



Universitatea Politehnica Bucureşti  
Facultatea de Automatică și Calculatoare  
Departamentul de Automatică și Ingineria Sistemelor

# LUCRARE DE DIPLOMĂ

## Sistem în timp real pentru controlul iluminatului public

Absolvent  
Durbală Cristian

Coordonator  
Prof. dr. ing. Monica Drăgoicea

Bucureşti, 2023

# Cuprins

<b>Glosar</b>	<b>iv</b>
<b>Listă de figuri</b>	<b>v</b>
<b>1. Introducere</b>	<b>1</b>
1.1. Formularea problemei generale . . . . .	1
<b>2. Elemente ale arhitecturii hardware</b>	<b>3</b>
2.1. Diagrama de context . . . . .	3
2.2. Prezentare elemente diagramă context . . . . .	4
2.2.1. Modul senzor poziție . . . . .	4
2.2.2. Fotorezistor . . . . .	5
2.2.3. LED-uri . . . . .	5
2.2.4. Arduino Mega . . . . .	5
2.3. Simulare a sistemului în Tinkercad și realizarea fizică . . . . .	6
<b>3. Elemente ale arhitecturii software</b>	<b>8</b>
3.1. Achiziție date și constrângeri de timp . . . . .	8
3.2. Descriere FreeRTOS . . . . .	9
3.3. Prezentare task-uri . . . . .	9
3.3.1. TaskLightRead . . . . .	9
3.3.2. TaskProd . . . . .	9
3.3.3. TaskCons . . . . .	10
3.4. Analiză a codului . . . . .	11
3.4.1. Prelucrare date citite și counter obiecte . . . . .	11
3.4.2. Transmitere date între task-uri . . . . .	13
3.4.3. Transformare date senzori mișcare în valori de timp și în viteză . . . . .	13
3.4.4. Control nivel intensitate lumină . . . . .	14
<b>4. Moduri alternative de funcționare și cazuri speciale</b>	<b>16</b>
4.1. Ecran LCD . . . . .	16
4.2. Citire pe portiuni a senzorilor . . . . .	17
4.2.1. Analiză rezultate . . . . .	19
4.3. Colectare date valori viteză . . . . .	19
4.4. Avantaje dublare senzori . . . . .	21
4.4.1. Creșterea vitezei maxime la care poate fi detectat un vehicul . . . . .	21
4.4.2. Permiterea funcționării sistemului în ambele sensuri de circulație . . . . .	22
4.5. Soluție implementare sistem pentru cazul unei autostrăzi . . . . .	25
<b>5. Dezvoltări ulterioare și posibile limitări</b>	<b>27</b>
5.1. Problema numărării vehiculelor pe străzi cu mai multe benzi de circulație . . . . .	27
5.2. Problema calculării vitezei pe străzi cu mai multe benzi de circulație . . . . .	28

5.3. Caz particular pietoni . . . . .	28
5.4. Utilizare timere . . . . .	28
<b>6. Concluzie</b>	<b>30</b>
<b>Anexe</b>	<b>32</b>
<b>A. Fișiere sursă</b>	<b>32</b>
A.1. TaskProd . . . . .	32
A.2. TaskCons . . . . .	33
A.3. TaskLightRead . . . . .	35
A.4. TaskLCD . . . . .	36
<b>Bibliografie</b>	<b>37</b>

# Glosar

**LCD** Liquid Crystal Display. [16](#)

**PWM** Pulse-Width Modulation. [6](#), [14](#)

**RTOS** Real-Time Operating System. [9](#)

# Listă de figuri

1.1. Intrare vehicul în sistem . . . . .	2
1.2. Vehicul în stare intermediară . . . . .	2
1.3. Vehicul ieșind din porțiunea reprezentată . . . . .	2
2.1. Diagrama de context a sistemului în timp real pentru monitorizarea iluminatului public . . . . .	3
2.2. Modul infraroșu . . . . .	4
2.3. Funcționare modul infraroșu . . . . .	5
2.4. Fotorezistor . . . . .	5
2.5. Microcontroller Arduino Mega . . . . .	6
2.6. Schema sistemului, realizată în Tinkercad . . . . .	7
2.7. Modelul implementat în realitate . . . . .	7
3.1. Timp citire date . . . . .	8
3.2. Diagramă task-uri . . . . .	10
3.3. Situație 2 vehicule pe aceeași porțiune de drum, moment 1 . . . . .	12
3.4. Situație 2 vehicule pe aceeași porțiune de drum, moment 2 . . . . .	12
4.1. LCD . . . . .	16
4.2. TaskLCD . . . . .	17
4.3. Calcul eronat al vitezei . . . . .	18
4.4. Comportarea sistemului mod 1: distanța între senzori 20 cm . . . . .	19
4.5. Comportarea sistemului mod 2: distanța între senzori 35 m . . . . .	20
4.6. Eroare posibilă la viteze mari . . . . .	20
4.7. Eroare posibilă la viteze mici . . . . .	21
4.8. Diagramă task-uri pentru funcționare pe timp de zi . . . . .	22
4.9. Cresterea vitezei maxime la care poate fi detectat un vehicul, prin dublarea senzorilor . . . . .	23
4.10. Sistem funcțional pentru ambele sensuri de circulație, moment 1 . . . . .	23
4.11. Sistem funcțional pentru ambele sensuri de circulație, moment 2 . . . . .	24
4.12. Noua distanță folosită la calcularea vitezei . . . . .	25
4.13. Model sistem autostradă, moment 1 . . . . .	26
4.14. Model sistem autostradă, moment 2 . . . . .	26
4.15. Model sistem autostradă, moment 3 . . . . .	26
5.1. Activare simultană a 2 senzori . . . . .	27
5.2. Sugestie sistem pentru detectarea pietonilor . . . . .	28

# 1. Introducere

În cadrul acestei teme, am dorit simularea unui sistem de iluminare publică, cu ajutorul unui microcontroller de tip Arduino Mega. Cel mai comun tip de sistem pentru iluminat stradal întâlnit în România la momentul actual este alcătuit din lămpi, care în timpul zilei nu sunt puse în funcțiune, dar rămân aprinse pe toată durata nopții. Deși iluminatul stradal este necesar pe timpul nopții, metoda aceasta de implementare este foarte ineficientă. Sistemul propus va păstra utilitatea celui deja existent, dar va ajuta la optimizarea consumului de energie. În studiile din [1, 5], bazate pe modele asemănătoare cu cel realizat de mine, implementarea unui astfel de sistem poate duce la scăderea consumului de energie electrică cu 40-45%. Nu poate fi făcută o comparație directă din cauza diferențelor în logica de funcționare, dar câștigul tot poate fi estimat a fi semnificativ.

## 1.1. Formularea problemei generale

Scopul lucrării este de a realiza un sistem alcătuit dintr-o serie de lămpi pentru iluminat stradal, care se aprind la intensitate mică doar când nivelul de luminozitate măsurat este destul de scăzut încât să poată fi considerat că este noapte. Tot pe timp de noapte, intrarea unui obiect în raza unui felinar duce la creșterea intensității luminii acestuia, în funcție de viteza obiectului.

Obiectivele principale ale lucrării sunt implementarea practică a sistemului, analiza calitativă a funcționării acestuia și tratarea, măcar teoretică, a cazurilor excepționale pe care nu le-am realizat în practică din motive ce țin de limitări hardware, software sau de timp.

Este important de menționat faptul că, **pe durata nopții**, felinarele nu se vor stinge niciodată în întregime pentru sistemul realizat de mine. Deoarece acesta este cazul studiat cel mai mult, **orice mențiune a stingerii lor va însemna trecerea pe modul de intensitate minimă de funcționare**. Voi face diferențierea între cele două, în cazurile în care aceasta este relevantă.

Prezența vehiculelor este detectată cu ajutorul unor senzori de prezență aflați între fiecare set de câte 2 felinare, iar acestea acționează astfel:

- La momentul depășirii de către vehicul al unui senzor, felinarul următor se va aprinde, gradul de luminozitate urmând să fie ales în funcție de viteza automobilului;
- Felinarul anterior acestuia urmează să se stingă, gradual.

În următoarele imagini este prezentat modul dorit de funcționare pentru cazul simplu al trecerii unui singur vehicul, în soluția propusă de mine pentru realizarea sistemului. La

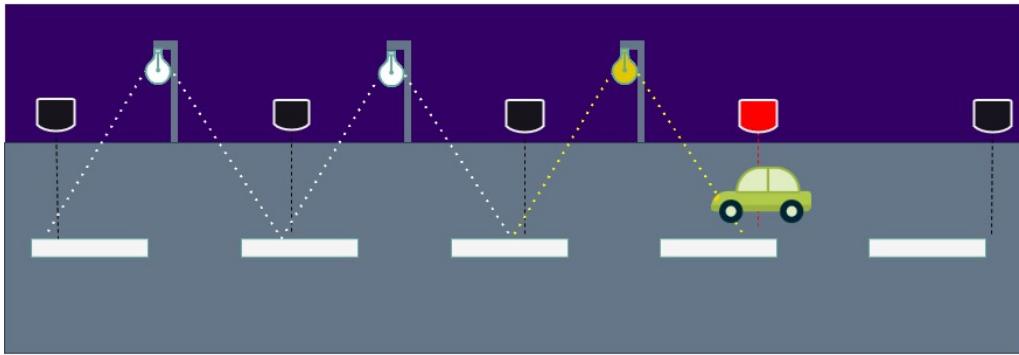


Figura 1.1.: Intrare vehicul în sistem

depășirea primilor 2 senzori se va aprinde primul felinar, în concordanță cu viteza calculată, după cum arată Figura 1.1.

Trecerea de următorul senzor, prezentată în Figura 1.2, duce la aprinderea următorului felinar, în timp ce nivelul de luminozitate al celui anterior începe să scadă treptat.

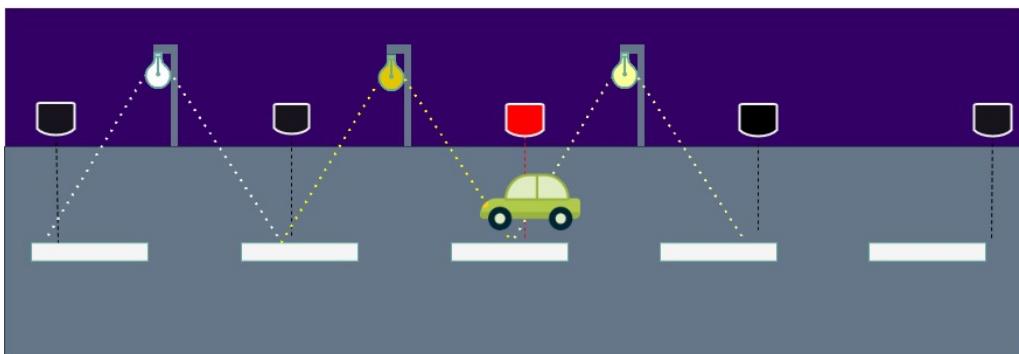


Figura 1.2.: Vehicul în stare intermediară

În Figura 1.3 am surprins momentul ieșirii vehiculului din sistem, în care ultimul felinar începe să se stingă, cel anterior încă nu a avut timp să se stingă complet, iar primul deja a revenit la starea inițială.

