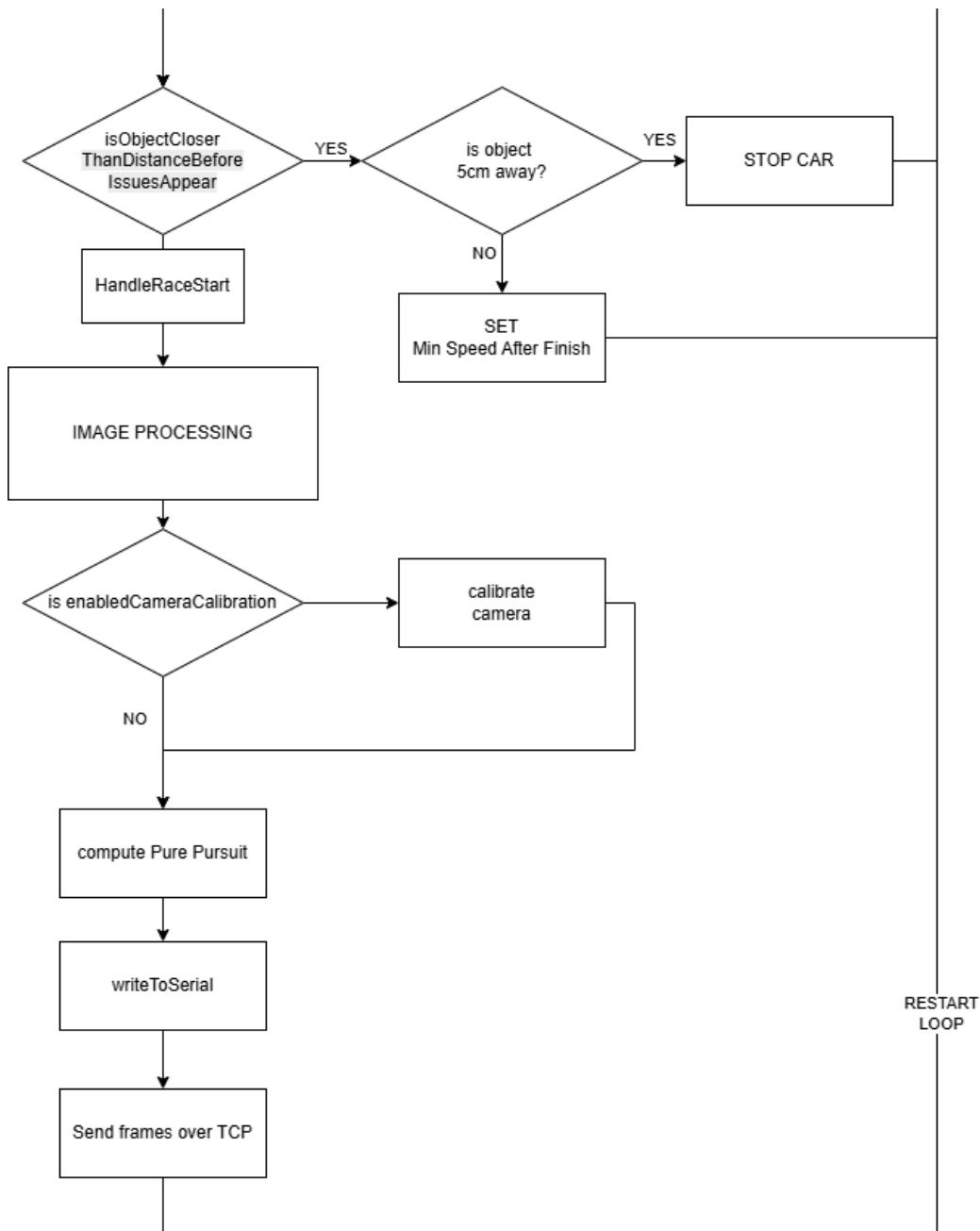


Figura 30: processFramesThread\_3

Threadul process frames conține tot calculul principal și modul în care acesta se execută. Blocul **Image Processing**(Prelucrarea imaginilor) și **compute Pure Pursuit** vor fi detaliate în capitolele următoare.

## 4.2 Progresul prelucrării imaginilor

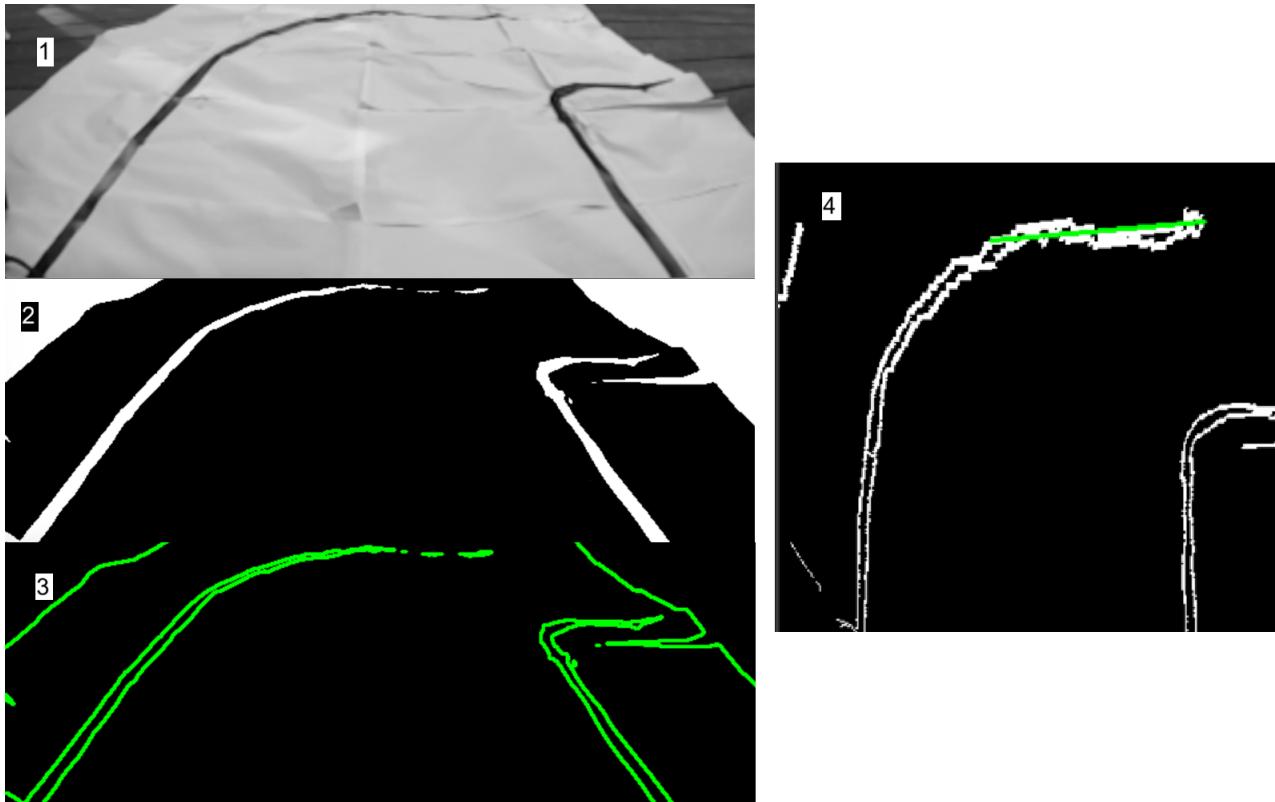


Figura 31: Canny și BirdEyeView

Imaginea este capturată în format  $200 \times 150$  de pixeli RGB (3 canale). În cadrul procesării de date trecem peste 90.000 de valori. Pentru a simplifica calculele la doar 30.000 de valori folosim doar un singur canal, Grayscale, adică Alb Negru (imagină 1 din figura 25). În documentația OpenCV în capitolul **Color Conversions** funcția de conversie este [9]:

$$\text{RGB}[A] \text{ to Gray: } Y \leftarrow 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \quad (1)$$

```
cv::cvtColor(frame, frame, cv::COLOR_BGR2GRAY);
```

Figura 32: cvtColor

Dupa acest pas aplicăm un "threshold binar" numit si threshold simplu.

Pentru a păstra matrici aproape goale, inversăm imaginea cu un bitwiseNot ca liniile să fie cele albe, iar restul negru(imaginea 2 din figura 25, respectiv figura 27).

1 1 1 1 1 1 1 1	0 0 0 0 0 0 0
1 1 1 1 1 1 1 1	0 0 0 0 0 0 0
1 1 1 1 1 1 1 1	0 0 0 0 0 0 0
0 0 0 0 0 0 0 0	---bitwise-not---
1 1 1 1 1 1 1 1	1 1 1 1 1 1 1
1 1 1 1 1 1 1 1	0 0 0 0 0 0 0
1 1 1 1 1 1 1 1	0 0 0 0 0 0 0
1 1 1 1 1 1 1 1	0 0 0 0 0 0 0

Figura 33: bitwise not

Ulterior am aplicat functia Canny gasită sub titlul "Canny Edge Detector" în documentație. Ea găsește marginile exterioare a unei forme. Nu vom intra în detaliu deoarece am folosit metoda thinning/skeletonizing în final.

Transformarea imaginii din poziția camerei în poziția bird eye (top down) view se poate observa în imaginea 4 din figura 25. În cazul acesta am realizat că dacă trebuie să translatez forme geometrice, **îmaginea se deformează în detrimentul detaliilor importante ca și curbele**. Transformarea va fi detaliată ulterior.

#### 4.2.1 Simple vs Otsu vs Adaptive Gaussian Threshold

Am examinat 3 metode de thresholding:

- Simple threshold
- Otsu threshold
- Adaptive Gaussian threshold

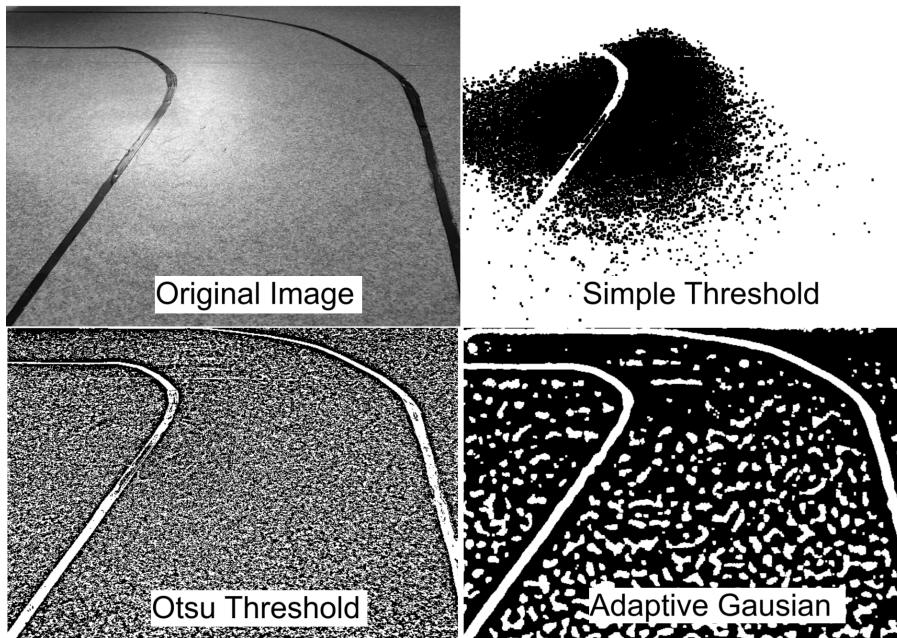


Figura 34: Metode de Threshold

## Threshold Simplu

Conform capitolului **Basic Thresholding Operations** daca un pixel are valoarea mai mare decat pragul setat de noi alege valoarea maximă, în caz contrar valoarea minimă [9]:

$$\text{dst}(x, y) = \begin{cases} \text{maxVal} & \text{if } \text{src}(x, y) > \text{thresh} \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

```
cv::threshold(frame, thresholdFrame, config->thresholdValue, maxThresholdValue, cv::THRESH_BINARY);
```

Figura 35: Simple Threshold

## Otsu threshold

Otsu threshold comparativ cu cel simplu (detaliat anterior) calculeaza valoarea de comparație automat pentru fiecare imagine. Valoarea aceasta de prag este calculată pe baza histogramei imaginii iar ea este alesă în asa fel încât să despartă cel mai bine fundalul de obiect.

```
cv::threshold(frame, thresholdFrame, 0, 255, cv::THRESH_BINARY | cv::THRESH_OTSU);
```

Figura 36: Otsu Threshold

Pentru a determina pragul optim de binarizare, algoritmul Otsu caută să **minimizeze varianța în interiorul claselor** (within-class variance), sau echivalent să **maximizeze varianța între clase**[8].

## Varianța totală în interiorul claselor

$$\sigma_w^2(t) = q_1(t) \cdot \sigma_1^2(t) + q_2(t) \cdot \sigma_2^2(t) \quad (3)$$

**Interpretare:** Scopul este ca aceste clase să fie cât mai compacte (varianță mică în interior), deci separate cât mai bine.

## Probabilitățile claselor

$$q_1(t) = \sum_{i=1}^t P(i) \quad q_2(t) = \sum_{i=t+1}^I P(i) \quad (4)$$

**Interpretare:**  $q_1(t)$  și  $q_2(t)$  reprezintă **proporția de pixeli** care aparțin fiecărei clase:

- $q_1(t)$  — clasa fundalului (pixeli cu nivel de gri  $\leq t$ )
- $q_2(t)$  — clasa obiectului (pixeli cu nivel de gri  $> t$ )

### Media fiecărei clase

$$\mu_1(t) = \frac{\sum_{i=1}^t i \cdot P(i)}{q_1(t)} \quad \mu_2(t) = \frac{\sum_{i=t+1}^I i \cdot P(i)}{q_2(t)} \quad (5)$$

**Interpretare:**  $\mu_1(t)$  și  $\mu_2(t)$  sunt **mediile nivelurilor de gri** pentru fiecare clasă. Acestea indică unde se află **centrul** fiecărei clase în histogramă.

### Varianța fiecărei clase

$$\sigma_1^2(t) = \frac{\sum_{i=1}^t [i - \mu_1(t)]^2 \cdot P(i)}{q_1(t)} \quad \sigma_2^2(t) = \frac{\sum_{i=t+1}^I [i - \mu_2(t)]^2 \cdot P(i)}{q_2(t)} \quad (6)$$

**Interpretare:**  $\sigma_1^2(t)$  și  $\sigma_2^2(t)$  măsoară **cât de ”împrăștiate”** sunt valorile fiecărei clase în jurul mediei sale. O varianță mică înseamnă că pixelii din clasă sunt bine separați de cealaltă clasă.

**Rezumat:** algoritmul Otsu încearcă toate pragurile  $t$  posibile și selectează pragul care **minimizează**  $\sigma_w^2(t)$  — adică asigură o separare optimă între fundal și obiect.

**INFO:** Pe parcursul testelor am folosit *threshold simplu*, însă la concurs datorită luminii neuniforme am folosit Otsu threshold.