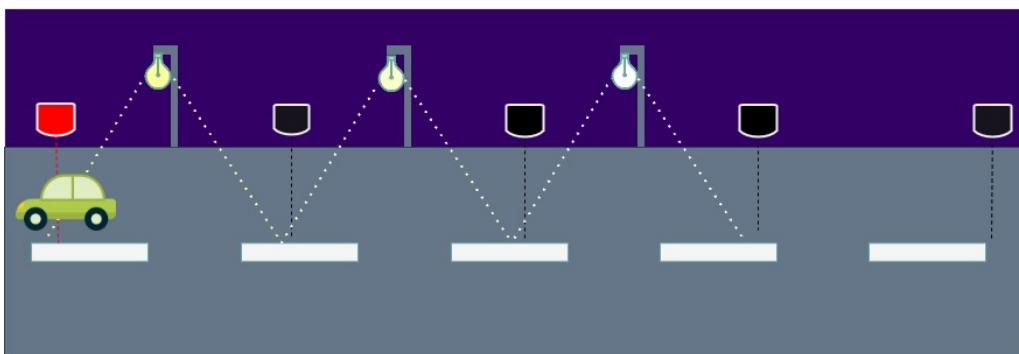


Figura 1.3.: Vehicul ieșind din portiunea reprezentată

## 2. Elemente ale arhitecturii hardware

### 2.1. Diagrama de context

Modul de funcționare al sistemului este prezentat în diagrama de context din Figura 2.1.

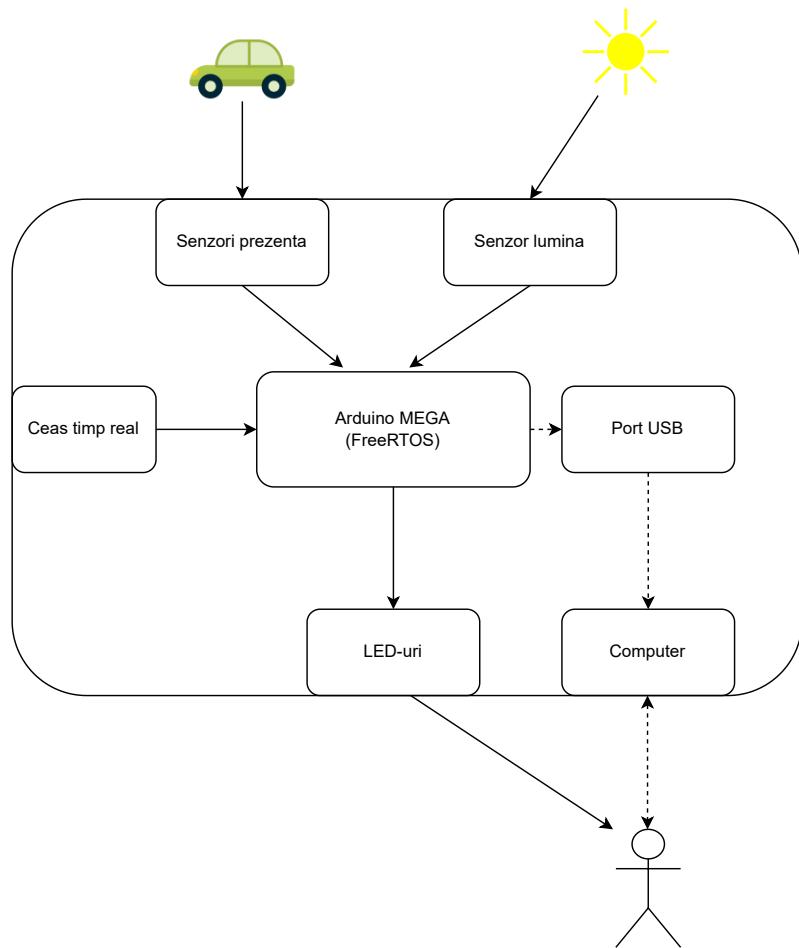


Figura 2.1.: Diagrama de context a sistemului în timp real pentru monitorizarea iluminatului public

Componentele sistemului în timp real sunt următoarele, conform diagramei de context:

- 5 module cu senzor infraroșu;
- 3 diode de tip LED;

- Un fotorezistor de tip LDR 5528;
- Microcontroller Arduino Mega;
- Un ceas în timp real;
- Port USB;
- Computer.

## 2.2. Prezentare elemente diagramă context

### 2.2.1. Modul senzor poziție

Am folosit 5 astfel de module HW-201, ilustrat în Figura 2.2, pentru detectarea prezenței vehiculelor de pe stradă. Acesta dispune și de un potențiometru, care permite ajustarea razei de detecție a obiectelor la valoarea dorită din intervalul 2-30 cm.

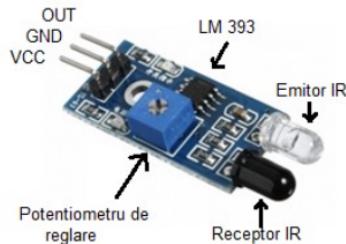


Figura 2.2.: Modul infraroșu

Elementul emițător eliberează constant un fascicul de lumină infraroșie, care se reflectă și ajunge la receptor atunci când există un obiect destul de aproape, după cum este prezentat în Figura 2.3. O mențiune legată de acest mod de funcționare este faptul că, la folosirea a 2 senzori pe părți opuse ale drumului, aceștia trebuie reglați din potențiometru pentru a avea raza de acțiune mai mică decât jumătate din lungimea drumului. În caz contrar, se vor activa reciproc.

O limitare a senzorilor ce se bazează pe detectarea radiațiilor infraroșii este că nu pot fi folosiți pe timp de zi fară a fi influențați negativ de lumina soarelui (e.g. se activează fără nimic în fața lor). Acest lucru nu afectează funcționarea dorită a sistemului, fiind luat în considerare că intrările de la acești senzori sunt citite doar noaptea, ci este doar un inconvenient la realizarea testării pe timp de zi.

În cazurile în care sunt alăturați 2 senzori, distanța dintre aceștia trebuie să fie destul de mare comparativ cu raza de detecție aleasă, pentru a nu permite activarea nedorită a unui senzor de reflecția fasciculului infraroșu al senzorului activat anterior, sau de cel de după el. O altă soluție este acoperirea parțială a receptorului fiecărui senzor, pentru a permite doar detecția fasciculelor venite de la unghiuri mici, adică de la propriul emițător.

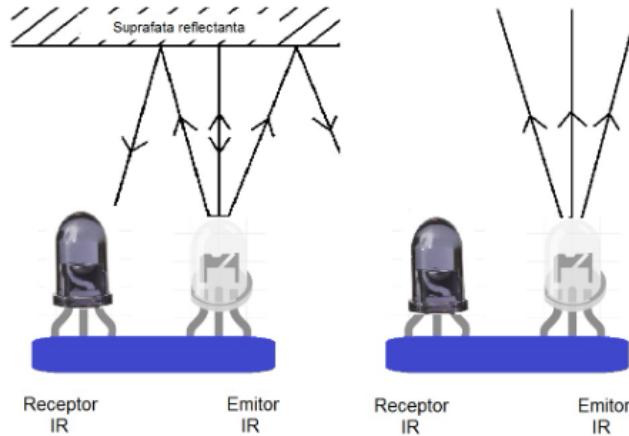


Figura 2.3.: Funcționare modul infraroșu

### 2.2.2. Fotorezistor

Am ales acest element (Figura 2.4), pentru a diferenția între momentele când este zi sau noapte. Fotorezistorul este un tip de rezistență ce variază în funcție de gradul de luminozitate aplicat suprafeței acestuia. El este conectat în serie cu o rezistență fixă de  $10k\Omega$ , iar valoarea analogică a tensiunii măsurate pe acest rezistor este convertită de Arduino Mega într-un număr natural din intervalul  $[0,1024]$ . Compararea acestei valori cu o constantă aleasă arbitrar va determina separarea între cele două moduri de funcționare.



Figura 2.4.: Fotorezistor

### 2.2.3. LED-uri

Am folosit 3 diode de tip LED pentru simularea felinarelor în realizarea proiectului. Am conectat în serie câte o rezistență de  $220 \Omega$  la anodul fiecărei dintre ele.

### 2.2.4. Arduino Mega

Placa de dezvoltare Arduino Mega, ilustrată în Figura 2.5, realizează calculele necesare funcționării corespunzătoare a sistemului. Aceasta dispune și de un ceas în timp real, care va fi util pentru dezvoltarea aplicației folosite.

Timpul este măsurat cu ajutorul unui ceas oscillator ce generează semnale drept-unghiulare cu frecvență, în cazul unui microcontroller Arduino, de 16MHz. Numărarea

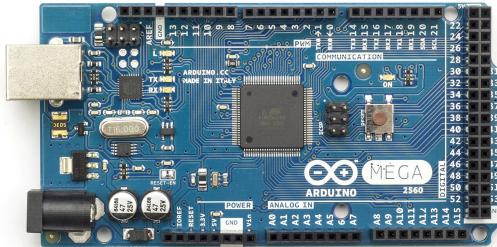


Figura 2.5.: Microcontroller Arduino Mega

repetărilor acestui semnal duce la determinarea cu acuratețe a intervalelor de timp, până la dimensiuni de  $\mu\text{s}$ , conform [6]. Tot prin microcontroller se realizează și posibila conexiune cu un computer. Cel din urmă nu este strict necesar pentru funcționarea sistemului, dar cu ajutorul său se deschide accesul la citirea de către utilizator a datelor de la senzori și la modificarea codului sursă. Poate fi înlocuit cu o baterie sau altă sursă cu tensiunea recomandată între 7-12V DC.

Deși nu dispune de ieșiri analogice, 15 dintre pinii digitali de pe Arduino Mega permit utilizarea de **Pulse-Width Modulation (PWM)**. Această metodă permite transmiterea parțială a curentului la LED-uri, prin mijloace digitale. Alternarea între valori de 0 și 1 creează un semnal dreptunghiular, iar controlul duratei impulsului acesta permite simularea unui semnal analogic [7]. Această metodă va permite controlul intensității luminii felinarelor.

### 2.3. Simulare a sistemului în Tinkercad și realizarea fizică

În Figura 2.6 este reprezentată o schemă a modului de conectare a componentelor descrise mai devreme, realizată de mine în Tinkercad. Este de menționat faptul că această platformă nu dispune de toate piesele necesare. Prin urmare, am înlocuit placă Arduino Mega cu un Arduino UNO, iar senzorii de prezență sunt reprezentați printr-o versiune ce dispune doar de receptor. În schimb, modul de conectare va rămâne același și pentru componentele ce au fost schimbate, doar pinii digitali folosiți pentru conectarea senzorilor vor varia față de modelul meu.

Toate componentele sunt conectate la pinul GND, iar senzorii de prezență și fotorezistorul sunt conectați și la pinul de alimentare cu 5V. Citirea datelor se va face pe pinii digitali 49, 50, 51, 52, 53 (8, 9, 10, 11, 12 în Figura 2.6), de pe Arduino Mega, pentru senzorii de prezență și de la pinul pentru intrări analogice A0 pentru fotorezistor. Intensitatea cu care se vor aprinde LED-urile va fi transmisă pe pinii digitali 5, 6, 7, prin **PWM**.

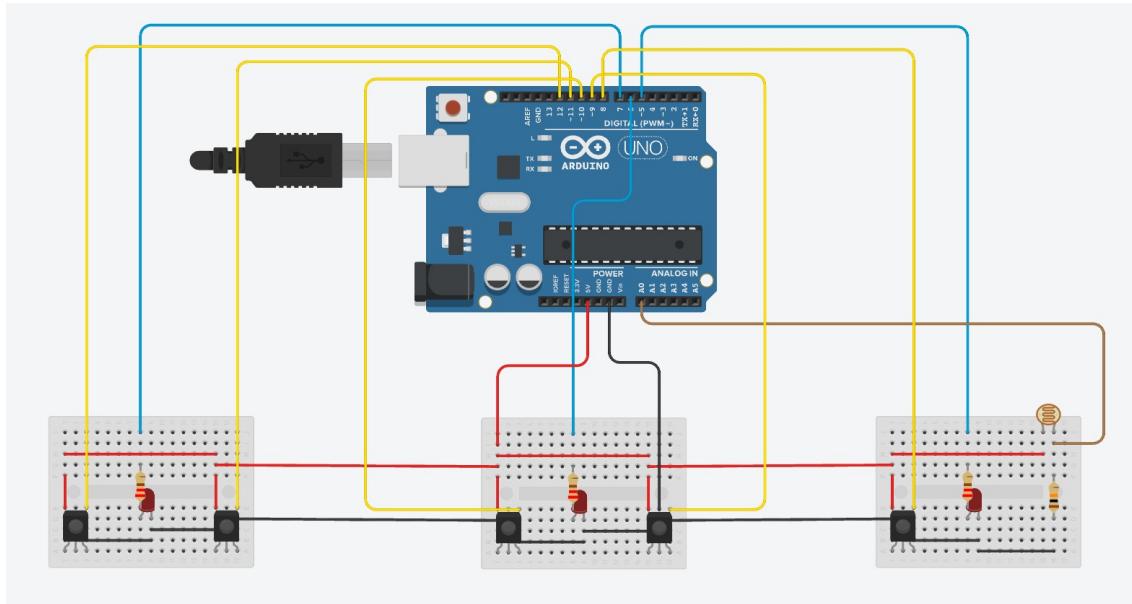


Figura 2.6.: Schema sistemului, realizată în Tinkercad

În Figura 2.7 este modelul realizat de mine, conform schemei prezentate anterior. Acoperirea acestuia ajută la blocarea parțială a luminii soarelui la testarea pe timp de zi, iar spațiul lăsat liber permite accesul unui cablu de alimentare la microcontroller. Distanța dintre doi senzori consecutivi este  $d = 14\text{cm}$ , egală cu cea dintre 2 felinare consecutive. Poziția fotorezistorului, în partea stângă a modelului, a fost aleasă pentru ca valoarea citită de la acesta să fie influențată cât mai puțin de aprinderea LED-urilor.

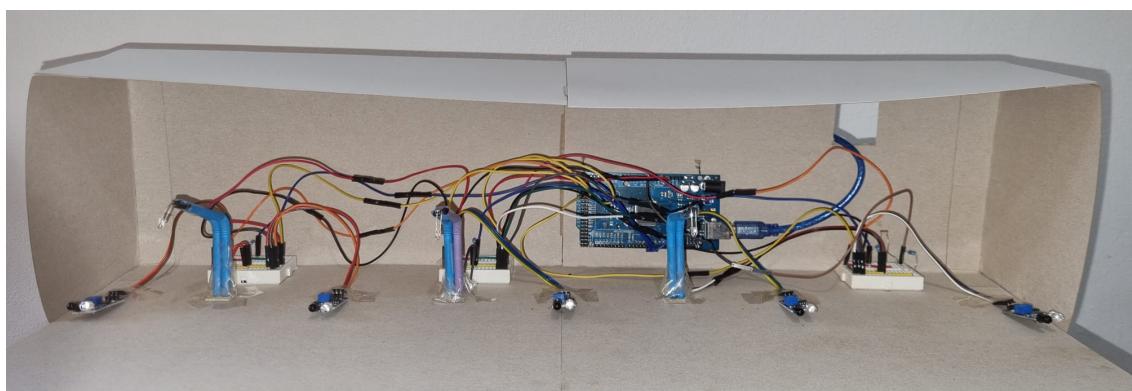


Figura 2.7.: Modelul implementat în realitate

## 3. Elemente ale arhitecturii software

### 3.1. Achiziție date și constrângeri de timp

Una dintre caracteristicile principale ale unui sistem în timp real este faptul că acesta trebuie să răspundă schimbărilor venite din mediul înconjurător în limita unui interval de timp, numit *deadline* [4]. Pentru funcționarea corespunzătoare a sistemului, sunt necesare date din 2 surse diferite: senzorii ce detectează prezența unui obiect și senzorul de luminozitate.

În cazul senzorilor de prezență, este important ca intervalul de timp dintre 2 citiri să fie mai mic decât cel necesar trecerii în întregime a unui vehicul prin fața acestuia. Dacă aș considera suprafața în care senzorii detectează un obiect a fi de lățime neglijabilă, pot afla viteza necesară pentru ca vehiculul să treacă nedetectat. În crearea programului Arduino după care funcționează sistemul, am observat faptul că intervalul de timp dintre două citiri ale aceluiași senzor este de 15-17 ms (puțin peste 1 tick).

În exemplul ilustrat în Figura 3.1, am analizat cazul extrem al unei motociclete de lungime relativ mică, 190cm. La convertirea din cm/ms în km/h a rezultat faptul că aceasta ar trebui să atingă o viteză de aproximativ 402km/h pentru a depăși un senzor complet nedetectată. Prin urmare, frecvența citirilor este suficient de mare pentru a acomoda cerințele sistemului real, și chiar pentru a permite unele modificări aduse cu scopul de a îmbunătăți performanțele sau de a adăuga utilități extra.

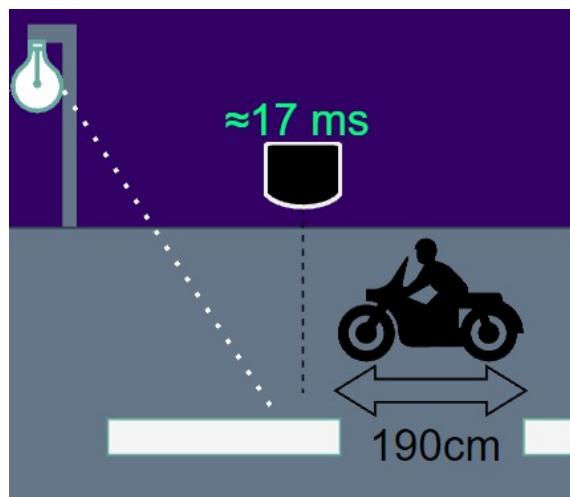


Figura 3.1.: Timp citire date

Pentru lumină, nu contează atât de mult cât de des se realizează achiziția de date de la senzor, deoarece valoarea citită oricum variază lent. Totuși, și aceasta se va citi la

fiecare repetare a task-ului dedicat ei, deoarece trebuie să țină pasul cu citirile senzorilor. Motivul pentru acest fapt este prezentat mai detaliat în [3.3.1](#).

## 3.2. Descriere FreeRTOS

Pentru a îndeplini cerințele sistemului în timp real pe care l-am simulat, și aplicația ce determină funcționarea acestuia trebuie să poată folosi concepte specifice programării în timp real [3]. Este necesară utilizarea firelor de execuție și a semafoarelor pentru a asigura funcționarea în conformitate cu performanțele dorite. De aceea folosirea unui **Real-Time Operating System (RTOS)** este obligatorie. Deși aceasta nu este o caracteristică de bază a Arduino IDE, includerea bibliotecii FreeRTOS permite utilizarea funcțiilor necesare creării și gestiunii firelor de execuție (sub formă de task-uri), a semafoarelor și a mutex-urilor. Această bibliotecă deschide accesul și la funcția vTaskDelay, ce oferă posibilitatea blocării unui anumit task, fără afectarea celorlalte task-uri. Poate fi apelată pentru o durată de timp specificată ori în număr de ticks, ori în milisecunde, prin conversie.

## 3.3. Prezentare task-uri

În cadrul aplicației am folosit 3 task-uri, ce funcționează după secvență logică din Figura [3.2](#). Ele au aceeași prioritate și se execută în același timp. Ordinea efectuării operațiilor din fiecare task, relativ la celelalte task-uri, este influențată prin incrementarea și decrementarea semafoarelor, unde este necesar.

### 3.3.1. TaskLightRead

În task-ul acesta ([A.3](#)) se realizează citirea intrării analogice de pe pin-ul plăcii Arduino la care este conectat fotorezistorul. Dacă valoarea acesteia depășește o anumită constantă, se consideră că afară este zi, urmând reinitializarea tuturor variabilelor folosite la valori nule, pentru a nu rămâne cu cele obținute anterior. Dacă felinarele erau aprinse, sau pe modul de intensitate scăzuta, aici se sting **complet**. În cazul în care valoarea citită este sub limita impusă, se incrementează semaforul binar SemNOAPTE, cu scopul de a debloca TaskProd.

Deși acesta este modul normal de funcționare al task-ului, în codul sursă ([A.3](#)) al aplicației realizate de mine verificarea are loc invers: deblocarea celorlalte două task-uri se întâmplă doar pe timp de zi, pentru ușurarea procesului de testare.

### 3.3.2. TaskProd

Acest task producător ([A.1](#)) are rolul de a citi valorile de la senzorii de prezență, lucrul care se întâmplă imediat după deblocarea sa, în urma incrementării semaforului SemNOAPTE. Mai departe, la prima rulare, se decrementează semaforul binar SemPLIN, dar se trece, totuși, mai departe, deoarece acesta este inițializat cu valoarea "1". Același lucru este