## Adaptive Gaussian threshold

Fiecare pixel are calculat un prag local în funcție de vecinii acestuia (blockSize). În loc de o simplă medie a pixelilor selectați, se face o pondere Gausiana, vecinii apropiati de centru contând mai mult. Rezultatele sunt impecabile.

```
cv::adaptiveThreshold(frame, thresholdFrame, 255, cv::ADAPTIVE_THRESH_GAUSSIAN_C,
| | cv::THRESH_BINARY, blockSize, constantValue);
```

Figura 37: Adaptive Gaussian Threshold

Pentru fiecare pixel  $(x, y)$ , pragul local  $T(x, y)$  se calculează astfel:

$$T(x, y) = \mu_G(x, y) - C \quad (7)$$

unde:

- $\mu_G(x, y)$  — media Gaussiană a vecinilor pixelului  $(x, y)$ , calculată într-o fereastră de dimensiune  $N \times N$  (ex.  $30 \times 30$ ).
- $C$  — o constantă care se scade din medie (ex. 11).

$$\text{pixel}(x, y) = \begin{cases} \text{maxValue}, & \text{dacă } I(x, y) > T(x, y) \\ 0, & \text{altfel} \end{cases} \quad (8)$$

unde  $I(x, y)$  este valoarea inițială a pixelului  $(x, y)$ .

### **Concluzie finală:**

Numarul total de pixeli:  $320 \times 240 = 76.800$

Numarul total de pixeli după reducere:  $200 \times 150 = 30.000$

Metodă	Complexitate
Simple Threshold	$O(N)$
Otsu Threshold	$O(N)$
Adaptive Gaussian Threshold	$O(N \cdot B)$

Tabela 1: Complexitatea algoritmilor ( $N$  = număr pixeli,  $B$  = blockSize<sup>2</sup>)

În cazul Adaptive Gaussian deși are rezultate foarte bune, timpul de execuție nu este cel mai bun. Pentru a crește performanțele de procesare și pentru a câștiga fps-uri am decis să continui cu Threshold simplu deoarece avea rezultate satisfăcătoare.

Număr de operații pentru Adaptive Gaussian:  $(30 \times 30) \times 30.000 = 27.000.000$

27.000.000 de operații comparativ cu 30.000 este foarte mult. Deși OpenCV are foarte multe metode de eficientizare, tot nu reușim să câștigăm performanțe considerabile.

#### 4.2.2 Thinning/Skeletonizing

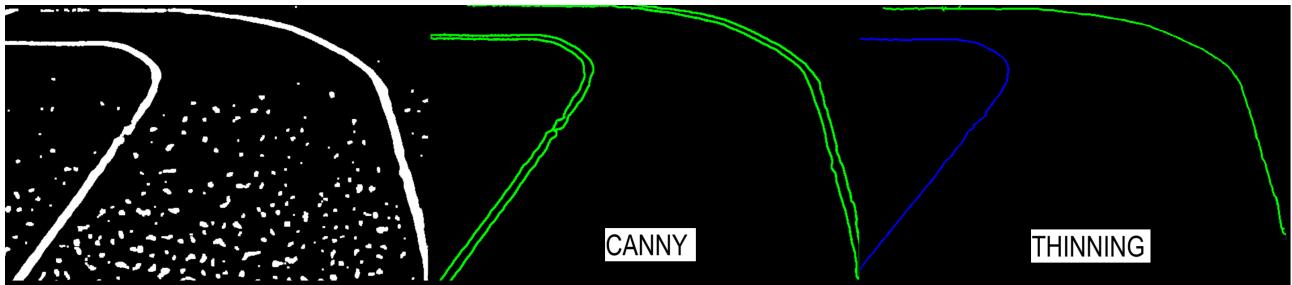


Figura 38: Canny vs Thinning

Algoritmul de thinning creat de ZHANG-SUEN a fost inclus, iar ulterior optimizat în OpenCv. Teoretic funcționează ca decojirea unei cepe. Se substrage strat cu strat până ajungem la linia din mijloc de 1px.

```
cv::Mat CameraProcessing::skeletonizeFrame(cv::Mat& thresholdedImage) {
    cv::Mat skeleton;
    cv::ximgproc::thinning(thresholdedImage, skeleton, cv::ximgproc::THINNING_ZHANGSUEN);
    return skeleton;
}
```

Figura 39: Thinning

#### 4.2.3 Bird Eye View

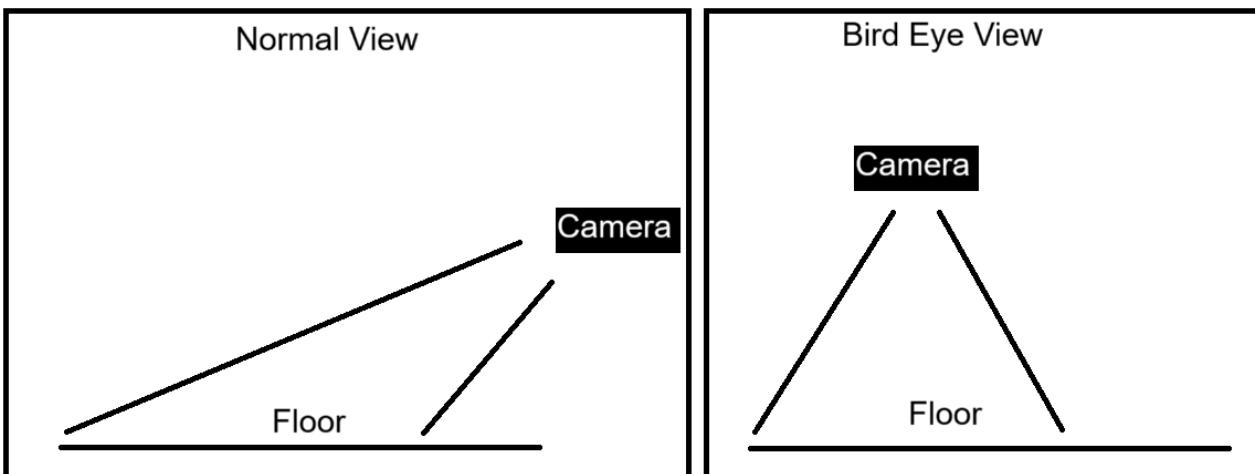


Figura 40: Normal View vs Bird Eye View

Bird Eye View este o formă de vizualizare a obiectelor de deasupra. Ea ne ajută în perceptia corectă a unghiurilor, în mod special a celor de 90°. În cadrul pistei am ales 4 puncte sursă plasate pe cele două linii, la o distanță de 55cm una de alta. Punctele formează un pătrat perfect. Mai apoi în imaginea translatătă am ales alte 4 puncte destinație care ne formează vizual pătratul și pe imaginea percepță de program. Ulterior în calculele specifice pentru unghi și direcție folosim imaginea translatătă.

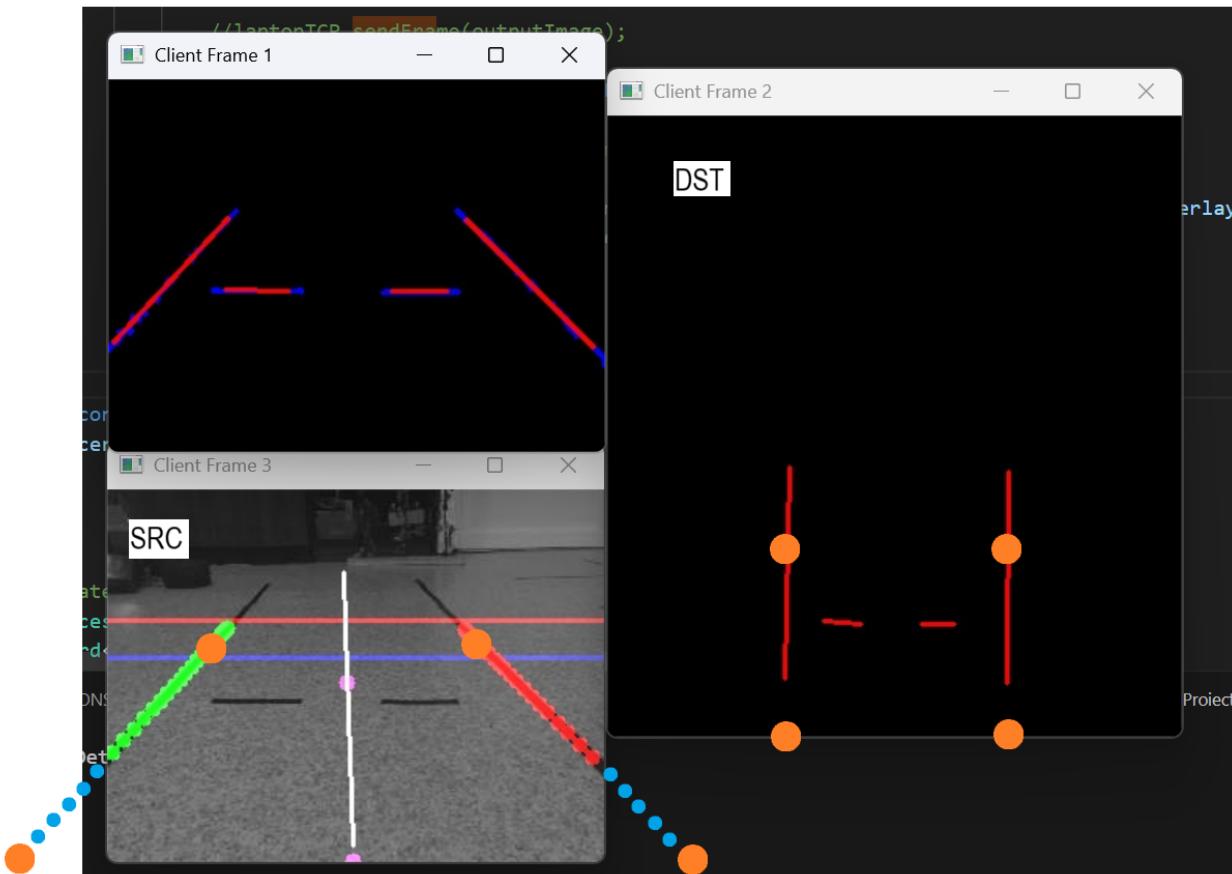


Figura 41: Bird Eye View Points

Pentru a putea duce la bun sfârșit aceste calcule OpenCV ne oferă două funcții importante. Prima funcție, **cv::getPerspectiveTransform()**, ne oferă o matrice de translatare a punctelor care se obține din punctele sursă și destinație[7]. În cadrul ecuației 9 această matrice este chiar "map\_matrix".

$$\begin{bmatrix} t_i x'_i \\ t_i y'_i \\ t_i \end{bmatrix} = \text{map\_matrix} \cdot \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \quad (9)$$

unde:  $dst(i) = (x'_i, y'_i)$ ,  $src(i) = (x_i, y_i)$ ,  $i = 0, 1, 2, 3$

```
this->MatrixBirdsEyeView = cv::getPerspectiveTransform(srcPoints, dstPoints);
this->MatrixInverseBirdsEyeView = this->MatrixBirdsEyeView.inv();
```

Figura 42: getPerspectiveTransform()

Astfel putem translata puncte din sursă în destinație prin matricea de transformare dar și din destinație în sursă prin inversa ei. A doua funcție, **cv::perspectiveTransform()**, înmulțește matricial punctul sursă și matricea de translatare pentru a obține noul punct. În cadrul programului avem două alte funcții care înglobează aceasta utilitate:

```
cv::Point2f CameraProcessing::perspectiveChangePoint(const cv::Point2f& point, const cv::Mat& transformMatrix)
{
    std::vector<cv::Point2f> src = {point};
    std::vector<cv::Point2f> dst;
    cv::perspectiveTransform(src, dst, transformMatrix);
    return dst[0];
}

std::vector<cv::Point2f> CameraProcessing::perspectiveChangeLine(const std::vector<cv::Point2f>& line,
                                                               const cv::Mat& transformMatrix)
{
    std::vector<cv::Point2f> transformedLine;
    cv::perspectiveTransform(line, transformedLine, transformMatrix);
    return transformedLine;
}
```

Figura 43: perspectiveChange(Point/Line)

Pentru concurs am ales să salvez două setup-uri pentru această mașină:

- interpolated\_points\_NEAR
- interpolated\_points\_FAR

**INFO:** Punctul (0, 0) al imaginii se află în colțul din stânga sus, iar punctul (width, height) se află în colțul din dreapta jos.

În cadrul figurii 33 "SRC" se poate observa cum punctele de jos sunt luate fix de pe linia cea mai joasă a frameului. Dacă imaginea are înălțimea de 150px atunci punctele de sus se află pe rândul 60 iar cele de jos pe rândul 149. Translatarea, figura 33 "DST", se face tot pe ultimul rând al imaginii. Setupul cu punctele "Near" funcționează exact în acest fel.

În cazul setupului cu punctele "Far" cele 4 puncte sursă sunt procurate de mai sus de pe linie. ex. rândul 50 și 139. În practică acestă configurație face sistemul să percepă eronat distanță, considerând că punctele îndepărțate se află mai aproape. Astfel vehiculul taie curbele, ceea ce într-un final poate însemna un avantaj.