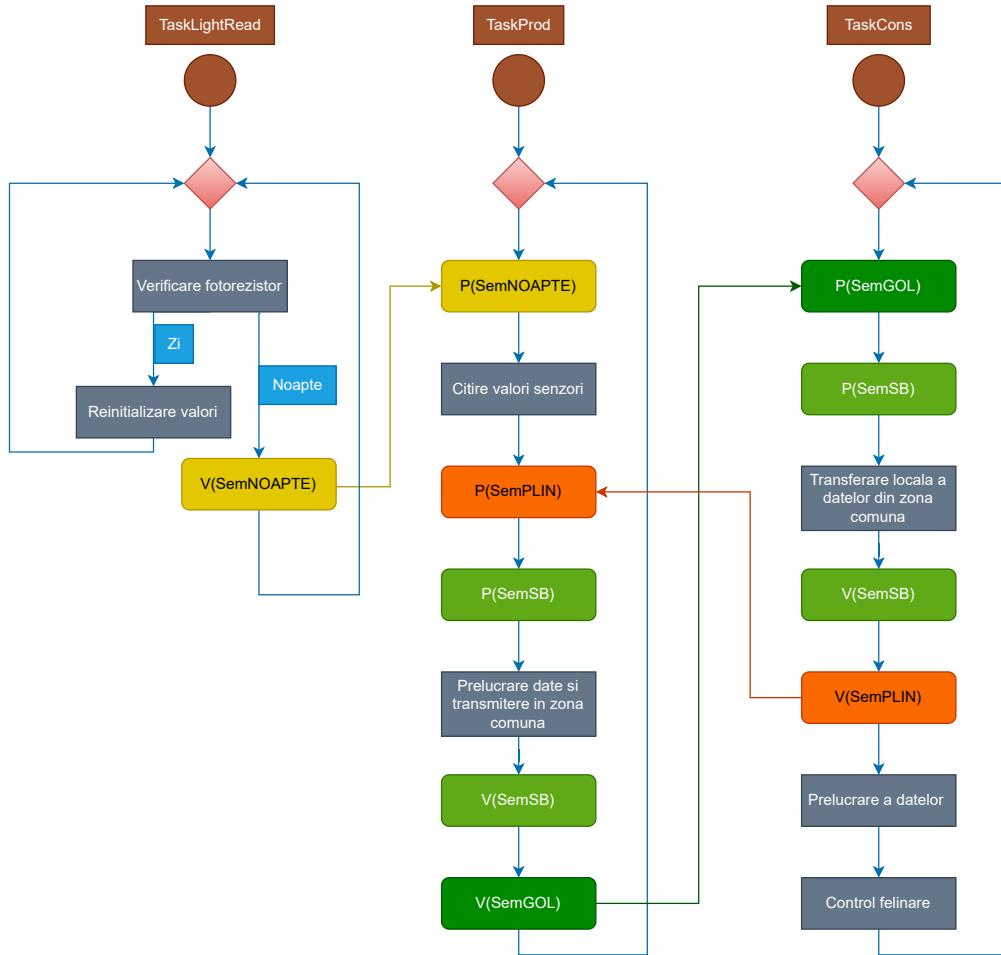


Figura 3.2.: Diagramă task-uri

valabil și pentru semaforul pentru secțiunea critică, SemSB, care asigură faptul că zona comună de date nu este accesată simultan de mai multe task-uri [3]. În urma alterării și transferării noilor date, acesta este deblocat. Apoi, semaforul binar SemGOL este incrementat, pentru deblocarea task-ului TaskCons. La rulări ulterioare ale task-ului, acesta se va bloca fie în SemNOAPTE, dacă s-a făcut zi, sau în SemPLIN, dacă nu a fost incrementat din TaskCons.

### 3.3.3. TaskCons

Deblocarea task-ului consumator (A.2) are loc la incrementarea în TaskProd a semaforului SemGOL. Urmează trecerea datelor din zona comună în plan local, facilitată de semaforul SemSB. În urma incrementării lui SemPLIN, ce permite deblocarea lui TaskProd, are loc prelucrarea datelor obținute și controlul felinarelor conform condițiilor impuse. În final, task-ul se repetă, în concordanță cu toate cele menționate până acum.

### 3.4. Analiză a codului

În această secțiune voi menționa și descrie câteva fragmente mai importante din codul sursă, cu scopul de a oferi o privire mai clară asupra modului de funcționare al aplicației. Toate variabilele scalare sau vectoriale vor fi inițializate în întregime cu 0, cu excepția unora dintre semafoare, conform Secțiunii 3.3. Constanta  $d$  reprezintă distanța (în cm) dintre 2 senzori consecutivi și este egală cu 14. În cazul buclelor for, acestea în general se repetă de  $n = 5$  ori dacă sunt folosite valorile corespondente senzorilor de prezență, sau de  $n - 1 = 4$  ori, pentru operații ce influențează calculul sau afișarea vitezei pe o anumită porțiune de drum.

#### 3.4.1. Prelucrare date citite și counter obiecte

Modulele cu senzor infraroșu folosite pot avea ca ieșiri valorile "1", când nu se află nimic în fața lor, sau "0", când au un obiect în față. Aceste valori numerice nu ajută foarte mult, deoarece scopul este de a obține viteza vehiculului. De aceea va fi utilizată funcția millis() din Arduino IDE, pentru a obține timpul (în ms) trecut de la momentul rulării aplicației până la activarea unui senzor.

În următorul fragment de cod este prezentată zona de *Prelucrare date și transmitere în zona comună* din TaskProd. În aplicația realizată de mine, drumul este împărțit în  $n-1$  porțiuni egale ca dimensiune, aflate între cei  $n$  senzori de prezență folosiți.

```
1   for(int in = 0; in < 5; in ++){  
2       if (f[in] == 1){  
3           b[in]= bn[in];  
4           bn[in] = 0;  
5           f[in] = 2;  
6       }  
7       if (aux[in] - bc[in] == 1) {  
8           if (in <= 3) {  
9               c[in]++;  
10          }  
11          if ((in >= 1) && c[in-1]>0){  
12              c[in - 1] --;  
13              if ((bn[in-1] != 0) &&(c[in - 1] == 1)){  
14                  f[in-1] = 1;  
15              }  
16          }  
17          if (in == 4)  
18              b[in] = millis();  
19          else if ((in <= 3) && (c[in] == 1)){  
20              b[in] = millis();  
21          }  
22          else if ((in <= 3) && (c[in] > 1)){  
23              bn[in] = millis();  
24          }  
}
```

```

25         }
26     aux[in] = bc[in];
27 }

```

Numărul de vehicule de pe fiecare porțiune de drum este salvat în vectorul counter  $c[n - 1]$ , iar acesta este actualizat la fiecare depășire a unuia dintre senzori de un vehicul. Este important ca atât modificarea valorii din  $c$ , cât și cea a momentului de timp înregistrat, să se întâpte o singură dată la fiecare activare a unui senzor de prezență. Pentru a îndeplini această cerință cu acuratețe, trebuie recunoscute doar momentele când un obiect apare în fața unui senzor, sau, mai precis, când valoarea citită de la senzor se transformă din "1" în "0". Vectorul  $bc[n]$  conține valorile obținute în secțiunea de Citire valori senzori, iar egalarea elementelor din  $aux[n]$  la finalul buclei for face ca la următoarea execuție a task-ului să pot compara valoarea actuală cu cea citită cu un pas în urmă. Diferența dintre cele două poate fi egală cu 1 doar în cazul dorit, cel în care  $aux[in] = 1$  și  $b[in] = 0$ . În cazul unui singur vehicul, momentul la care a fost activat un senzor este salvat în  $b[n]$ . Am adăugat variabilele vectoriale  $f[n]$  și  $bn[n]$ , pentru tratarea cazului în care mai multe vehicule se află pe aceeași porțiune de drum. Un exemplu al unei astfel de situații este reprezentat în Figura 3.3 și Figura 3.4 .

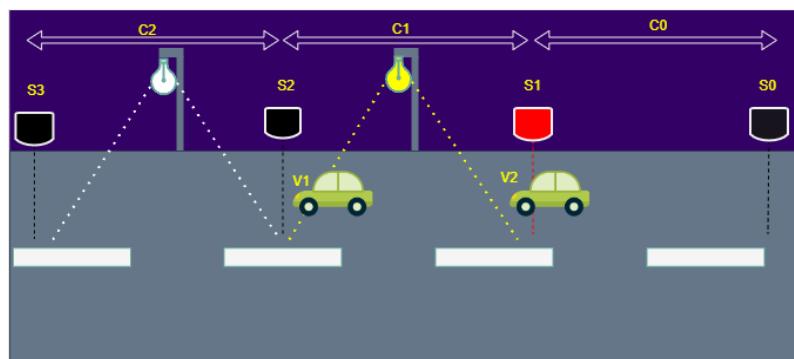


Figura 3.3.: Situație 2 vehicule pe aceeași porțiune de drum, moment 1

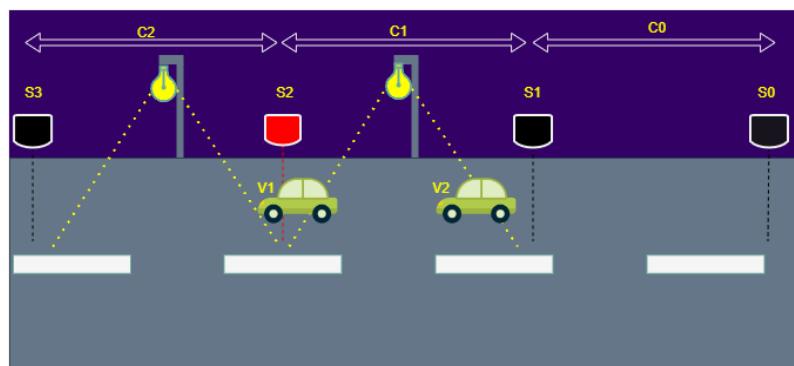


Figura 3.4.: Situație 2 vehicule pe aceeași porțiune de drum, moment 2

În această situație, comportamentul aplicației pe porțiunea de cod descrisă este următorul:

1. La depășirea senzorului S1 de vehiculul V2, counter-ul **c[1]** este incrementat și are valoarea 2. Momentul de timp este salvat în vectorul **bn[1]**.
2. La activarea lui S2, momentul este salvat în **b[2]**, iar flag-ul **f[1]** primește valoarea 1.
3. La următoarea execuție a task-ului, acest flag indică faptul că valoarea din **b[1]** nu mai este necesară, deoarece a fost folosită deja în calculul unei viteze. Ea va fi înlocuită cu valoarea din **bn[1]**, iar **f[1]** va deveni 2. În această stare, el indică faptul că următorul calcul al vitezei cu noul **b[1]** este doar unul intermedian și nu va influența luminozitatea felinarelor.

Această metodă de implementare acoperă și cazul apariției unui vehicul V3 în porțiunea C1, înaintea ieșirii lui V2 din aceasta, prin eliberarea lui **bn[1]** pentru trecerea lui V3 prin S1. Modelul realizat funcționează doar până la limita de 2 vehicule existente simultan în aceeași porțiune. Pentru generalizare, vectorul **bn** ar trebui să fie o matrice de  $n$  coloane și  $m$  linii, unde  $m$  este un număr natural, cu 1 mai mic decât numărul de automobile maxime admise într-o porțiune a drumului, ales în funcție de lungimea porțiunii respective.