### 4.3 Prelucrarea imaginilor

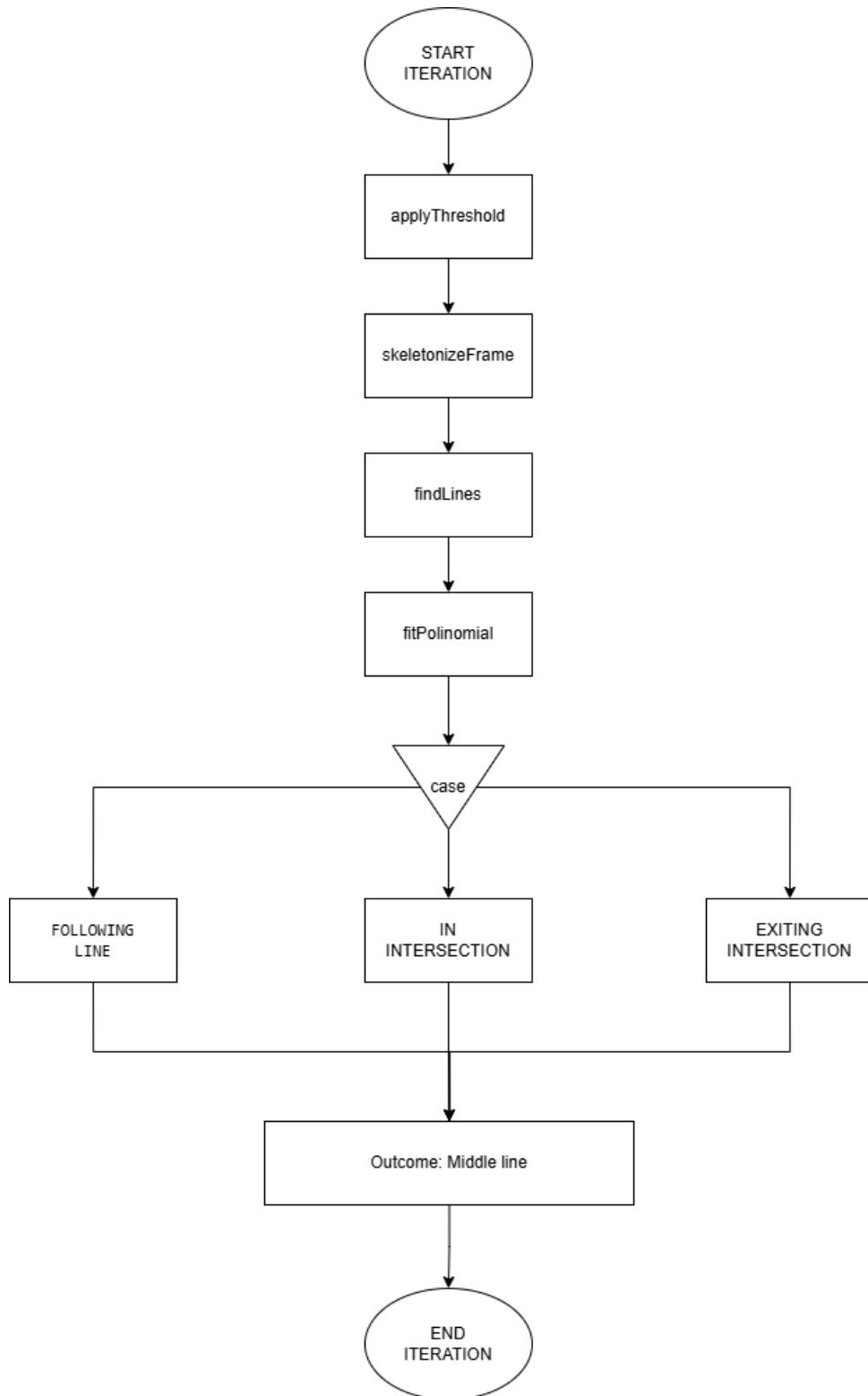


Figura 44: procesul prelucrării

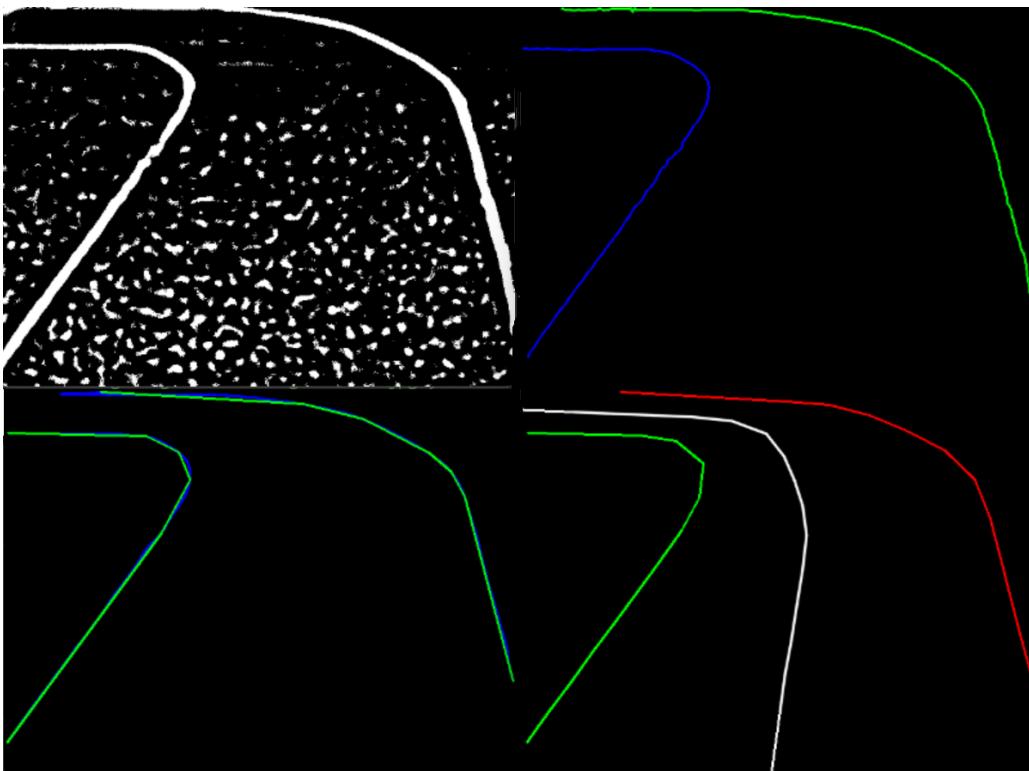


Figura 45: linia de mijloc

În cadrul capitolului anterior am comparat cele trei metode de thresholding, am discutat despre thinning și despre perspectiva Birds Eye View. În acest capitol vom prezenta o iteratie intreagă a procesului actual folosit în prelucrarea imaginii găsit în threadul process frames(figura 41) și rezultatul acestuia (figura 42).

Thresholdul a fost tratat în capitolul **"Simple vs Otsu vs Adaptive Guassian Threshold"** pe larg iar skeletonul a fost tratat în capitolul **"Thinning/Skeletonizing"**.

Vehiculul poate avea trei stări:

- FOLLOWING LINE
- IN INTERSECTION
- EXITING INTERSECTION

FindLines inglobează o funcție complexă numită customConnectedComponents. În cazul nostru maximul de linii admis este 4 deoarece în cazul cel mai amplu al pistei putem avea maxim 2 linii pe margine și 2 linii de finish pe mijloc (figura 38 DST).

```
// Function to find lines in a skeletonized frame
std::vector<std::vector<cv::Point2f>> CameraProcessing::findLines(const cv::Mat& thresholdedImage,
|   int currentState, const int rowTopCutOffThreshold)
{
    // Perform connected component analysis
    cv::Mat labels;
    std::vector<std::vector<cv::Point2f>> lines;
    int numLabels = customConnectedComponents(thresholdedImage, labels, 6, lines, currentState, rowTopCutOffThreshold);

    // Return only the first 4 lines (or fewer if there are less than 4)
    if (lines.size() > 4) {
        lines.resize(4);
    }

    return lines;
}
```

Figura 46: FindLines

CustomConnectedComponents este o funcție care caută toti pixelii legați unul de celalalt. După procesul de thinning se poate observa cu ochiul liber care pixeli alcătuiesc o linie și care pixeli nu. Căutarea începe în colțul din stânga jos în cazul normal, iar în alte cazuri începe din sânga sus.

Uneori se întâmplă ca imaginea capturată să aibă o linie întreruptă. Pentru a ne asigura că procesăm doar o linie întreagă, nu două, creăm o arie în care căutam pixeli. Pentru asta avem o funcție numită generateNeighborhood(radius).

```
// Function that generates a radius of values for customConnectedComponents
std::vector<cv::Point2f> generateNeighborhood(int radius)
{
    std::vector<cv::Point2f> neighbors;
    for (int dy = -radius; dy <= radius; ++dy) {
        for (int dx = -radius; dx <= radius; ++dx) {
            // Ensure point lies within the circle
            if (dx * dx + dy * dy <= radius * radius)
            {
                neighbors.emplace_back(dx, dy);
            }
        }
    }
    return neighbors;
}
```

Figura 47: generateNeighborhood

Aceasta primește ca și parametru o raza a unui cerc măsurată în pixeli. În cazul acesta particular se consideră o raza de 8 pixeli. Se caută într-un patrat al cărui rând indexat ar arăta astfel:

$$dx : -8, -7, -6, \dots, 0, \dots, 6, 7, 8 \rightarrow 17 \text{ valori}$$

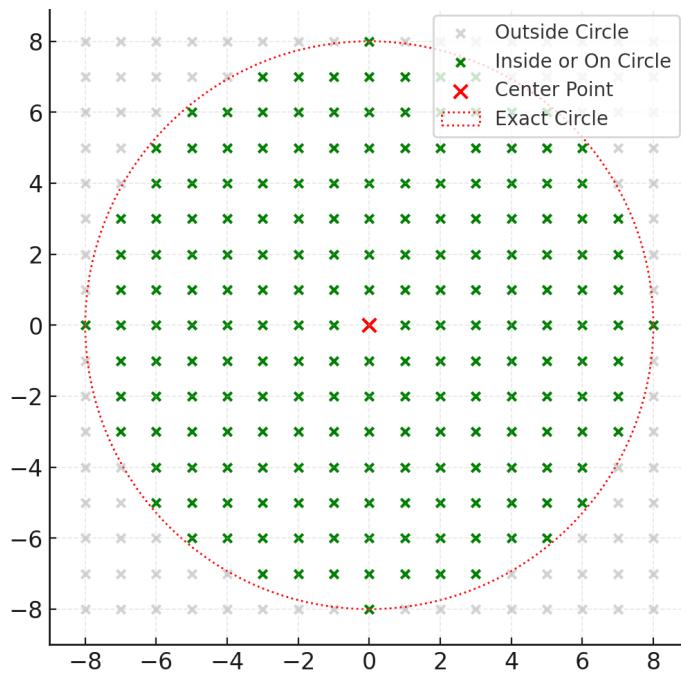


Figura 48: diagramă generateNeighborhood

Astfel dacă linia noastră se termină în punctul central marcat cu X roșu iar a doua parte a linii începe într-un X de pe cerc, ea va fi categorizată ca o linie.

```

int CameraProcessing::customConnectedComponents(const cv::Mat& binaryImage, cv::Mat& labelImage,
    int radius, std::vector<std::vector<cv::Point2f>>& lines, int currentState, const int rowTopCutOffThreshold)
{
    CV_Assert(binaryImage.type() == CV_8UC1);
    labelImage = cv::Mat::zeros(binaryImage.size(), CV_32S); // Initialize label matrix
    lines.clear(); // Clear any existing lines

    int label = 1; // Start labeling from 1
    std::vector<cv::Point2f> neighborhood = generateNeighborhood(radius); // Generate dynamic neighborhood
    int pixelCount;

    int startY, endY, stepY;
    if (EXITING_INTERSECTION == currentState || IN_INTERSECTION == currentState)
    {
        startY = rowTopCutOffThreshold;
        endY = binaryImage.rows - 1;
        stepY = +1; // Top to bottom
    }
    else
    {
        startY = binaryImage.rows - 1;
        endY = rowTopCutOffThreshold;
        stepY = -1; // Bottom to top
    }
}

```

Figura 49: CustomConnectedComponents\_1

FOLLOWING LINE se uită după liniile de 90 de grade și procesează imaginea din stânga jos în sus. Ca polyfit să funcționeze corect este nevoie ca punctele să fie corect sortate una după alta. Știind că liniile mereu merg de jos în sus, nu este nevoie de o sortare complexă. La ieșirea din intersecție însă ne întâlnim cu un caz special. În cazurile IN INTERSECTION și EXITING INTERSECTION un colț al intersecției în formă de V se procesează incorect.

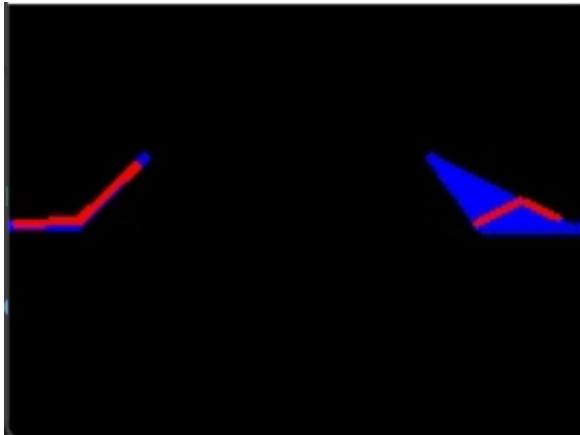


Figura 50: Colț procesat de jos în sus

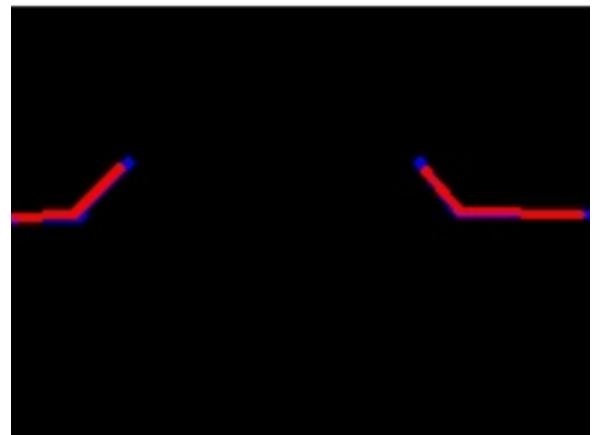


Figura 51: Colț procesat de sus în jos

Se procesează incorect deoarece algoritmul adaugă în listă odată un punct de pe o liniă albastră, iar ulterior un punct de pe linia verde. Pentru a ne asigura că punctele se iau una după alta am abordat acest stil de procesare în funcție de cazuri.

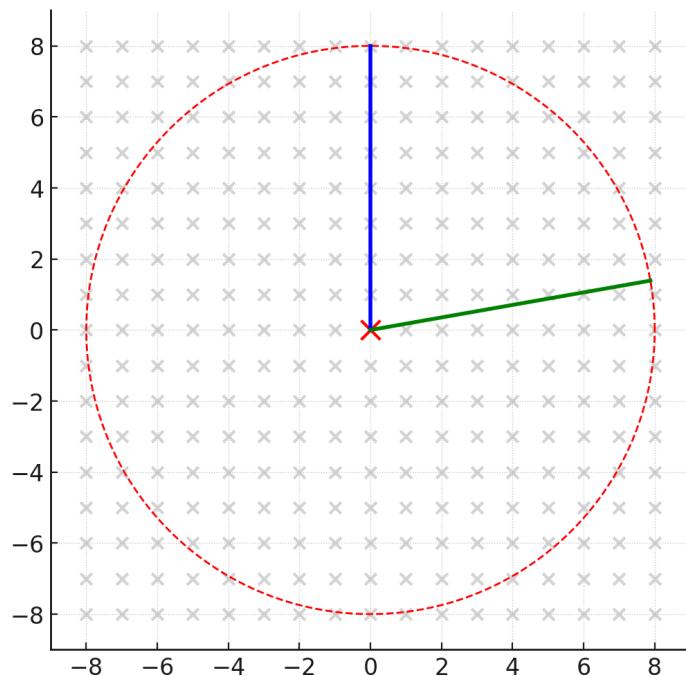


Figura 52: explicație caz V

```
for (int y = startY; (stepY > 0) ? (y <= endY) : (y >= endY); y += stepY)
{
    for (int x = 0; x < binaryImage.cols; ++x)
    {
        if (binaryImage.at<uchar>(y, x) == 255 && labelImage.at<int>(y, x) == 0)
        {
            std::queue<cv::Point2f> queue;
            queue.push(cv::Point2f(x, y));
            labelImage.at<int>(y, x) = label;

            pixelCount = 0;
            std::vector<cv::Point2f> componentPixels;
```