### 3.4.2. Transmitere date între task-uri

Pentru folosirea datelor primite de la senzorii de prezență, este necesar transferul între task-uri prin folosirea unor vectori declarați global, astfel:

```
1 for (int i = 0; i < 5; i++) {  
2     z[i] = b[i];  
3     zn[i] = bn[i];  
4 }
```

### 3.4.3. Transformare date senzori mișcare în valori de timp și în viteză

Pentru aflarea vitezei, se scad momentele de timp la care au fost activați 2 senzori consecutivi și se realizează transformarea prin calcule matematice, în funcție de unitatea de măsură dorită că rezultat. În cazul meu, conversia este din cm/ms în km/h. Diferențierea între valorile lui **c** ale secțiunii în care intră vehiculul se face pentru a determina dacă valoarea de timp corespondentă acestuia trebuie luată din **z** sau **zn**. Deși viteza este calculată la fiecare rulare a task-ului în **vint[n - 1]**, aceasta este salvată în **v[n - 1]** și afișată doar la momentul activării unui senzor, numai în situațiile în care flag-ul **f** nu este egal cu 2. Dacă acesta este 2, va fi reinitializat cu 0. În **auxv[n - 1]** este salvată valoarea vitezei intermediiare la execuția anterioară a task-ului, pentru comparare cu cea actuală, similar cu **aux** în Subsecțiunea 3.4.1.

```
1 for (int i = 0; i < 4; i++) {  
2     if (i==3)  
3         t[i] = z[i + 1] - z[i];  
4     else if (c[i + 1] == 1)
```

```
5         t[i] = z[i + 1] - z[i];
6     else if (c[i + 1] == 2)
7         t[i] = zn[i + 1] - z[i];
8
9     if (t[i] != 0)
10        vint[i] = 10*d * 3.6/((float)t[i]);
11    else if (t[i] == 0)
12        vint[i] = 0;
13    if (auxv[i] != vint[i]){
14        if ((f[i] != 2)&&(vint[i]>=0)){
15            v[i] = vint[i];
16            Serial.print ...
17        }
18        else {
19            Serial.print ...
20            f[i] = 0;
21        }
22    }
23    auxv[i] = vint[i];
24    ...
25 }
```

### 3.4.4. Control nivel intensitate lumină

Pentru controlul intensității luminii, este folosită o abordare bazată pe [PWM](#). Aceasta permite folosirea ieșirilor digitale corespondente LED-urilor, pe post de ieșiri analogice ([Subsecțiunea 2.2.4](#)), convertite din valori numerice aparținând intervalului [0,255]. Aceste valori sunt păstrate în vectorul **lum[n – 2]**, și sunt influențate de viteza calculată a vehiculelor, astfel:

```
1   for (int i = 0; i < 4; i++) {
2       ...
3       if (i > 0){
4           if (lum[i - 1] < 3)
5               lum[i - 1] = 3;
6           if ((c[i] == 0) && ( lum[i - 1] > 3)) {
7               if (timp == 0) {
8                   if (lum[i - 1] > 8)
9                       lum[i - 1] = lum[i - 1] * 0.7;
10                  else
11                      lum[i - 1] --;
12              }
13              timp++;
14              timp = timp % 5;
15          }
16          else if (c[i] == 0) {
17          }
18          else if (c[i] == 1){
```

```
19         if (v[i-1] > 4){
20             lum[i - 1] = 255;
21         }
22         else if (v[i-1] > 1){
23             if (lum[i - 1] > 150)
24                 lum[i - 1] = lum[i - 1] - 2;
25             else
26                 lum[i - 1] = 150;
27         }
28         else if (v[i - 1] > 0){
29             if (lum[i - 1] > 40)
30                 lum[i - 1] = lum[i - 1] - 2;
31             else
32                 lum[i - 1] = 40;
33         }
34     }
35     else if (c[i] == 2){
36         if ((v[i - 1] > 4)&& (lum[i - 1] < 150)){
37             lum[i - 1] = 255;
38         }
39         else if ((v[i - 1] > 1) && (lum[i - 1] < 150)){
40             lum[i - 1] = 150;
41         }
42     }
43     analogWrite(4 + i, lum[i - 1]);
44 }
45 }
```

LED-urile primesc ca ieșire valoarea afărată în **lum**, în cazul în care valoarea counter-ului **c** corespondent portiunii lor de drum este diferită de 0. Când aceasta devine iar 0 (atunci când nu se mai află niciun vehicul în raza felinarului), valoarea din **lum** începe să scadă treptat, până devine 3. Valorile din **lum** se schimbă doar la modificări aduse lui **v**, iar valorile din **vint** nu au niciun efect asupra lor.

Diferențierea dintre cazurile în care **c[i] = 1** și **c[i] = 2** se face pentru a asigura faptul că este oferită o prioritate mai mare vehiculelor cu viteze mai mari, la stabilirea intensității felinarelor. Pentru momentul 1 al exemplului descris în Subsecțiunea 3.4.1 (Figura 3.3), trecerea lui V2 prin fața lui S1 va determina schimbarea intensității luminii felinarului doar dacă viteza acestuia este mai mare decât viteza pe care a avut-o V1 la trecerea prin S1. În cazul contrar, valoarea din **lum** va începe să scadă treptat doar la ieșirea lui V1 din portiunea C1, la momentul 2(Figura 3.4).

## 4. Moduri alternative de funcționare și cazuri speciale

După cum am menționat în [Secțiunea 3.1](#), frecvența citirilor senzorilor de prezență este principalul factor ce limitează numărul de posibilități ale sistemului. În modelul prezentat, fiind o versiune la scară mult redusă, frecvența citirilor aduce două probleme principale:

1. Obiectele folosite pentru a simula vehiculele devin mult mai scurte;
2. Distanțele mai mici între senzori duc la calcule eronate în cazul obiectelor cu viteze mari.

Sistemul real permite scăderea frecvenței citirilor, fară a ieși din limitele performanțelor dorite, în schimbul adăugării unor caracteristici suplimentare.

### 4.1. Ecran LCD

Sistemul permite adăugarea unui [Liquid Crystal Display \(LCD\)](#), ce permite observarea anumitor date intermedii (e.g. viteza unui vehicul în fiecare zonă a sistemului), fară a fi necesară conectarea la computer. Acesta folosește un modul I2C și se conectează la 5V și GND împreună cu restul componentelor, iar adițional, la pinii dedicați SDA și SCL de pe Arduino Mega.



Figura 4.1.: LCD

Pentru utilizarea [LCD](#)-ului, este nevoie de un task suplimentar ([A.4](#)) ce se ocupă cu gestiunea acestuia. Incrementarea semaforului binar SemLCD are loc în TaskProd, la schimbarea oricărei dintre viteze. Acest lucru asigură funcționarea corectă a programului și oprește rescrierea datelor de pe [LCD](#) la fiecare execuție a task-ului. Astfel, acestea sunt rescrise doar când apare o schimbare. Verificarea și incrementarea valorii variabilei *p* are loc pentru a asigura faptul că inițializarea display-ului se întâmplă doar la prima rulare a task-ului, ilustrat în Figura [4.2](#).

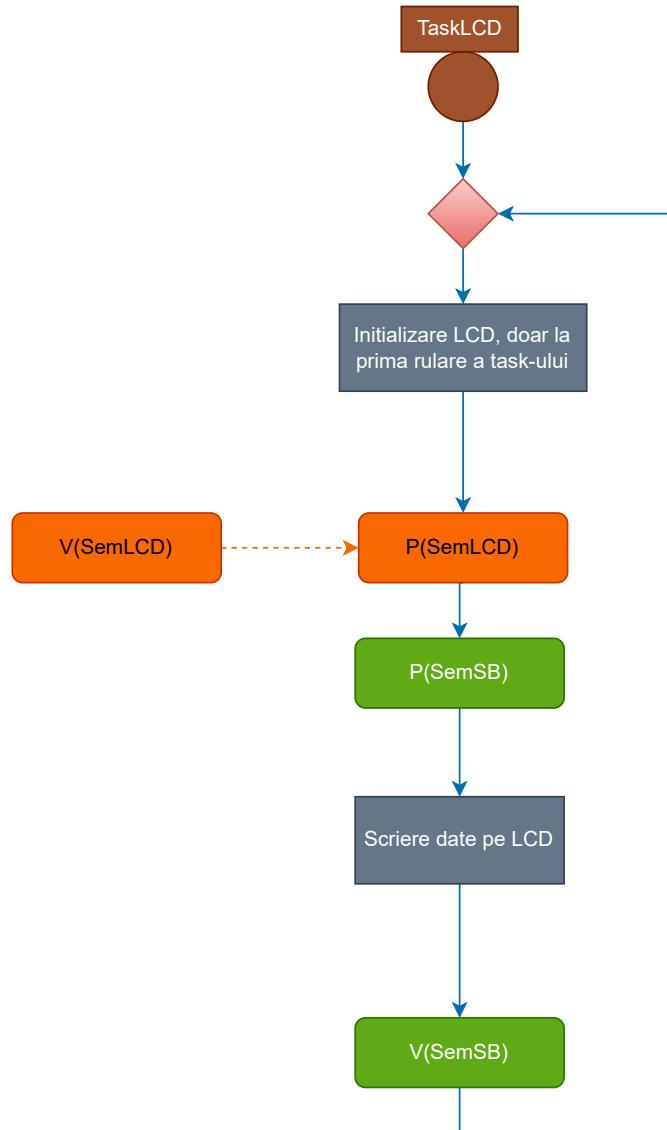


Figura 4.2.: TaskLCD

Prezența și funcționarea acestuia necesită sincronizări adiționale între task-uri și un mic delay adăugat la citirile senzorilor. La încercarea rulării în starea inițială, simpla scriere pe LCD bloca tot programul. În cazul soluției găsite de mine pentru rezolvarea problemei, intervalul de timp dintre două citiri a ajuns să fie aproximativ dublu, în jur de 33-34ms.

## 4.2. Citire pe portiuni a senzorilor

O versiune alternativă a aplicației pentru sistemul de iluminat în timp real este cea în care fiecare senzor este citit individual, într-o rulare a task-ului TaskPROD, iar în TaskCONS se calculează doar o singură viteză, cu ajutorul valorii de timp primite la momentul

actual și cea de la execuția anterioară a task-ului. Această metodă este ușor de generalizat. Modificările necesare pentru extinderea sistemului constau în schimbarea dimensiunii buffer-ului din zona comună de date și a transformării semafoarelor binare SemPLIN și SemGOL din Figura 3.2 în semafoare generalizate. Atât dimensiunea vectorului de date, cât și a semafoarelor va deveni egală cu numărul nou de senzori de prezență [3].

În schimb, aici se întâlnește problema scăderii frecvenței citirilor. La citirea unui singur senzor la fiecare rulare a lui TaskCONS, intervalul de timp dintre două citiri ale aceluiași senzor se multiplică cu  $n$ , unde  $n$  este egal cu numărul senzorilor de prezență. Astfel, citirile se realizează la fiecare aproximativ 85ms, în loc de cele 17ms necesare inițial. Acest lucru nu numai că face viteza necesară pentru ca un vehicul să treacă neobservat de  $n$  ori mai mică decât cea calculată în Secțiunea 3.1, dar duce și la calcularea greșită a vitezei vehiculului.

În Figura 4.3 este reprezentat un exemplu de calcul al vitezei cu citirile aceluiași senzor la un interval de 85 ms, după cum arată variația variabilei  $t$ . În acest caz, este calculată viteza  $v[0]$ , iar distanța dintre 2 senzori este destul de mică(20cm) pentru a se obține valori foarte mici pentru  $t$ (variabilă scalară pentru modalitatea de implementare din această secțiune). Viteza se calculează prin împărțire la  $t$ , ceea ce înseamnă că diferențele mari între valorile sale, relativ la el însuși, duc la obținerea unui rezultat ce nu reflectă realitatea. În exemplul prezentat, asta înseamnă că  $v[0]$  va fi afișat ca 42.35 km/h pentru orice viteză mai mare sau egală cu acest număr, 7.06km/h pentru orice valoare din intervalul [7.06, 42.35) ș.a.m.d.

```

t = 17  t1 = 59291  t2 = 59274  v[0] = 42.35
v[1] = -0.01
.nf
.nf
t = 102  t1 = 59376  t2 = 59274  v[0] = 7.06
v[1] = -0.01
.nf
.nf
t = 188  t1 = 59462  t2 = 59274  v[0] = 3.83
v[1] = -0.01
.nf
.nf
t = 273  t1 = 59547  t2 = 59274  v[0] = 2.64
v[1] = -0.01
.nf
.nf
t = 359  t1 = 59633  t2 = 59274  v[0] = 2.01
v[1] = -0.01
.nf
.nf
t = 444  t1 = 59718  t2 = 59274  v[0] = 1.62
v[1] = -0.01
.nf
.nf
t = 530  t1 = 59804  t2 = 59274  v[0] = 1.36
v[1] = -0.01
.nf
.nf
t = 615  t1 = 59889  t2 = 59274  v[0] = 1.17

```

Figura 4.3.: Calcul eronat al vitezei

#### 4.2.1. Analiză rezultate

Rezultatele de pe Serial Monitor, din Figura 4.3 pot fi observate și în Figura 4.4, sub formă de grafic.

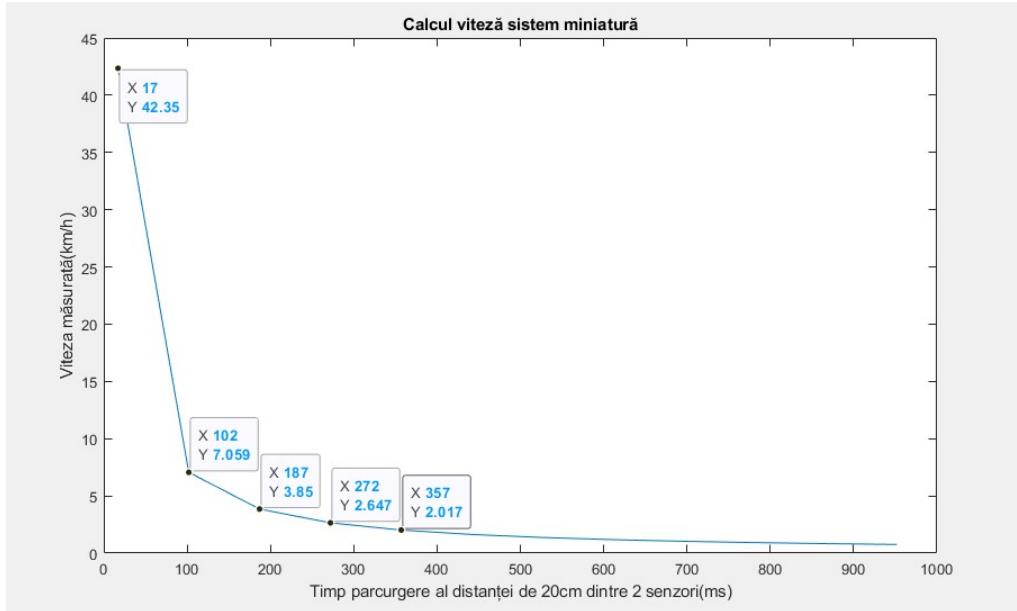


Figura 4.4.: Comportarea sistemului mod 1: distanța între senzori 20 cm

În Figura 4.5 am realizat același calcul al vitezei, la modificarea distanței dintre senzori la 35m, în concordanță cu sistemul real. Forma graficului rămâne aceeași, iar singura diferență este creșterea valorii vitezei, direct proporțional cu distanța.

În schimb, dacă extind graficul până la momentele de timp în care viteză ar avea valori mai realiste pentru un automobil, ca în Figura 4.6 și Figura 4.7, se poate observa cum eroarea maximă posibilă, egală cu diferența între două viteze consecutive de pe grafic, începe să scadă.

În concluzie, eroarea maximă posibilă pentru măsurarea vitezei devine din ce în ce mai mică, la scăderea vitezei obiectului observat, iar acest proces poate fi grăbit prin mărirea distanței dintre senzori. Astfel, aceste valori pot fi ajustate pentru un sistem de iluminare publică, în funcție de intervalul numeric în care poate fi încadrată viteză vehiculelor pe acea porțiune de drum.

### 4.3. Colectare date valori viteză

Din moment ce sistemul, în starea sa de bază, deja calculează vitezele vehiculelor pe fiecare porțiune de drum, nu este dificilă modificarea acestuia pentru a salva într-o variabilă vectorială numărul automobilelor ce depășesc o anumită viteză în porțiunea respectivă. Deși am stabilit în Secțiunea 4.2 faptul că viteză măsurată nu este niciodată exactă cu această metodă, valorile obținute sunt destul de apropiate de cele reale pentru a servi, de

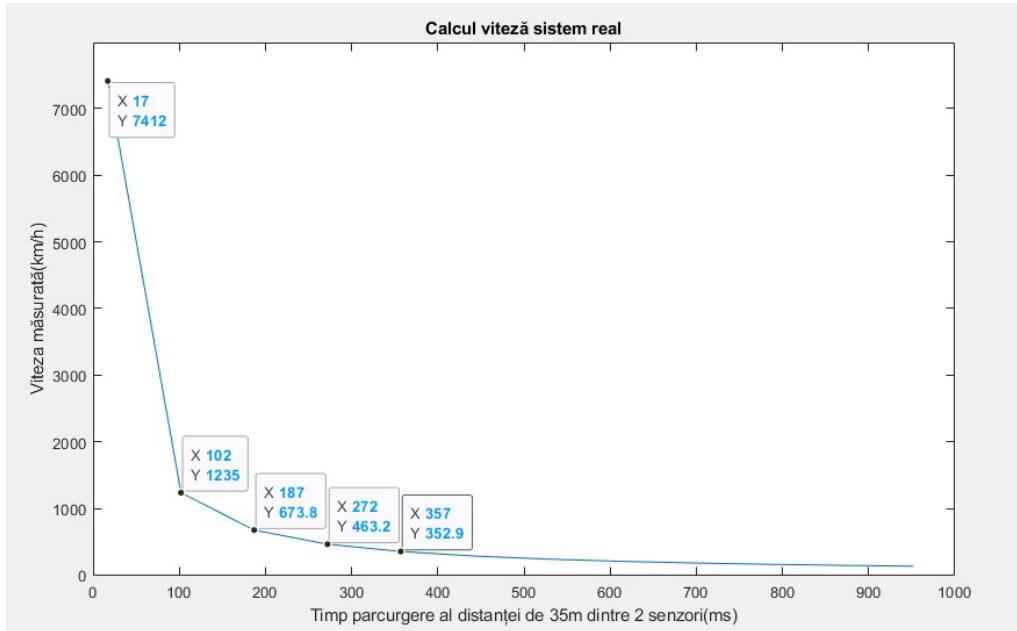


Figura 4.5.: Comportarea sistemului mod 2: distanță între senzori 35 m

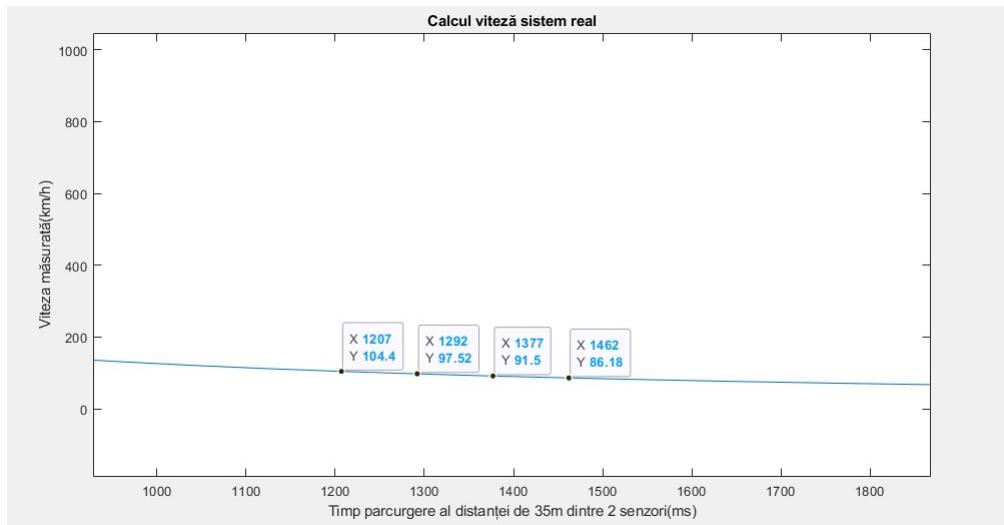


Figura 4.6.: Eroare posibilă la viteze mari

exemplu, autorităților locale. Datele pot fi folosite, împreună cu statisticile avute deja(e.g. locații frecvente ale accidentelor din trafic) la alegerea locațiilor pentru puncte de instalare a camerelor fixe, sau a punctelor mobile de măsurat viteza.

Factorul ce limitează posibilitatea acestui mod de utilizare al sistemului, este faptul că funcționează doar noaptea. Astă înseamnă că achiziția de date nu se poate realiza pe timp de zi. Pentru a face acest lucru posibil, este nevoie modificări aduse sistemului, atât la partea hardware, cât și la cea software. În primul rând, reiese din prezentarea elementelor hardware din [Subsecțiunea 2.2.1](#) faptul că nu se pot folosi acești senzori de prezență pe timp de zi, fără a obține date eronate. De aceea este necesară folosirea altor modalități

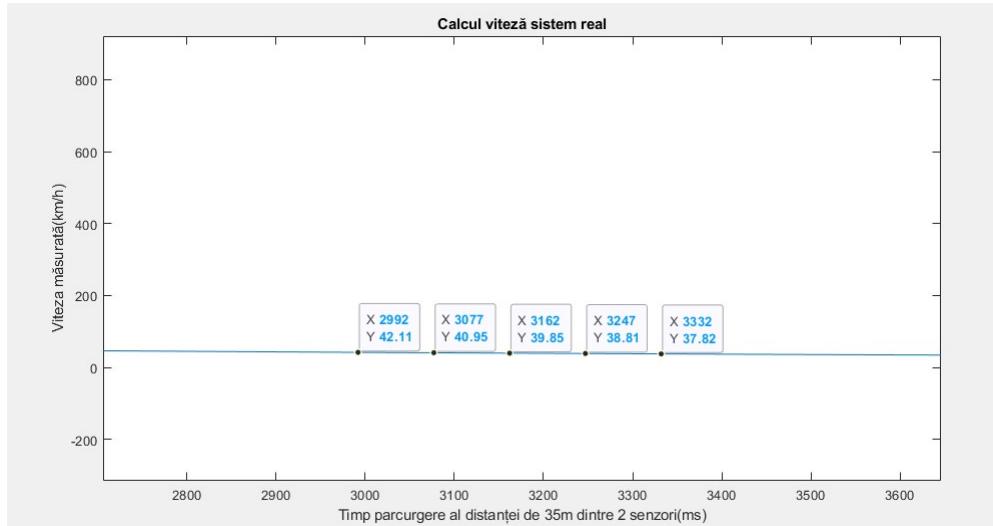


Figura 4.7.: Eroare posibilă la viteze mici

de detectare a prezenței, prin senzori care nu sunt influențați de radiații infraroșii. Un exemplu este utilizarea unor senzori ultrasonici [2], care oferă ca ieșire distanța dintre ei și un obiect. Viteza vehiculelor se poate calcula prin analiza variației acestei distanțe, la 2 momente de timp ulterioare activării unuia dintre senzori.

Pe urmă, trebuie aduse modificări structurii task-urilor, iar acestea sunt ilustrate în Figura 4.8. TaskLightRead nu mai reinitializează valorile variabilelor, ci doar modifică valoarea unei variabile  $p$ . Aceasta îndeplinește rolul de flag, pentru a determina dacă este zi sau noapte. Dacă valoarea sa indică faptul că este zi, partea de *Control felinare* din TaskCons va egala cu 0 valorile din vectorul de luminozitate, pentru stingerea completă a acestora.

## 4.4. Avantaje dublare senzori

DUBLAREA SENZORILOR constă în folosirea a doi senzori de prezență pe porțiunea de drum în care, în mod normal, să fi folosit unul singur. În funcție de distanța dintre ei și de modul în care sunt utilizate datele citite, aceștia pot îndeplini mai multe scopuri.