Figura 53: CustomConnectedComponents\_2

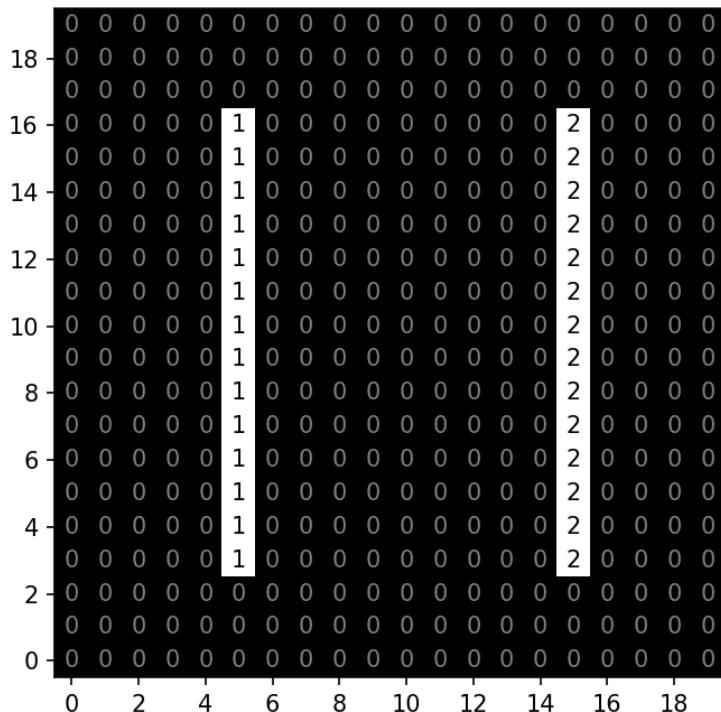


Figura 54: label Image

În cazul în care un punct alb este detectat iar acesta nu a fost deja procesat, label este 0, atunci începem procesul de prelucrare. Fiecare punct nou este adăugat în lista componentPixels.

```

        while (!queue.empty())
        {
            cv::Point2f p = queue.front();
            queue.pop();
            pixelCount++;
            componentPixels.push_back(p);

            for (const auto& offset : neighborhood)
            {
                int nx = p.x + offset.x;
                int ny = p.y + offset.y;

                if (nx >= 0 && ny >= rowTopCutOffThreshold &&
                    nx < binaryImage.cols && ny < binaryImage.rows)
                {
                    if (binaryImage.at<uchar>(ny, nx) == 255 &&
                        labelImage.at<int>(ny, nx) == 0)
                    {
                        labelImage.at<int>(ny, nx) = label;
                        queue.push(cv::Point2f(nx, ny));
                    }
                }
            }
        }
    }
}

```

Figura 55: CustomConnectedComponents\_3

**INFO:** Se poate observa că ne asigurăm ca linia detectată să nu treacă de `rowTopCutOffThreshold`. Dacă nu avem această verificare, gasim punctele înainte de cutoff însă algoritmul caută puncte și peste limită, ceea ce nu este neapărat indicat.

```

if (pixelCount < config->lineMinPixelCount)
{
    for (const auto& p : componentPixels)
    {
        labelImage.at<int>(p.y, p.x) = 0;
    }
}
else
{
    if (stepY > 0)
        std::reverse(componentPixels.begin(), componentPixels.end());

    lines.push_back(std::move(componentPixels));
    label++;
}
}

```

Figura 56: CustomConnectedComponents\_4

Pentru a elibera impuritățile, stergem liniile care au o componentă de puncte mai mică de `lineMinPixelCount`.

```
cv::approxPolyDP(smoothedLine, smoothedLine, fitPolyEpsilon, false);
```

Figura 57: fitPolynomial inner function

Pentru funcția approxPolyDP avem două wrapper-uri: fitPolynomial și rdpSimplify. La baza funcției approxPolyDP stă algoritmul dezvoltat de Ramer–Douglas–Peucker. Acesta este un algoritm utilizat pentru a simplifica o curbă alcătuită din segmente de dreaptă, care reduce numărul de puncte fără a modifica semnificativ forma generală a curbei [10]. În funcție de variabila epsilon algoritmul decide cât de tare simplifică forma liniei.

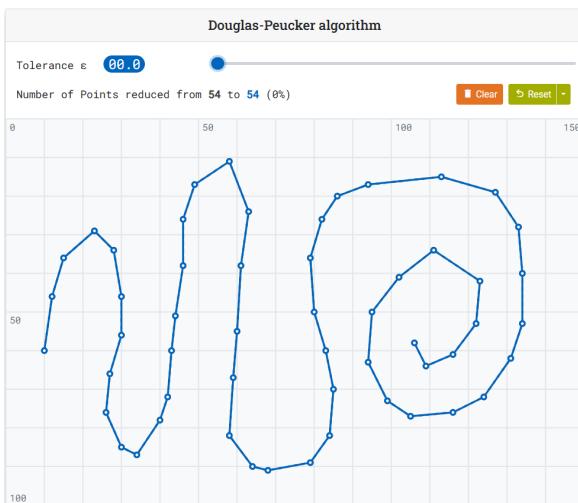


Figura 58: RDP Epsilon 0.0



Figura 59: RDP Epsilon 3.8

#### 4.3.1 Cazul Following Line

În cazul following line încercăm să aflăm dacă trecem în cazul următor sau rămanem în aceeași stare. Pentru a trece în cazul IN\_INTERSECTION trebuie să îndeplinim o condiție, să avem doar linii în partea de sus a imaginii. Această parte de sus este delimitată de o linie (figura 38 SRC linia albastră):

```
frameHeight * config->lineStartPointY
```

De-a lungul acestor calcule de început tratăm și enableFinishLineDetection. Cand este dezactivată pastrăm doar liniile mai mari decât "INTERSECTION\_minLineLength".

Dacă rămânem în același caz facem un schimb de perspectivă pentru a trece liniile în Birds Eye View

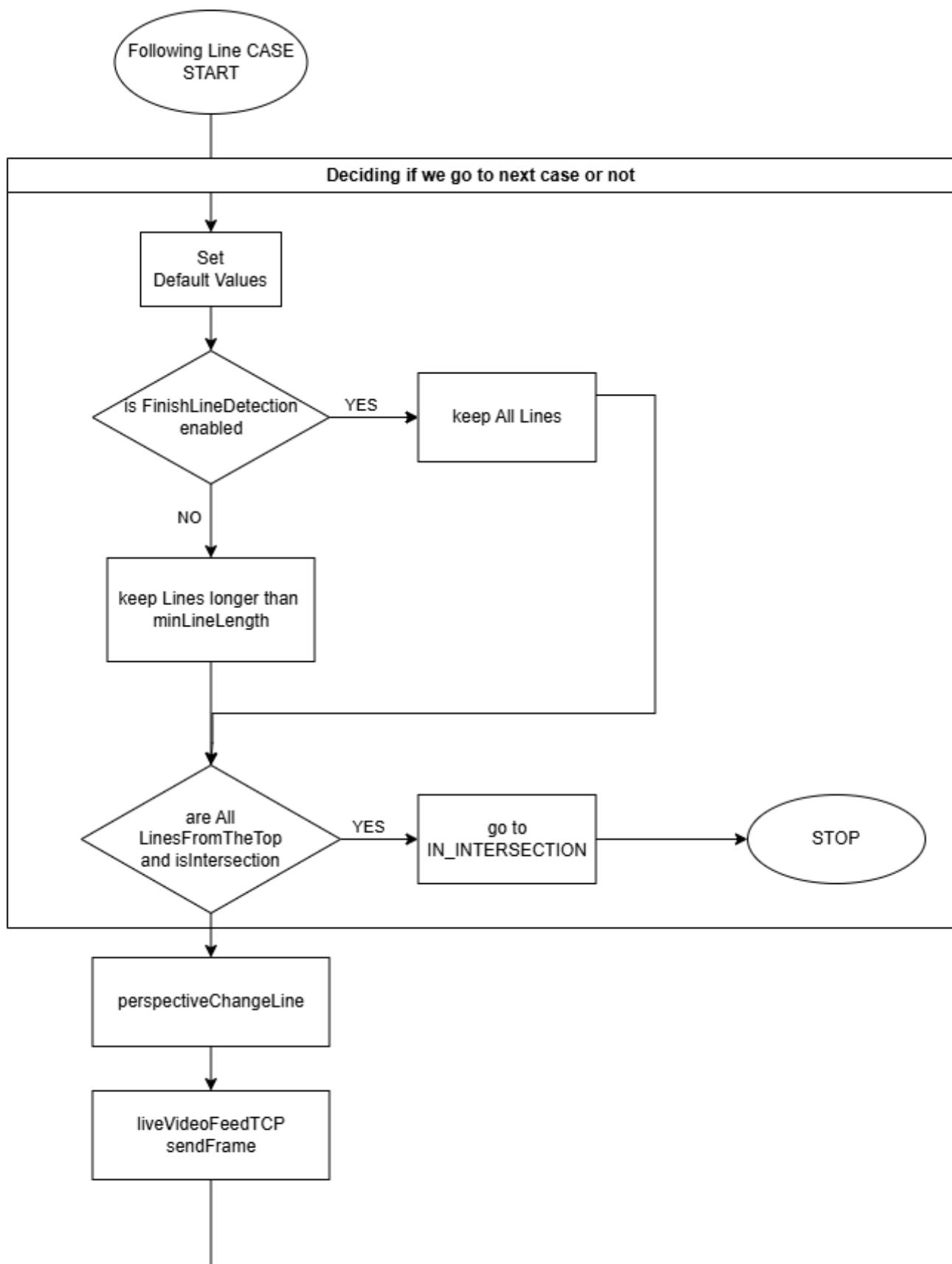


Figura 60: FOLLOWING LINE 1

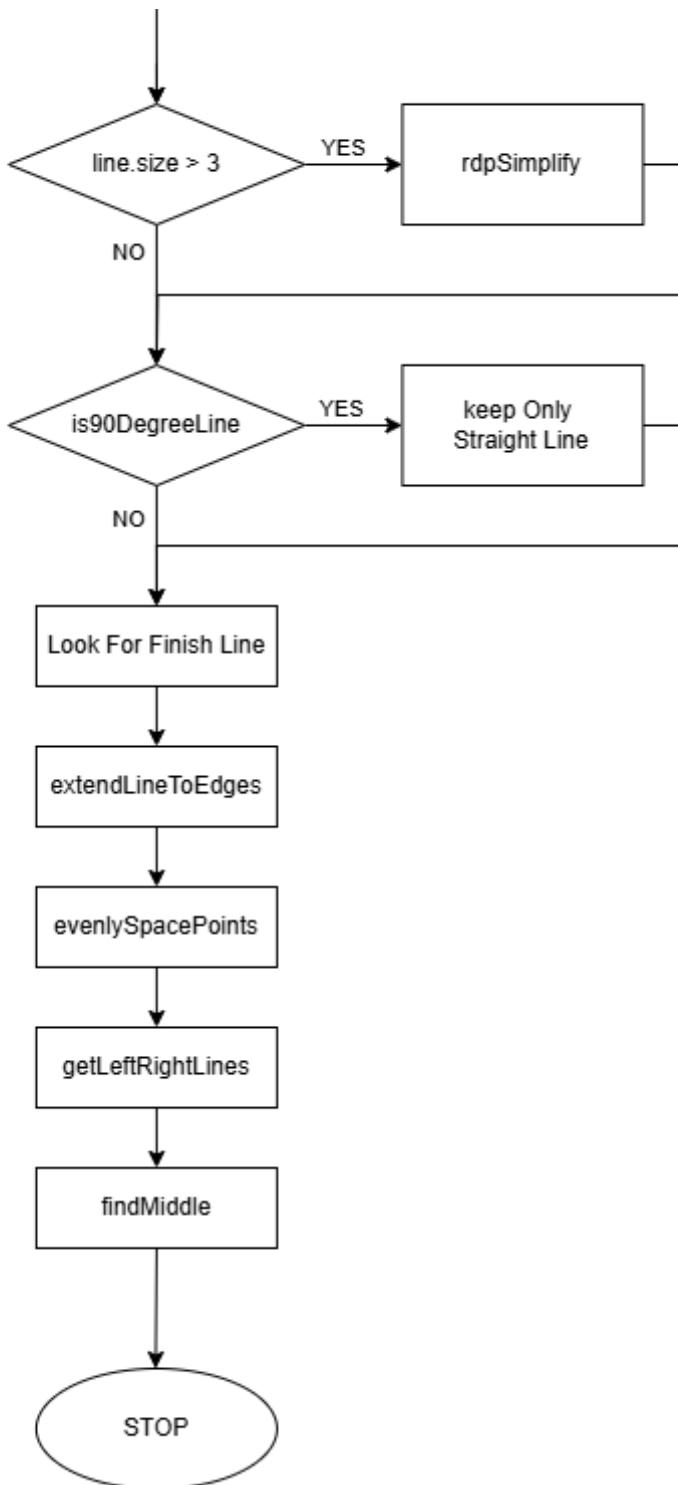


Figura 61: FOLLOWING LINE 2

În cazul intersecțiilor căutăm un unghi de 90 de grade. Din cauza aproximării uneori pentru un unghi perfect de 90 de grade format din trei puncte, putem primi 4 puncte. Situația asta se traduce automat în unghiuri mai mari de 90 de grade formate din cele 4 puncte. Deși vizibil ele

arată a colț de intersecție, algoritmul nu detectează. Pentru a ajusta erorile acestea folosim un algoritm de simplificare RdpSimplify discutat anterior.

Dacă găsim un unghi de 90 de grade printre punctele liniei păstrăm doar linia cea mai lungă. Acest stil de abordare a dat cel mai mare randament. În cazul în care nu găsim 90 de grade căutăm printre toate liniile structura de finish.

Mai apoi extindem liniile existente până în marginea ecranului pentru a putea pune 12 puncte egale.

#### 4.3.2 FINISH LINE DETECTION

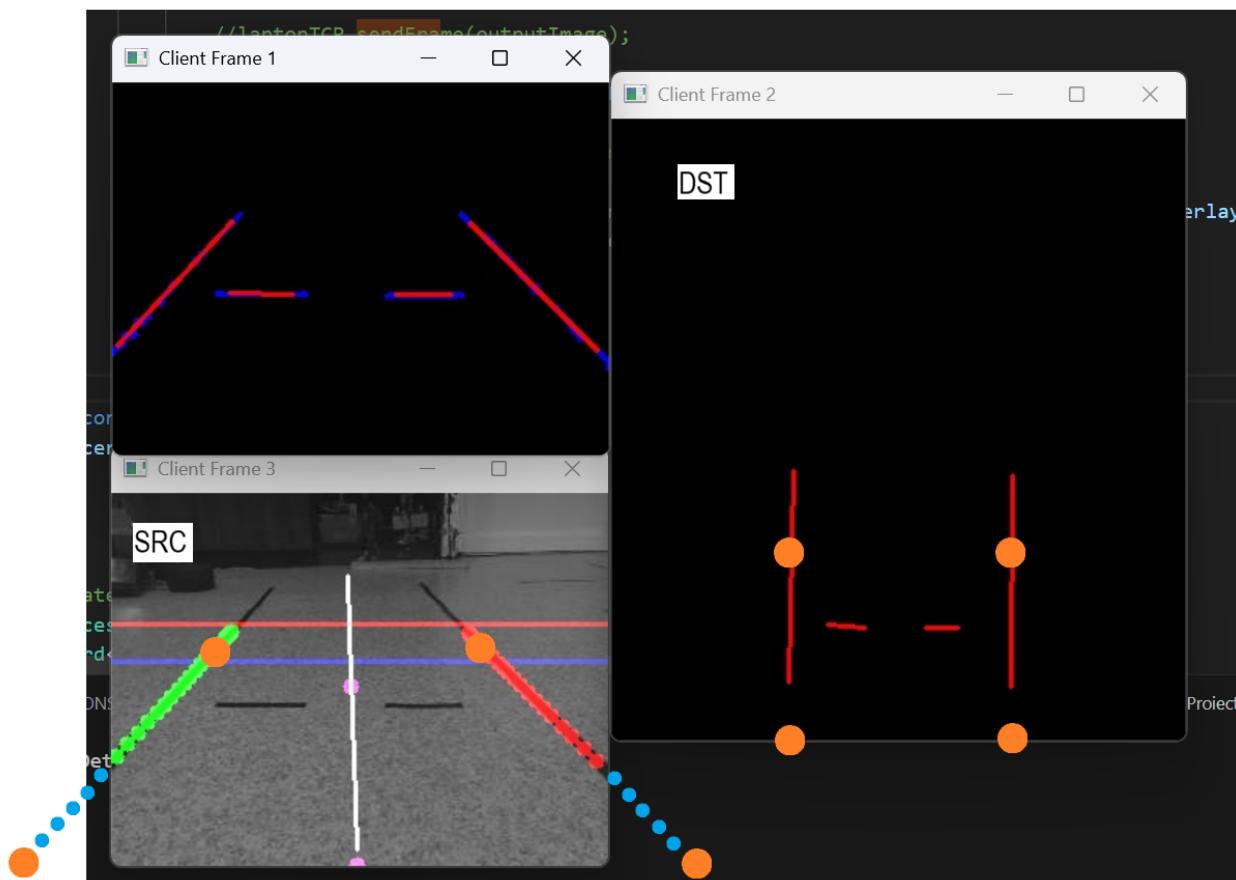


Figura 62: Finish Line View

Datorită câmpului vizual (FOV) al camerei, în zona liniei de finish sunt vizibile cel puțin, trei linii, în imagine chiar patru. Știm că o linie de finish este întotdeauna a treia în lista:

0. linia exterioară stânga
1. linia exterioară dreapta

2. linia finish stânga
3. linia finish dreapta

Se calculează punctul de mijloc al liniilor începând de la indexul 2(finish) și se identifică segmentul din liniile exterioare care se află cel mai aproape de acest punct. Calculam distanța dintre punct și segmentele exterioare pentru a ne asigura că linia de finish se află între cele două linii mari, ci nu în afara lor. Ulterior calculam unghiul dintre vectorii acestor segmente. Dacă se află într-un range de 75-105 grade (aproape 90) atunci am detectat linia de final.