### 4.4.1. Creșterea vitezei maxime la care poate fi detectat un vehicul

Majoritatea elementelor adiționale adăugate sistemului vor duce la scăderea frecvenței citirii senzorilor de prezență, ceea ce poate cauza trecerea neobservată a vehiculelor, fenomen explicat în Secțiunea 3.1.

Problema poate fi rezolvată prin înlocuirea fiecărui senzor cu o grupare de doi senzori aflați la distanța  $k$  între ei. În această situație, distanța pe care trebuie să o parcurgă un vehicul în intervalul de timp dintre două citiri ale senzorilor, pentru a nu fi detectat, crește cu o valoare egală cu  $k$ .

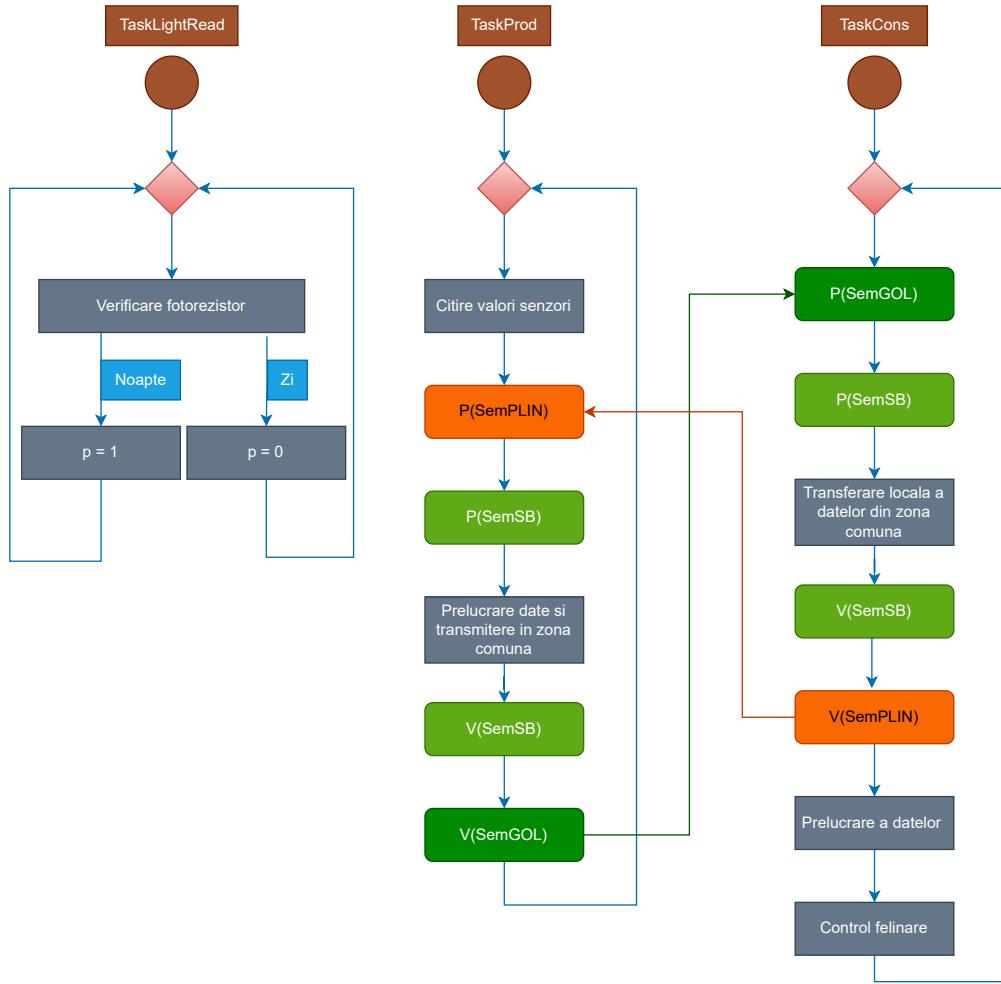


Figura 4.8.: Diagramă task-uri pentru funcționare pe timp de zi

În Figura 4.9 este ilustrat cazul unui vehicul ce nu a fost văzut de primul senzor, dar următoarea citire a avut loc înainte ca acesta să-l depășească și pe al doilea. În acest caz, considerând  $k = 150\text{cm}$  și citirea senzorilor la fiecare  $85\text{ ms}$ , viteza maximă cu care un vehicul cu lungimea de  $190\text{cm}$  poate să se deplaseze fără să fie vreodată neobservat crește de la  $80.47\text{ km/h}$  la  $144\text{km/h}$ . Counter-ul ce determină aprinderea următorului felinar se va comporta la fel ca în situația activării primului senzor, dar se va stoca într-o variabilă de tip flag care dintre senzori au fost activați, pentru a ajusta distanța folosită la calcularea vitezei.

#### 4.4.2. Permiterea funcționării sistemului în ambele sensuri de circulație

Un alt lucru făcut posibil de dublarea senzorilor este identificarea sensului de deplasare al unui vehicul. În continuare este prezentat cazul unei străzi cu două benzi și sensuri opuse de circulație. În Figura 4.10 este prezentat momentul depășirii unui senzor de un vehicul aflat pe sensul său de circulație, iar Figura 4.11 este reprezentarea aceluiași vehicul în urma trecerii pe contrasens, în cazul unei depășiri.

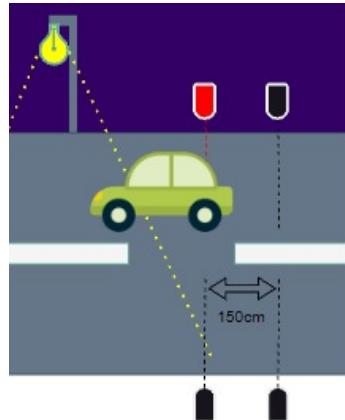


Figura 4.9.: Creșterea vitezei maxime la care poate fi detectat un vehicul, prin dublarea senzorilor

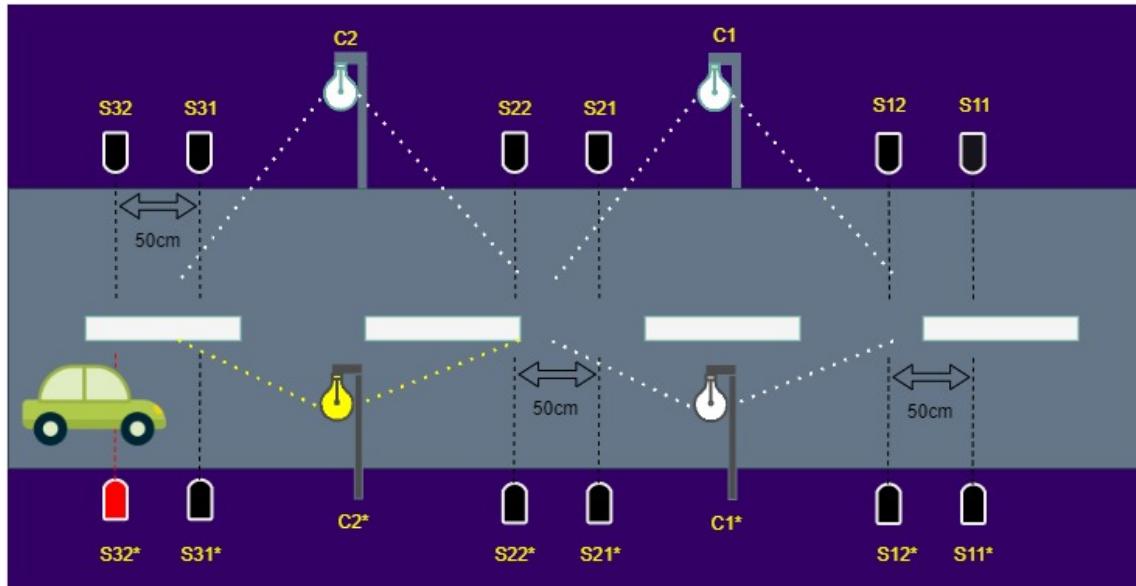


Figura 4.10.: Sistem funcțional pentru ambele sensuri de circulație, moment 1

Primul senzor întâlnit în fiecare pereche pe sensul normal ar circulației ( $S_{11}$ ,  $S_{21}$ ,  $S_{31}$ ,  $S_{32}^*$ ,  $S_{22}^*$ ,  $S_{12}^*$ ) este folosit pentru incrementarea sau decrementarea variabilelor counter  $c$  și pentru măsurarea vitezei automobilului, iar valoarea citită de cel de-al doilea ( $S_{12}$ ,  $S_{22}$ ,  $S_{32}$ ,  $S_{31}^*$ ,  $S_{21}^*$ ,  $S_{11}^*$ ) are rolul de a stabili sensul de deplasare. Deoarece înregistrarea momentului de timp la care un vehicul depășește un senzor se face doar la modificarea valorii citită de la acesta (ref), pot verifica tot atunci valoarea citită de la senzorul ajutător. Dacă aceasta indică prezența unui obiect, înseamnă că vehiculul se află pe contrasens. În această situație, se întâmplă următoarele:

1. Se incrementează un element al unui counter  $cs$ , corespunzător grupării de senzori ( $S_{21}-S_{22}$ ).
2. Se verifică starea counter-ului grupării de senzori anterioare, relativ la sensul de

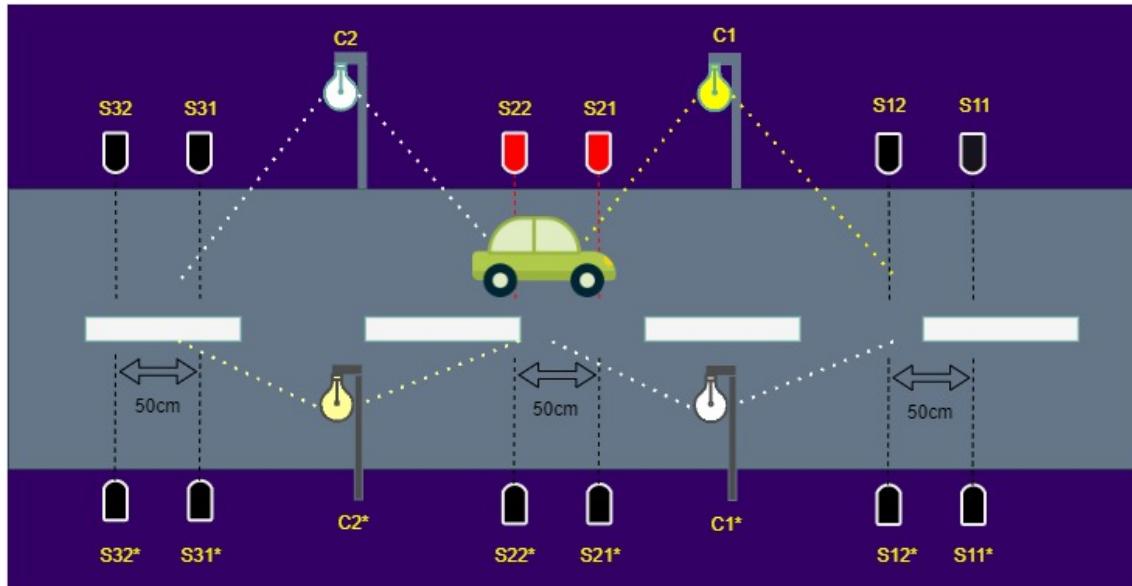


Figura 4.11.: Sistem funcțional pentru ambele sensuri de circulație, moment 2

deplasare al vehiculului, pe același sens (în acest caz, S31-S32, și egal cu 0). Dacă aceasta este:

- egală cu 0 - ultimul senzor depășit de vehicul a fost pe cealaltă bandă, deci se decrementează **c** pentru cel mai apropiat felinar de pe acea bandă (aici, C2\*)
- mai mare ca 0 - ultimul senzor depășit de vehicul a fost pe aceeași bandă deci se decrementează **c** pentru cel mai apropiat felinar depășit de acesta (aici, C2).