#### 4.3.3 EXTEND LINE TO EDGES

Această funcție ia o linie/curbă formată din mai multe puncte. Calculează panta primelor și ultimelor 2 puncte din linie și le extinde până la margine.

De exemplu luăm ultimele două puncte de pe linie:

$$P_1 = (x_1, y_1), \quad P_2 = (x_2, y_2)$$

Panta dreptei este:

$$m = \frac{y_2 - y_1}{x_2 - x_1} \quad (10)$$

Pentru a determina valoarea lui  $x$  corespunzătoare unei anumite valori  $y$  de pe această dreaptă:

$$x = x_1 + \frac{y - y_1}{m} \quad (11)$$

$y$  poate avea ori valoarea 0 or frameHeight. Dacă înlocuim expresia pantei  $m$ , obținem:

$$x = x_1 + \frac{(y - y_1)(x_2 - x_1)}{y_2 - y_1} \quad (12)$$

#### 4.3.4 EVENLY SPACE POINTS

Calculăm dimensiunea totală a liniei/curbei din punctele pe care le avem de la aproximarea polinomială prin intermediul distanței euclidiene.

$$L_i = \sum_{k=1}^i \sqrt{(x_k - x_{k-1})^2 + (y_k - y_{k-1})^2} \quad (13)$$

Lungimea totală a curbei este:

$$L_{\text{total}} = L_{n-1} \quad (14)$$

Dacă dorim să plasăm  $N$  puncte distribuite uniform de-a lungul curbei, distanța constantă dintre ele este:

$$dist = \frac{L_{\text{total}}}{N - 1} \quad (15)$$

Pentru fiecare valoare:

$$s_i = i \cdot dist, \quad \text{unde } i = 1, 2, \dots, N - 2$$

căutăm indicele  $j$  astfel încât:

$$L_{j-1} \leq s_i < L_j$$

Apoi, interpolăm liniar între punctele  $P_{j-1}$  și  $P_j$ :

$$\begin{aligned} \text{raport} &= \frac{s_i - L_{j-1}}{L_j - L_{j-1}} \\ x_i &= x_{j-1} + \text{raport} \cdot (x_j - x_{j-1}) \\ y_i &= y_{j-1} + \text{raport} \cdot (y_j - y_{j-1}) \end{aligned} \quad (16)$$

Punctul  $(x_i, y_i)$  este adăugat la lista finală de puncte uniform distribuite.

#### 4.3.5 GET LEFT RIGHT & FIND MIDDLE

Această funcție clarifică care linie va fi considerată stânga, care dreapta. Funcția tratează trei cazuri: 2 sau mai multe linii, 1 linie, 0 linii.

**INFO:** *cv::Point2f firstPointLeftLine = undefinedPoint;*  
*cv::Point2f firstPointRightLine = undefinedPoint;*  
*unde "undefinedPoint" este colțul stanga jos al imaginii*

În primul caz se vor lua primele 2 linii și se va calcula distanța euclidiană între primele puncte istorice (firstPointLeftLine, firstPointRightLine) cu primul punct al liniei[0]

```
// Weighted distances
double weightedDistAtoLeft = euclideanDistance(firstPointLineA, firstPointLeftLine)
    + alpha * abs(firstPointLineA.y - firstPointLeftLine.y);
double weightedDistAtoRight = euclideanDistance(firstPointLineA, firstPointRightLine)
    + alpha * abs(firstPointLineA.y - firstPointRightLine.y);
double weightedDistBtoLeft = euclideanDistance(firstPointLineB, firstPointLeftLine)
    + alpha * abs(firstPointLineB.y - firstPointLeftLine.y);
double weightedDistBtoRight = euclideanDistance(firstPointLineB, firstPointRightLine)
    + alpha * abs(firstPointLineB.y - firstPointRightLine.y);
```

Figura 63: getLeftRightLines\_1

Ne uităm care linie este mai aproape de punctul istoric din stanga și mai departat de cel din dreapta, și viceversa.

```
// Correct classification
if (weightedDistAtoLeft < weightedDistAtoRight) {
    leftFitted = lines[0];
    rightFitted = lines[1];
    firstPointLeftLine = firstPointLineA;
    firstPointRightLine = firstPointLineB;
} else {
    leftFitted = lines[1];
    rightFitted = lines[0];
    firstPointLeftLine = firstPointLineB;
    firstPointRightLine = firstPointLineA;
}
```

Figura 64: getLeftRightLines\_2

În unele cazuri cand apar mai mult de 2 linii pot apărea și impostori și linia își dă flip. Adică linia din dreapta este considerată stânga iar linia care nu face parte din pistă este considerată dreapta. Pentru a corecta asta ne luam doar după o singură linie, cea care este cel mai aproape de adevăr, adică punctul istoric. Ulterior știind mărimea pistei facem mirroring.

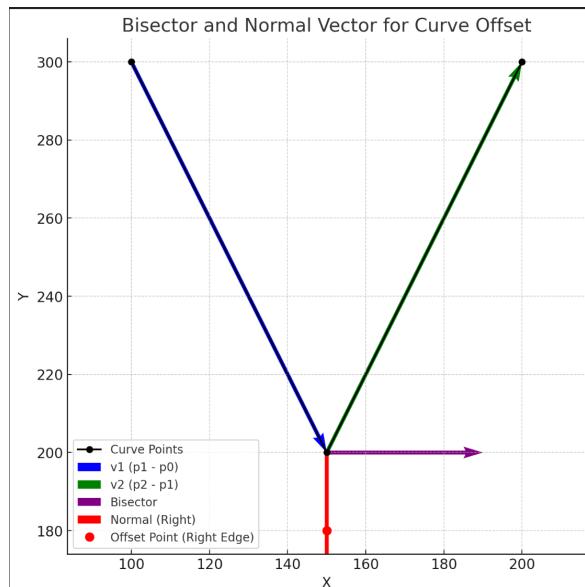


Figura 65: Mirroring Line 1

În cazul cu o singură linie ca și în situația cu impostori, facem mirroring în funcție de linia cea mai apropiată de adevăr, adică punctul istoric.

Partea de oglindire(mirroring) se face cu ajutorul bisectoarei. În direcția bisectoarei la o distanță setată de noi (în cazul asta

`trackLaneWidthInPixel +  
config->trackLaneWidthOffset)`

creăm un punct nou.

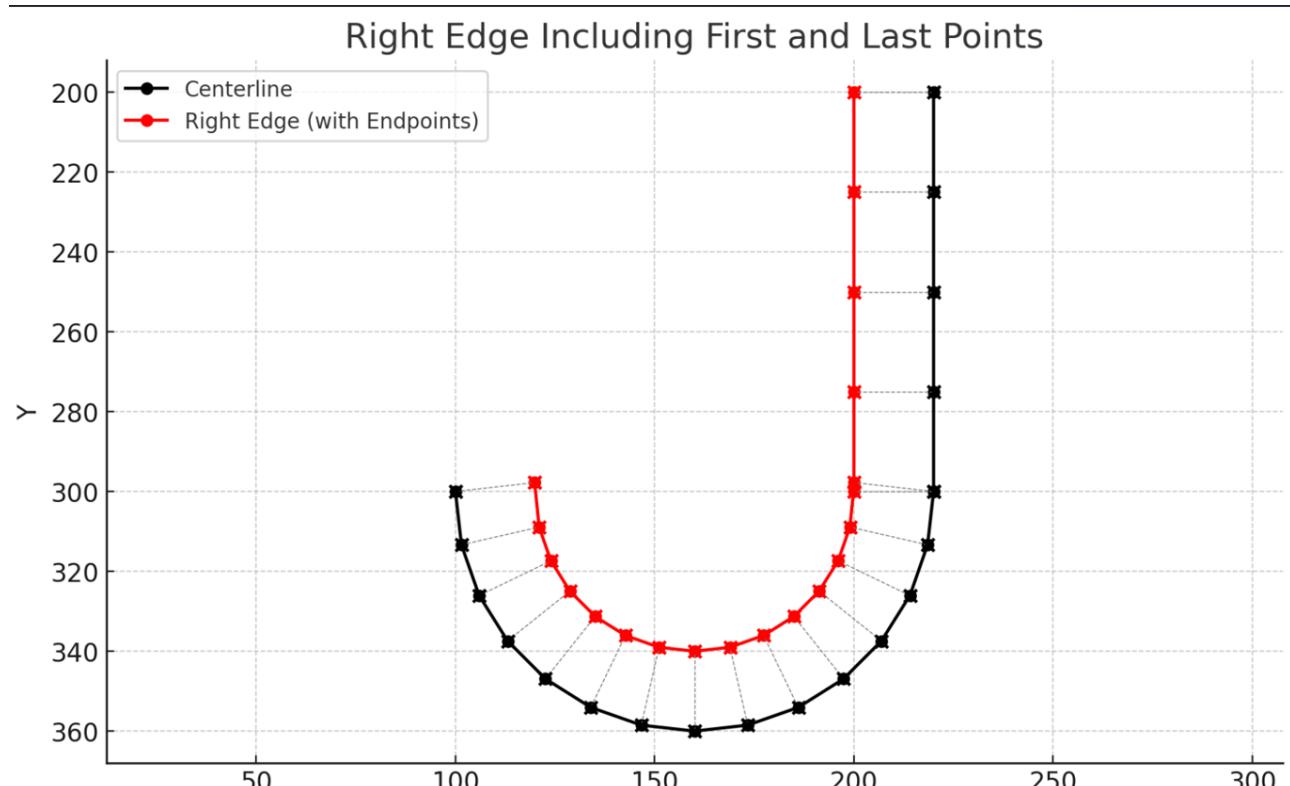


Figura 66: Mirroring Line 2

În cazul în care mașina nu vede nici o linie își pastrează direcția inițială.

#### 4.3.6 Cazul În Intersecție

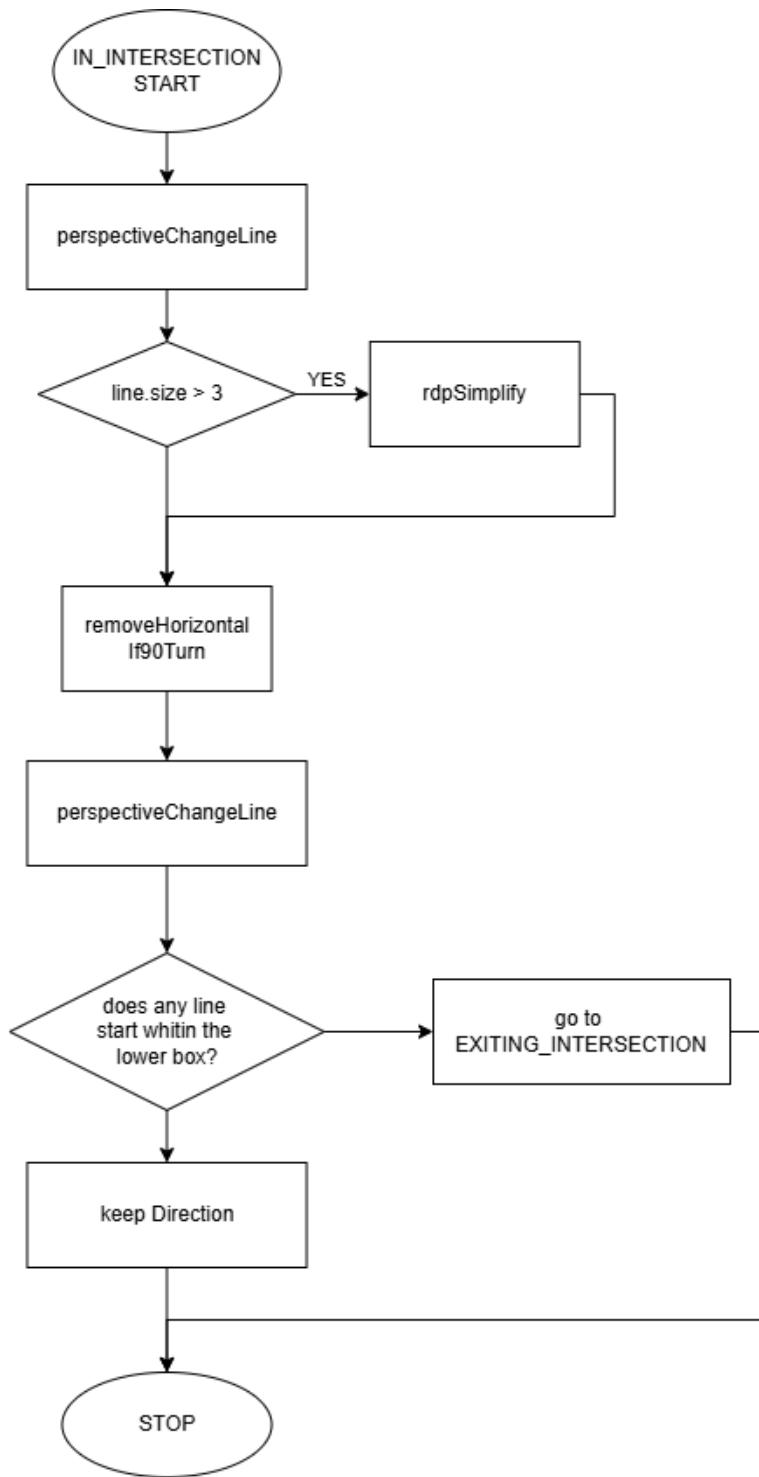


Figura 67: IN INTERSECTION

În cazul din mijlocul intersecției căutăm linii în cutia delimitată din partea de jos a imaginii a căror punct de start începe din acea zonă. Dacă găsim trecem în cazul următor și începem

procesarea. Daca nu găsim linii mașina își păstrează direcția initială. Uneori dacă mașina ieșe de pe pistă tinde să intre în acest caz și se învarte in jur.

#### 4.3.7 Cazul Exiting Intersection

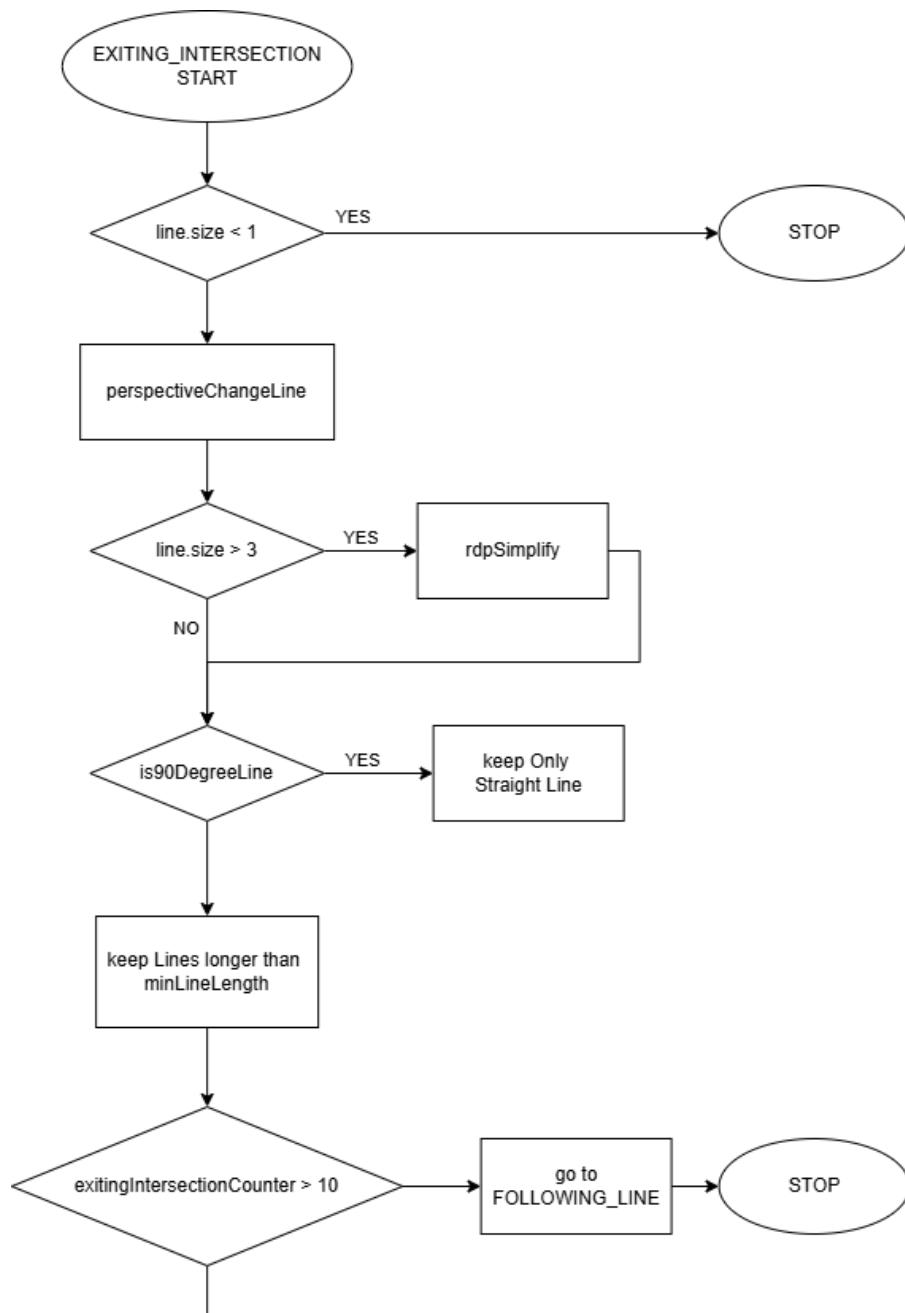


Figura 68: EXITING INTERSECTION 1

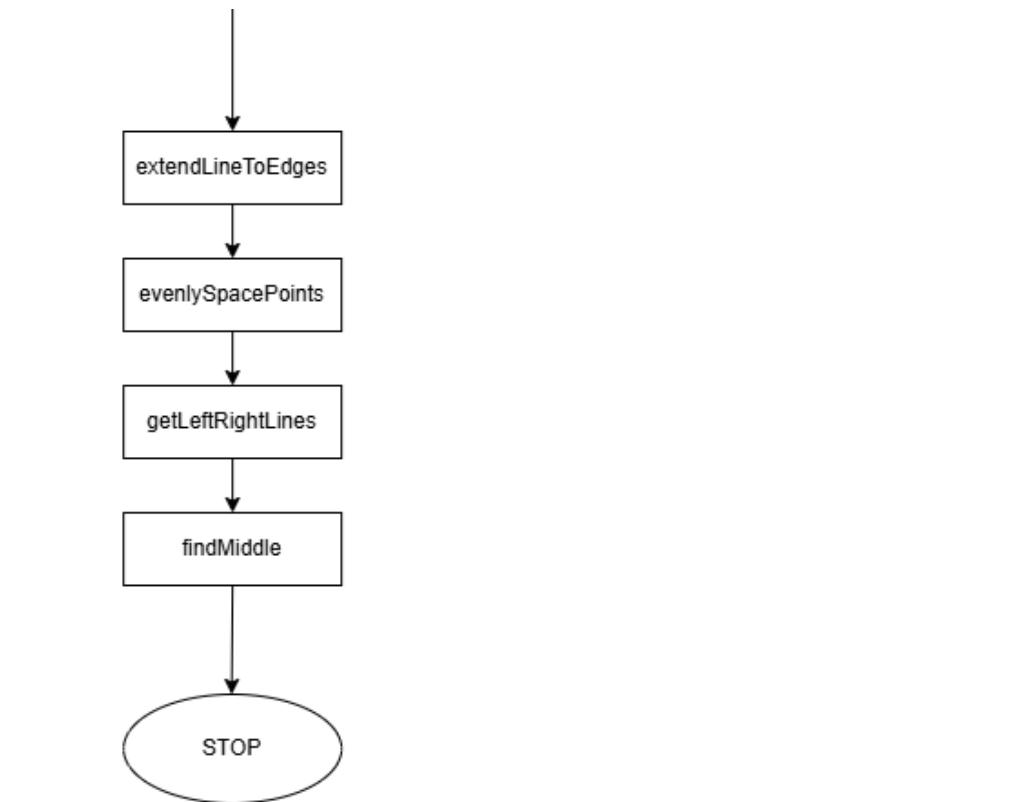


Figura 69: EXITING INTERSECTION 2

În cazul de ieșire din intersecție procesarea imaginii se va întâmpla de sus în jos pentru a preveni eroarea discutată în figura 47. Această eroare apare doar la colțurile intersecției. Astfel până vedem unghi de 90 de grade nu trecem în cazul de început "Following Line". Odată ce nu mai vedem acest colț, iterăm de minim 10 ori (este doar o acțiune de siguranță).

#### 4.4 Algoritmul Pure Pursuit și modelul de viteză

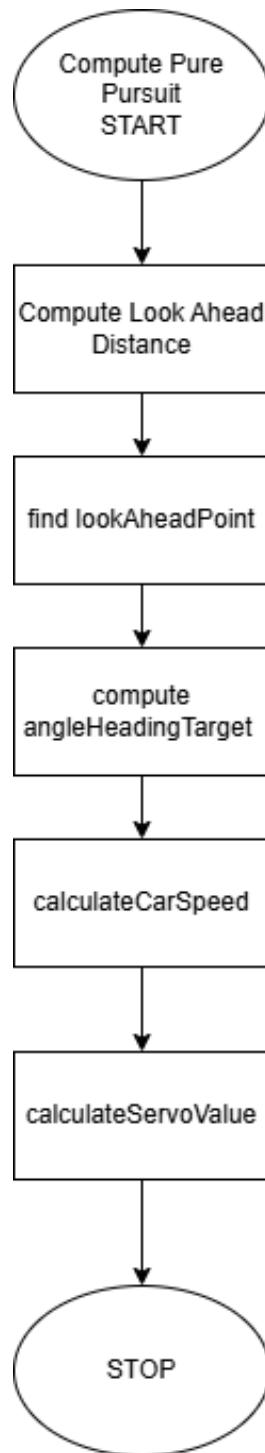


Figura 70: computePurePursuit

Algoritmul **Pure Pursuit** este utilizat pentru urmărirea unei traiectorii. Acesta presupune selectarea unui punct aflat la o anumită distanță în fața vehiculului, numit **punct de anticipare**

(lookahead), și calcularea unghiului de direcție necesar pentru a ajunge la acel punct, ca și cum vehiculul ar urma o traекторie circulară.

Totuși, în forma sa clasică, algoritmul are limitări în scenarii care implică curbe strânse, unde poate apărea fenomenul de **tăiere a curbelor**, ceea ce duce la erori de urmărire. Pe baza lucrării [16], a fost implementată o variantă îmbunătățită a acestui algoritm. Îmbunătățirea constă în **ajustarea dinamică** a distanței lookahead în funcție de viteza vehiculului și de raza curbei detectate. Astfel, în zonele cu viraje accentuate, punctul de anticipare este mai aproape de mașină, oferind o traекторie mai precisă și un comportament mai stabil.

#### 4.4.1 Calculare look Ahead Distance

LookAheadDistance este calculat cu ajutorul funcției computeK.

```
double PurePursuit::computeK(double angle)
{
    // std::cout << "Computed K R Parameter: " << R << "\n";
    if (angle <= config->minAngleLookAheadReference) return config->maxLookAheadInCm;
    if (angle >= config->maxAngleLookAheadReference) return config->minLookAheadInCm;

    // Linear interpolation between k_min and k_max
    double ratio = (config->maxAngleLookAheadReference - angle) / (config->maxAngleLookAheadReference - config->minAngleLookAheadReference);
    //std::cout << "angle: " << angle << " Ratio: " << ratio << "\n";
    double k = config->minLookAheadInCm + ratio * (config->maxLookAheadInCm - config->minLookAheadInCm);
    return k;
}
```

Figura 71: computeK

Următoarele constante sunt definite în prealabil:

- $\theta$  – unghiul de intrare
- $\theta_{\min}$  – minAngleLookAheadReference
- $\theta_{\max}$  – maxAngleLookAheadReference
- $k_{\min}$  – minLookAheadInCm
- $k_{\max}$  – maxLookAheadInCm

Funcția computeK se definește astfel:

$$K(\theta) = \begin{cases} k_{\max}, & \text{dacă } \theta \leq \theta_{\min} \\ k_{\min}, & \text{dacă } \theta \geq \theta_{\max} \\ k_{\min} + \left( \frac{\theta_{\max} - \theta}{\theta_{\max} - \theta_{\min}} \right) (k_{\max} - k_{\min}), & \text{altfel} \end{cases} \quad (17)$$

Dacă unghiul  $\theta$  este mai mic decât pragul minim, returnăm distanța maximă de lookahead ( $k_{\max}$ ). Dacă este mai mare decât pragul maxim, returnăm distanța minimă de lookahead ( $k_{\min}$ ).

Între cele două, aplicăm o interpolare liniară descrescătoare între  $k_{\max}$  și  $k_{\min}$  în funcție de unghiul  $\theta$  [5].

Ulterior unghiul este blocat în intervalul calculat de `shortestDistanceToCurve` și `longestDistanceOnCurveFromPoint`.

#### 4.4.2 shortestDistanceToCurve & longestDistanceOnCurveFromPoint

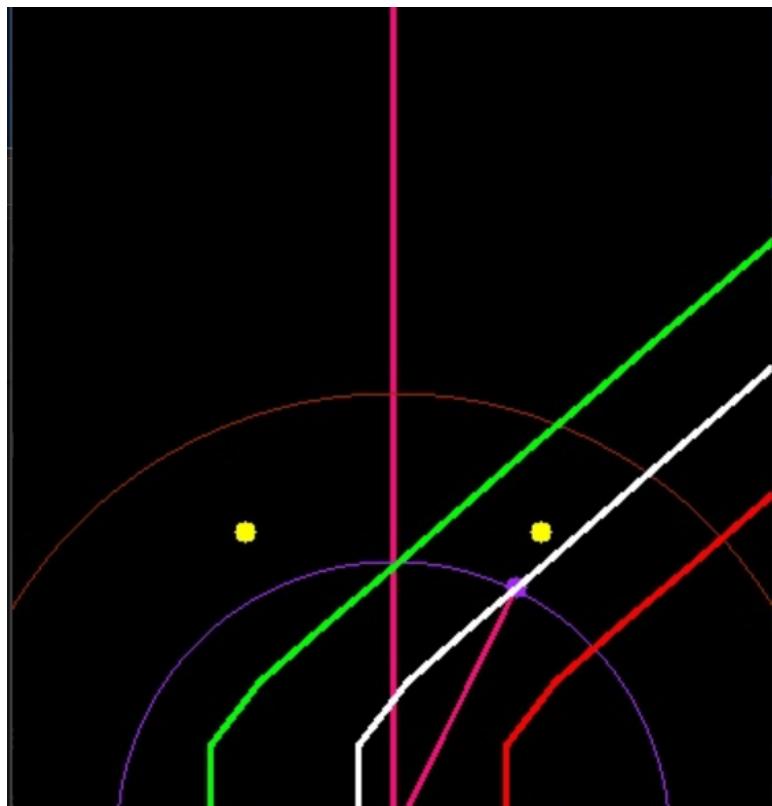


Figura 72: shortestDistanceToCurve

Funcția `shortestDistanceToCurve` determină cea mai mică distanță de la un punct  $P$  la o curbă formată din secvențe. Curbă este reprezentată ca o listă de puncte:

$$\{A_0, A_1, \dots, A_n\} \quad (18)$$

Pentru fiecare segment de dreaptă dintre două puncte consecutive  $A$  și  $B$ :

- Definim vectorul segmentului:

$$v = B - A \quad (19)$$

- Definim vectorul de la începutul segmentului până la punctul P:

$$\mathbf{w} = \mathbf{P} - \mathbf{A} \quad (20)$$

- Proiectăm w pe v [14]:

$$t = \frac{\mathbf{w} \cdot \mathbf{v}}{\mathbf{v} \cdot \mathbf{v}} \quad (21)$$

- Determinăm punctul cel mai apropiat de pe segment:

$$\mathbf{C} = \begin{cases} \mathbf{A}, & \text{dacă } t < 0 \\ \mathbf{A} + t\mathbf{v}, & \text{dacă } 0 \leq t \leq 1 \\ \mathbf{B}, & \text{dacă } t > 1 \end{cases} \quad (22)$$

- Calculăm distanța euclidiană:

$$d = \|\mathbf{P} - \mathbf{C}\| = \sqrt{(P_x - C_x)^2 + (P_y - C_y)^2} \quad (23)$$

Iar ulterior se reține minimul dintre toate aceste distanțe:

$$d_{\min} = \min(d_1, d_2, \dots, d_k)$$

Funcția `longestDistanceOnCurveFromPoint` în aceeași manieră determină cea mai mare distanță de la un punct P la curbă formată din secvențe. Se ia distanța euclidiană maximă[2].

#### 4.4.3 Căutarea look Ahead Point-ului

Funcția apelată se numește `findHighestIntersection`. Dorim să aflăm unde intersectează o curbă (formată din segmente de dreaptă) un cerc. Fiind o linie curbată, dar nu ca o sinusoidă, ne așteptăm să intersecteze linia în maxim 2 puncte. Alegem intersecția în direcția de deplasare a mașinii,(valoarea y cea mai mică, automat cea mai sus din imagine).

Ca și date de intrare avem:

- linia de mijloc
- $C(c_x, c_y)$ : centrul cercului  
(care este considerat începutul mașinii)
- $r$ : raza cercului

##### 1. Ecuația cercului

$$(x - c_x)^2 + (y - c_y)^2 = r^2 \quad (24)$$

Aceasta spune că toate punctele  $(x, y)$  aflate la distanță exactă  $r$  față de centru aparțin cercului.

## 2. Norma euclidiană (distanță)

$$\|\mathbf{P} - \mathbf{C}\| = \sqrt{(x - c_x)^2 + (y - c_y)^2} \quad (25)$$

Dacă un punct  $\mathbf{P}$  este pe cerc, atunci:

$$\|\mathbf{P} - \mathbf{C}\| = r \Leftrightarrow \|\mathbf{P} - \mathbf{C}\|^2 = r^2 \quad (26)$$

Automat ridicăm totul la pătrat ca să nu calculăm radical.

## 3. Parametrizarea segmentului

Fie un segment  $\overline{\mathbf{P}_1 \mathbf{P}_2}$ . Vectorul segmentului este:

$$\mathbf{v} = \mathbf{P}_2 - \mathbf{P}_1 = (dx, dy) \quad (27)$$

Orice punct de pe segment se scrie ca:

$$\mathbf{P}(t) = \mathbf{P}_1 + t \cdot \mathbf{v}, \quad t \in [0, 1] \quad (28)$$

## 4. Intersecția segmentului cu cercul

Substituim parametrizarea  $\mathbf{P}(t) = \mathbf{P}_1 + t \cdot \mathbf{v}$  în ecuația cercului:

$$\|\mathbf{P}(t) - \mathbf{C}\|^2 = r^2 \Rightarrow \|\mathbf{P}_1 + t \cdot \mathbf{v} - \mathbf{C}\|^2 = r^2 \quad (29)$$

Notăm cu  $\mathbf{d} = \mathbf{P}_1 - \mathbf{C}$ . Atunci:

$$\|\mathbf{d} + t \cdot \mathbf{v}\|^2 = r^2 \quad (30)$$

Dezvoltăm pătratul normei folosind produsul scalar:

$$(\mathbf{d} + t \cdot \mathbf{v}) \cdot (\mathbf{d} + t \cdot \mathbf{v}) = r^2 \quad (31)$$

$$t^2 \cdot \mathbf{v} \cdot \mathbf{v} + 2t \cdot \mathbf{d} \cdot \mathbf{v} + \mathbf{d} \cdot \mathbf{d} = r^2 \quad (32)$$

Care este o ecuație de gradul 2 în  $t$ , adică:

$$at^2 + bt + c = 0 \quad (33)$$

unde coeficienții sunt:

$$a = \mathbf{v} \cdot \mathbf{v} = dx^2 + dy^2 \quad (34)$$

$$b = 2 \cdot \mathbf{d} \cdot \mathbf{v} = 2(dx(x_1 - c_x) + dy(y_1 - c_y)) \quad (35)$$

$$c = \mathbf{d} \cdot \mathbf{d} - r^2 = (x_1 - c_x)^2 + (y_1 - c_y)^2 - r^2 \quad (36)$$

## 5. Soluțiile ecuației

Calculăm discriminantul:

$$\Delta = b^2 - 4ac \quad (37)$$

- Dacă  $\Delta < 0$ : nu există intersecții reale.
- Dacă  $\Delta \geq 0$ : calculăm soluțiile:

$$t = \frac{-b \pm \sqrt{\Delta}}{2a} \quad (38)$$

Dacă  $t \in [0, 1]$ , intersecția este pe segment.

## 6. Alegerea punctului în direcția de deplasare(cel mai de sus)

Păstrăm toate intersecțiile valide. Alegem punctul cu  $y$  minim.

## 7. Concluzie

Funcția:

- Parurge fiecare segment din curbă
- Verifică dacă intersectează cercul
- Calculează soluțiile reale de intersecție (dacă există)
- Reține doar punctele de pe segment
- Returnează punctul cel mai de "sus" (cu  $y$  minim)

### 4.4.4 Calculare angle Heading Target

Această funcție ne calculează unghiul la care se află punctul de interes pentru noi. Se iau trei puncte: punctul de mijloc jos al imaginii unde se află mașina, punctul de mijloc sus și punctul de pe curbă (figura 69).

### Funcția pentru calculul unghiului semnat dintre trei puncte:

Fie punctele  $P_1, P_2, P_3 \in \mathbb{R}^2$  astfel:

$$\vec{v}_1 = P_1 - P_2 \quad (39)$$

$$\vec{v}_2 = P_3 - P_2 \quad (40)$$

$$\text{Produsul scalar:dot} = \vec{v}_1 \cdot \vec{v}_2 = v_{1x} \cdot v_{2x} + v_{1y} \cdot v_{2y} \quad (41)$$

$$\text{Produsul vectorial:cross} = \vec{v}_1 \times \vec{v}_2 = v_{1x} \cdot v_{2y} - v_{1y} \cdot v_{2x} \quad (42)$$

$$\theta = \text{atan2(cross, dot)} \quad (43)$$

**Rezultat:**  $\theta$  este unghiul cu semn (în radiani) dintre vectorii  $\vec{v}_1$  și  $\vec{v}_2$ , pozitiv dacă rotația de la  $\vec{v}_1$  la  $\vec{v}_2$  este în sens anterior și negativ dacă este în sens orar.

#### 4.4.5 Calculul vitezei mașinii în funcție de curbă

Pentru a determina viteza corectă într-o curbă, folosim următorii pași:

##### 1. Calculul razei de virare:

$$R = \left| \frac{L}{\tan(\theta)} \right|$$

unde:

- $R$  este raza de virare,
- $L$  este distanța dintre puntea față și puntea spate (empattement),
- $\theta$  este unghiul de direcție.

##### 2. Calculul vitezei maxime în curbă:

$$v_{\max, \text{curbă}} = \sqrt{|\mu \cdot R \cdot a|}$$

unde:

- $\mu$  este coeficientul de frecare cu solul,
- $R$  este raza curbei,
- $a$  este accelerația efectivă în jos (gravitațională sau cu forță de apăsare).

##### 3. Boost la ieșirea din curbă:

Dacă unghiul de virare este mai mic decât un prag  $\theta_{\text{limit}}$ , se adaugă un boost de viteză:

$$\text{dacă } |\theta| < \theta_{\text{limit}} \Rightarrow v_{\text{boost}} = v + \Delta v$$

#### 4. Limitarea vitezei între minim și maxim:

$$v_{\text{final}} = \min(v_{\text{max}}, \max(v_{\text{min}}, v))$$

##### 4.4.6 Calculul valorii servomotorului pentru direcție

Pentru a transforma unghiul dorit de direcție într-o valoare de control pentru servomotor, se aplică următorii pași:

###### 1. Calculul unghiului de direcție real:

$$\theta_s = \arctan \left( \frac{2 \cdot L \cdot \sin(\theta)}{d_{\text{la}}} \right)$$

unde:

- $\theta$  este unghiul dintre direcția curentă și punctul de urmărire (lookahead point),
- $L$  este lungimea ampatamentului (în cm),
- $d_{\text{la}}$  este distanța lookahead.

###### 2. Conversia unghiului în grade:

$$\theta_s^\circ = \theta_s \cdot \frac{180}{\pi}$$

###### 3. Calculul valorii servomotorului:

$$\text{servo} = \left( \frac{\theta_s^\circ}{\theta_{\text{max}}} \right) \cdot \theta_{\text{servo,max}} \cdot k_{\text{ajustare}}$$

unde:

- $\theta_{\text{max}}$ : unghiul maxim mecanic de direcție (în °),
- $\theta_{\text{servo,max}}$ : amplitudinea maximă a semnalului pentru servomotor,
- $k_{\text{ajustare}}$ : coeficient de ajustare (calibrat).

###### 4. Limitarea valorii:

$$\text{servo}_{\text{final}} = \min(\theta_{\text{dreapta,max}}, \max(\theta_{\text{stanga,max}}, \text{servo}))$$

##### 4.4.7 Implementarea și eliminarea PID

Pentru calculul vitezei am avut implementat în sistem un senzor infraroșu care funcționa ca și tuometru. Din măsurători reiese faptul că la 16 impulsuri citite de senzor avem 10cm deplasare

fizică. Din cauza spațiului redus, în momentul adăugării al celui de al doilea motor am renunțat la senzor. Am trecut de la un regulator PID la o regulare pe baza unei funcții pătratice care funcționează și fără turometru.

Tabela 2: Reglaj PID folosind metoda Ziegler–Nichols (reglaj oscilatoriu)

Tip regulator	$K_p$	$T_i$	$T_d$
P	$0.5 \cdot K_u$	—	—
PI	$0.45 \cdot K_u$	$T_u/1.2$	—
PID	$0.6 \cdot K_u$	$T_u/2$	$T_u/8$

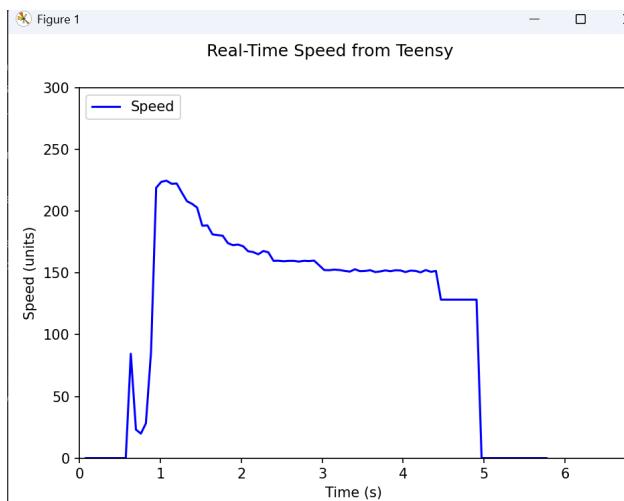


Figura 73: Calibrare inițială

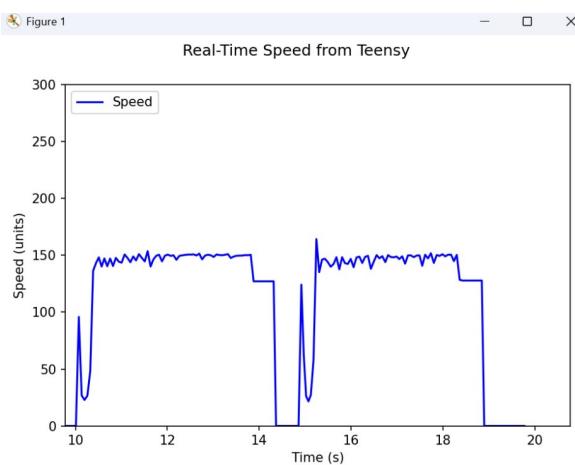


Figura 74: Calibrare finală

## Transformarea unei viteze într-o valoare de servo folosind o funcție pătratică inversabilă

Dorim să știm ce valoare numerică trebuie transmisă unui servo motor pentru a obține o anumită viteză  $v$  (în cm/s). Relația dintre valoarea  $t$  transmisă și viteză rezultată este aproximată printr-o funcție de gradul al doilea.

### 1. Definirea relației între viteză și semnalul servo

Presupunem că există o relație de forma:

$$v = at^2 + bt + c \quad (44)$$

unde:

- $v$  este viteză dorită în cm/s,

- $t$  este valoarea transmisă către servo motor (în intervalul [90, 120]),
- $a, b, c$  sunt coeficienți obținuți prin ajustare (fit) pe baza unor măsurători experimentale.

## 2. Reformularea problemei ca ecuație de gradul al doilea în $t$

Dorim să determinăm  $t$  pentru o viteza dată  $v$ . Astfel, rescriem ecuația:

$$at^2 + bt + (c - v) = 0 \quad (45)$$

## 3. Calculul discriminantului

Discriminantul este:

$$\Delta = b^2 - 4a(c - v) \quad (46)$$

## 4. Rezolvarea ecuației de gradul al doilea

Dacă  $\Delta \geq 0$ , atunci avem soluții reale:

$$t = \frac{-b \pm \sqrt{\Delta}}{2a} \quad (47)$$

Alegem soluția cu  $+\sqrt{\Delta}$ , deoarece valoarea  $t$  crește cu viteza:

$$t = \frac{-b + \sqrt{b^2 - 4a(c - v)}}{2a} \quad (48)$$

## 5. Exemplu de coeficienți reali:

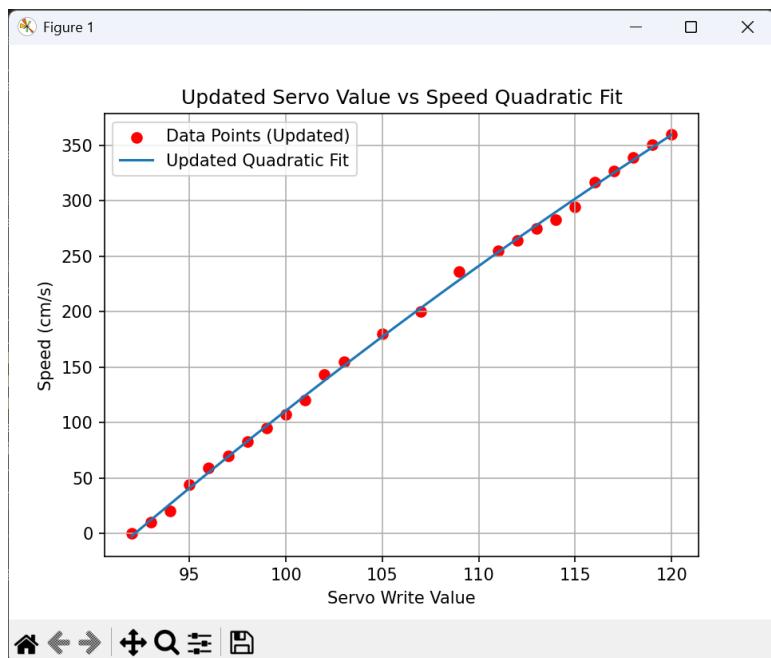


Figura 75: Măsurători Viteză

Într-un caz real obținut din măsurători, coeficienții au fost:

$$a = -0.0020, \quad b = 13.9775, \quad c = -1263.7335 \quad (49)$$

## 6. Returnarea valorii de servo

Valoarea obținută  $t$  este transmisă direct funcției de scriere servo:

$$\text{servo.write}(t) \quad (50)$$

## 4.5 Funcționalitatea de oprire de urgență

```

if(this->isObjectCloserThanDistanceBeforeIssuesAppear)
{
    #if 1 == ENABLE_TCP_FRAMES
        liveVideoFeedTCP.sendFrame(frame);
    #endif
    if(distance <= stoppingDistanceBoxLidar){
        ppObject.setSpeed(0);
        isRadarEnabled = false;
    }else{
        ppObject.setSpeed(config->minSpeedAfterFinish);
    }

    serialString = this->createSerialString(isBlueStatusLedOn,
                                              isYellowStatusLedOn, isRadarEnabled);
    serial.writeToSerial(serialString);
}
else
{
    // * cv::TickMeter timer;
    // * timer.start();
    cv::Mat thresholdFrame = this->applyThreshold(frame);
}

```

Figura 76: Cod de Oprită

După finish line mașina reduce viteza brusc și pornește mișcarea radarului. Dacă detectează un obstacol la o distanță de 60 de cm oprește procesarea imaginii pentru a nu introduce erori, deci automat ține ultima direcție calculată. Când se apropie de cub mașina pună frână la o distanță de 5cm.

## 4.6 Scenarii de rulare

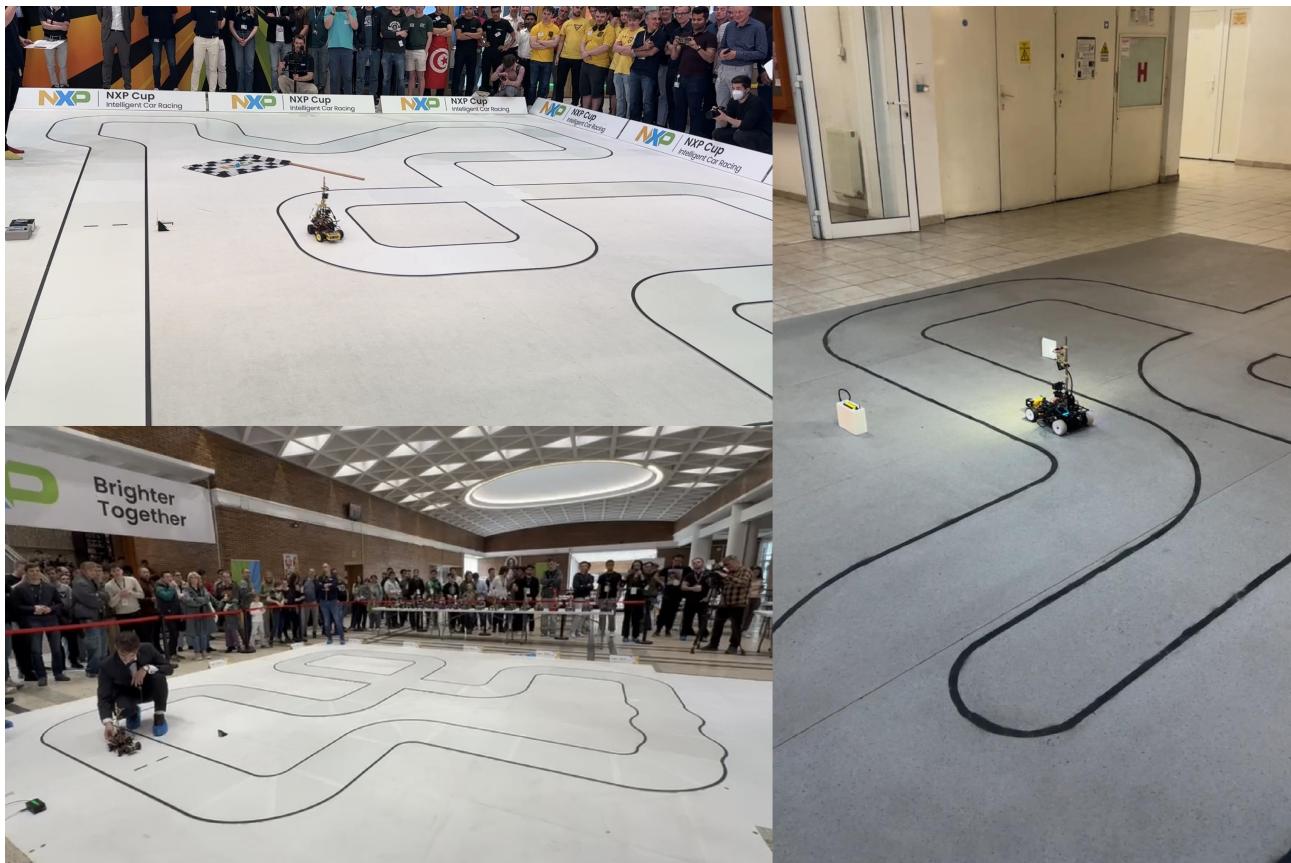


Figura 77: Piste rulate

Pe partea din stânga avem pista oficială NXP Cup, pe dreapta avem pista creată la facultate. Cele două diferă. Performanțele sunt puțin diferite.

## 4.7 Probleme întâmpinate, soluții și posibile îmbunătățiri

La început am avut probleme de operare a pinilor GPIO de pe placuța Debix. Linux oferă acces la anumite fisiere în care se scrie 1 sau 0 pentru a activa pinii. Din cauza aceasta am adăugat placuța Teensy. Într-un final am aflat că pot utiliza sistemul de fisiere de pe linux.

La fiecare repornire a programului camera și usb portul de pe Debix primeau alt id. A fost automatizat cu un script shell.

ESC(Electronic Speed Controller)-ul nu funcționa corect. La anumite prescrieri mergea în sens opus. A trebuit calibrat, odată prin firmware iar odata prin cod prin intermediul delayuri.

În loc de double am folosit int în multe locuri iar calculele erau eronate toate.

Împărțiri la 0 sau Segmentation fault.

FPS scăzut, a trebuit să reduc dimensiunea imaginilor.

Colțul intersecțiilor a fost o mare problemă (figura 50) soluția este tratată în document.

Librăriile folosite pentru lcd nu funcționau deloc pe C++ astfel am trecut la librarii de Python.

Fenomenul Flickering a fost o mare problemă la concurs. Solutia finală a fost schimbarea timpului de expunere la lumină.

Possible imbunătățiri:

- Folosirea unei singure plăci. Ori va trebui folosirea ADC-ului de pe Debix, ori va trebui implementată o procesare de imagine pe Teensy. A doua variantă ar reduce foarte mult din greutatea mașinii, automat și dimensiunile dar este și de o complexitate majoră.
- Când am detectat liniile am incercat o variantă cu funcții de gradul 12 care aproximau foarte bine curba și puteau prezice ce urmează însă nu am continuat dezvoltarea.
- Cele mai multe îmbunătățiri sunt hardware:
  - sasiu mai mic
  - roți mai mici
  - motoare mai tari
  - step down mai mic
  - baterie mai mică
- Diferențial software. Diferențialul mecanic actual de pe mașină are limite. Ar fi trebuit să am ori 4x4, ori diferențial software. Uneori mașina rămâne suspendată dacă o roată nu atinge solul. Diferențialul software are și posibilitatea de a forța curba pentru a merge mai rapid.
- aderența roților

## 4.8 Concluzii generale

Acest proiect a fost dezvoltat de-a lungul a unui întreg an. Mașina a avut rezultate satisfăcătoare și mai are loc destul pentru îmbunătățiri. Proiectul înglobează folosirea a mai multor componente hardware și limbaje de programare. El rezolvă un caz izolat, concursul NXP Cup, line followingul într-un mediu controlat. Tehnica Pure Pursuit dovedește robustețea algoritmului, iar detectia liniilor stabilită în această lucrare pune bazele pentru oricine dorește să pornească la drum în această competiție.

## Bibliografie

- [1] *Ackermann Steering Geometry*, în Wikipedia, 5 Feb. 2025, URL: [https://en.wikipedia.org/w/index.php?title=Ackermann\\_steering\\_geometry&oldid=1274130697](https://en.wikipedia.org/w/index.php?title=Ackermann_steering_geometry&oldid=1274130697) (accesat în 31.5.2025).
- [2] *Euclidean Distance*, în Wikipedia, 30 Apr. 2025, URL: [https://en.wikipedia.org/w/index.php?title=Euclidean\\_distance&oldid=1288127620](https://en.wikipedia.org/w/index.php?title=Euclidean_distance&oldid=1288127620) (accesat în 23.6.2025).
- [3] *Frecvență*, în Wikipedia, 9 Iul. 2024, URL: <https://ro.wikipedia.org/w/index.php?title=Frecven%C8%9B%C4%83&oldid=16380927> (accesat în 30.5.2025).
- [4] Xiaohui Li et al., *An Adaptive Preview Path Tracker for Off-Road Autonomous Driving*, 1 Iun. 2013, p. 1723, 1718 pp., ISBN: 978-1-4673-4707-5, DOI: 10.1109/ICCA.2013.6564867.
- [5] *Linear Interpolation*, în Wikipedia, 19 Apr. 2025, URL: [https://en.wikipedia.org/w/index.php?title=Linear\\_interpolation&oldid=1286306558](https://en.wikipedia.org/w/index.php?title=Linear_interpolation&oldid=1286306558) (accesat în 23.6.2025).
- [6] *Lock (Computer Science)*, în Wikipedia, 30 Apr. 2025, URL: [https://en.wikipedia.org/w/index.php?title=Lock\\_\(computer\\_science\)&oldid=1288077056](https://en.wikipedia.org/w/index.php?title=Lock_(computer_science)&oldid=1288077056) (accesat în 1.6.2025).
- [7] *OpenCV: Geometric Image Transformations*, URL: [https://docs.opencv.org/4.10.0/da/d54/group\\_\\_imgproc\\_\\_transform.html#gae66ba39ba2e47dd0750555c7e986ab85](https://docs.opencv.org/4.10.0/da/d54/group__imgproc__transform.html#gae66ba39ba2e47dd0750555c7e986ab85) (accesat în 8.6.2025).
- [8] *OpenCV: Image Thresholding*, URL: [https://docs.opencv.org/4.10.0/d7/d4d/tutorial\\_py\\_thresholding.html#autotoc\\_md1459](https://docs.opencv.org/4.10.0/d7/d4d/tutorial_py_thresholding.html#autotoc_md1459) (accesat în 4.6.2025).
- [9] *OpenCV: OpenCV Modules*, URL: <https://docs.opencv.org/4.10.0/> (accesat în 3.6.2025).
- [10] *Ramer–Douglas–Peucker Algorithm*, în Wikipedia, 8 Iun. 2025, URL: [https://en.wikipedia.org/w/index.php?title=Ramer%E2%80%93Douglas%E2%80%93Peucker\\_algorithm&oldid=1294599147](https://en.wikipedia.org/w/index.php?title=Ramer%E2%80%93Douglas%E2%80%93Peucker_algorithm&oldid=1294599147) (accesat în 18.6.2025).
- [11] *Regular Expression*, în Wikipedia, 26 Mai 2025, URL: [https://en.wikipedia.org/w/index.php?title=Regular\\_expression&oldid=1292329025](https://en.wikipedia.org/w/index.php?title=Regular_expression&oldid=1292329025) (accesat în 2.6.2025).
- [12] NXP Semiconductors, *I.MX RT Crossover MCUs*, URL: <https://www.nxp.com/products/processors-and-microcontrollers/arm-microcontrollers/i-mx-rt-crossover-mcus:IMX-RT-SERIES> (accesat în 29.5.2025).
- [13] NXP Semiconductors, *NXP-CUP-2025-RULES*, URL: <https://www.nxp.com/nxpcup> (accesat în 27.5.2025).

- 
- [14] *Vector Projection*, în *Wikipedia*, 8 Mai 2025, URL: [https://en.wikipedia.org/w/index.php?title=Vector\\_projection&oldid=1289415674](https://en.wikipedia.org/w/index.php?title=Vector_projection&oldid=1289415674) (accesat în 23.6.2025).
  - [15] *Video4Linux*, în *Wikipedia*, 1 Feb. 2025, URL: <https://en.wikipedia.org/w/index.php?title=Video4Linux&oldid=1273253875> (accesat în 30.5.2025).
  - [16] Meng Wang et al., „Improved Pure Pursuit Algorithm Based Path Tracking Method for Autonomous Vehicle”, în *Journal of Advanced Computational Intelligence and Intelligent Informatics* 28.4 (20 Iul. 2024), pp. 1034–1042, DOI: 10.20965/jaciii.2024.p1034, URL: <https://www.fujipress.jp/jaciii/jc/jacii002800041034/> (accesat în 24.6.2025).

**DECLARAȚIE DE AUTENTICITATE A  
LUCRĂRII DE FINALIZARE A STUDIILOR\***

Subsemnatul GULYASY ALEXANDRU - CĂTĂLIN

legitimat cu C1 seria T7 nr. 685918,

CNP 5020813350033

autorul lucrării VEHICUL AUTONOM NXP CUP 2025

elaborată în vederea susținerii examenului de finalizare a studiilor de  
LICENȚĂ AUTOMATICĂ și CALCULATORARE organizat de către Facultatea  
Politehnica Timișoara, sesiunea IUNIE a anului universitar  
2025, coordonator SL.DR.ING. PAUL NEGÎRLĂ, luând în  
considerare art. 34 din *Regulamentul privind organizarea și desfășurarea examenelor de  
licență/diplomă și disertație*, aprobat prin HS nr. 109/14.05.2020 și cunoscând faptul că în  
cazul constatării ulterioare a unor declarații false, voi suporta sancțiunea administrativă  
prevăzută de art. 146 din Legea nr. 1/2011 – legea educației naționale și anume anularea  
diplomei de studii, declar pe proprie răspundere, că:

- această lucrare este rezultatul propriei activități intelectuale;
- lucrarea nu conține texte, date sau elemente de grafică din alte lucrări sau din alte surse fără ca acestea să nu fie citate, inclusiv situația în care sursa o reprezintă o altă lucrare/alte lucrări ale subsemnatului;
- sursele bibliografice au fost folosite cu respectarea legislației române și a convențiilor internaționale privind drepturile de autor;
- această lucrare nu a mai fost prezentată în fața unei alte comisii de examen/prezentată public/publicată de licență/diplomă/disertație;
- În elaborarea lucrării am utilizat instrumente specifice inteligenței artificiale (IA) și anume \_\_\_\_\_ (denumirea) \_\_\_\_\_ (sursa), pe care le-am citat în conținutul lucrării/nu am utilizat instrumente specifice inteligenței artificiale (IA)<sup>1</sup>.

Declar că sunt de acord ca lucrarea să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțînd inclusiv la introducerea conținutului său într-o bază de date în acest scop.

Timișoara,

Data

25.06.2025

Semnătura

Gulyasy

\*Declarația se completează de student, se semnează olograf de acesta și se inserează în lucrarea de finalizare a studiilor, la sfârșitul lucrării, ca parte integrantă.

<sup>1</sup> Se va păstra una dintre variante: 1 - s-a utilizat IA și se menționează sursa 2 – nu s-a utilizat IA