3. Se incrementează elementul counter **c** pentru stâlpul următor(C1).
4. Se decrementează valoarea **cs** analizată mai sus în urma calculului vitezei, dacă aceasta este mai mare ca 0.

Este important să fie cunoscut sensul de circulație pe care se află vehiculul la depășirea senzorului anterior până la momentul calcului vitezei, deoarece trecerea pe celălalt sens duce la schimbarea valorii folosite pentru distanța dintre senzori. Noua distanță  $d'$  este reprezentată în Figura 4.12 și, cunoscând lățimea  $w$  a drumului, se află prin:

$$d' = \sqrt{(w/2)^2 + (d + 50cm)^2} \quad (4.1)$$

O problemă ce trebuie rezolvată pentru implementarea unui astfel de sistem este calculul distanței optime dintre senzorii ce aparțin aceluiași perechi. Aceasta trebuie să fie cât mai mică posibil, pentru a asigura imposibilitatea unui vehicul de a activa doar unul dintre senzori la trecerea pe contrasens. În același timp, ea trebuie să fie și destul de mare pentru ca vehiculul să nu aibă timp să ajungă la al doilea senzor din pereche, înainte să fie detectat de primul, pe sensul normal de mers. În ambele cazuri negative, rezultatul este incrementarea counterului **c** al felinarului din spatele vehiculului, în loc de cel din față sa. Pentru folosirea unei distanțe destul de mici între senzori, este nevoie de o frecvență mare

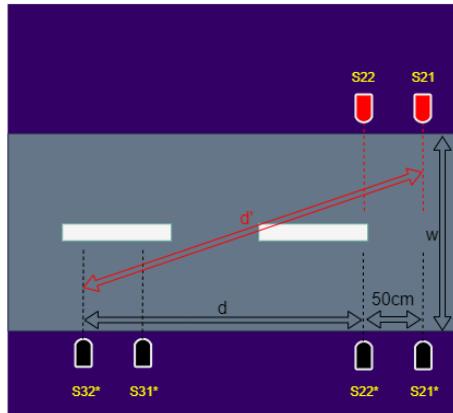


Figura 4.12.: Noua distanță folosită la calcularea vitezei

a citirilor, sau de implementarea sistemului pe drumuri cu limite legale mici pentru viteza de circulație.

#### 4.5. Soluție implementare sistem pentru cazul unei autostrăzi

Pentru abordarea cazului implementării sistemului în timp real pe o autostradă cu 4 benzi de circulație(2 pentru fiecare sens, cu separare la mijloc), trebuie rezolvate următoarele probleme:

- Vitezele mai mari aduc și erori mai mari în măsurarea lor, conform analizei realizate în Subsecțiunea 4.2.1.
- Vehiculele pot atinge viteze care le permit depășirea neobservată a senzorilor.
- Variabilele counter din **c** pot să primească valori greșite în momentul trecerii a două mașini în paralel.
- Aprinderea unui singur felinar în fața unui vehicul ce se deplasează cu 130-140 km/h poate fi deranjant pentru ochii șoferului și poate cauza accidente din cauza vizibilității scăzute.

Soluția propusă de mine constă în câteva modificări aduse sistemului, în starea sa inițială, pentru a rezolva aceste probleme.

În primul rând, toți senzorii de prezență vor fi dublați, pentru creșterea vitezei maxime la care un vehicul poate fi detectat, ca în Subsecțiunea 4.4.1. Apoi, este dublată distanța dintre aceste grupări, pentru a mări precizia cu care este calculată viteza. În Figura 4.13, Figura 4.14 și Figura 4.15 sunt surprinse trei momente de interes din trecerea unui vehicul prin sistem. Fiecare senzor din aceste imagini reprezintă, în realitate, câte o pereche de senzori pe care le voi numi, pentru restul secțiunii, grupuri. Pentru a număra corect automobilele din fiecare porțiune a drumului, se folosește câte un grup pentru fiecare bandă de circulație, amândouă influențând valoarea counter-ului **c** corespunzător acelorași felinare. Problema vizibilității poate fi rezolvată prin asigurarea faptului că un automobil va avea în orice moment cel puțin două felinare apropiate în fața sa.

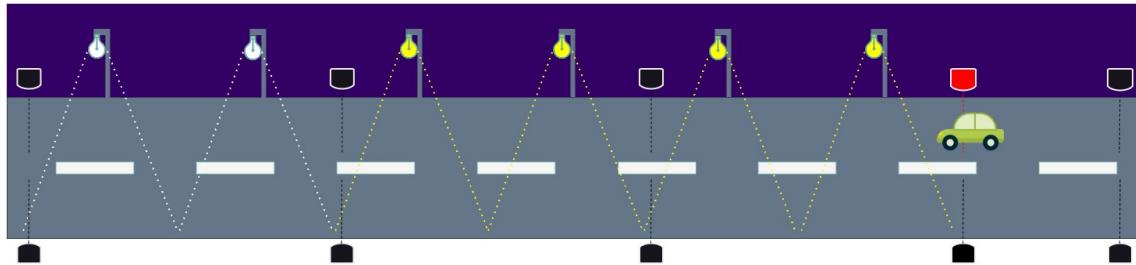


Figura 4.13.: Model sistem autostradă, moment 1

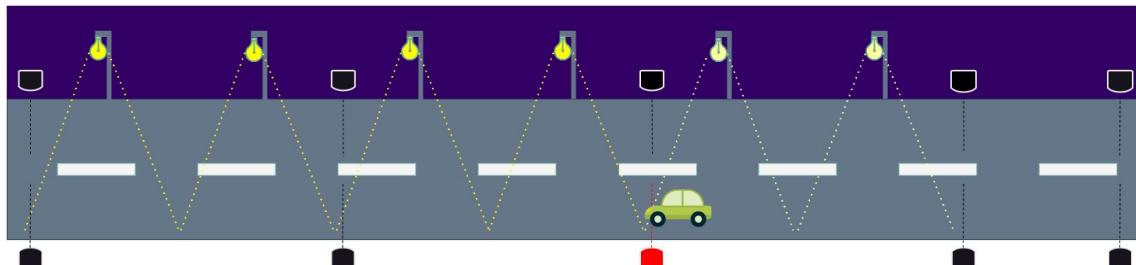


Figura 4.14.: Model sistem autostradă, moment 2

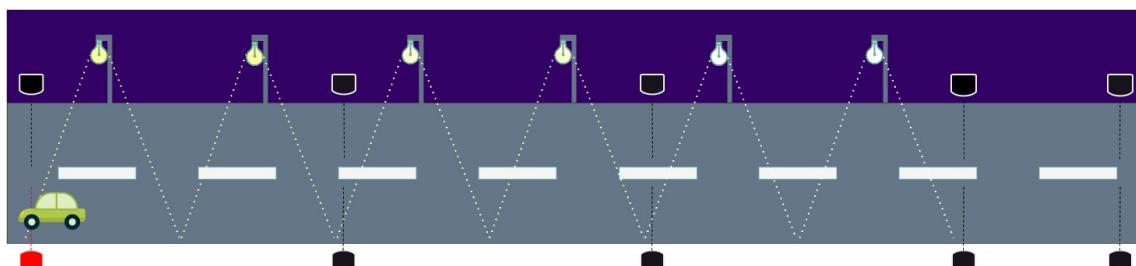


Figura 4.15.: Model sistem autostradă, moment 3

După cum este prezentat în imagini, la momentul depășirii unui grup, cele 2 felinare din urma acestuia se vor stinge, următoarele două vor rămâne aprinse, iar al 3-lea și al 4-lea după acesta se vor aprinde.

Din moment ce felinarele funcționează în grupuri de câte două, și dimensiunea vectorului  $\mathbf{c}$  poate fi înjumătățită. Nu este necesar a fi luat în calcul cazul circulației în sens opus, deoarece pe fiecare bandă a autostrăzii sensul este unic. Astfel, o versiune identică a sistemului poate fi instalată și pentru celelalte două benzi.

## 5. Dezvoltări ulterioare și posibile limitări

Ca scopuri pentru dezvoltarea ulterioară a proiectului pot menționa testarea și implementarea ideilor din [Capitolul 4](#), dar și rezolvarea problemelor cărora nu le-am găsit încă o soluție.

### 5.1. Problema numărării vehiculelor pe străzi cu mai multe benzi de circulație

Una dintre aceste probleme este activarea senzorilor la momentul trecerii unui vehicul pe cealaltă bandă de circulație. În exemplul din [Figura 5.1](#), aplicat modelului din [Sectiunea 4.5](#), vehiculul ar incrementa counter-ul corespunzător următoarelor felinare de 2 ori cu o singură trecere.

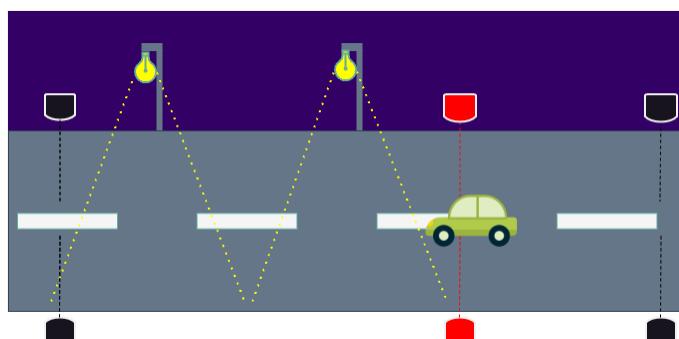


Figura 5.1.: Activare simultană a 2 senzori

Data viitoare când acesta se află în întregime pe o singură bandă, el va decrementa counter-ul din urmă doar cu 1, iar consecința va fi o desincronizare dintre valorile din counter și numărul real de vehicule de pe stradă (rezultat asemănător și în cazul situației din [Subsecțiunea 4.4.2](#)). Nici micșorarea razei de detecție a senzorilor nu rezolvă această problemă, deoarece sunt create situații în care un vehicul poate trece prin mijlocul drumului complet neobservat, cauzând aceeași desincronizare menționată anterior. Aici m-am gândit la suplimentarea cu metode de măsurat a distanței de la senzori, prin adăugarea, de exemplu, a senzorilor ultrasonici menționati în [Sectiunea 4.3](#). Un dezavantaj este faptul că nu am lucrat cu astfel de senzori, deci nu cunosc toate limitările ce pot fi întâlnite în practică.

Problema numărării vehiculelor apare și în cazul străzilor cu mai mult de 2 benzi pe același sens de circulație, deoarece 2 astfel de senzori plasați pe părți opuse ale străzii nu pot diferenția între 2 sau mai multe automobile care trec în același timp.

## 5.2. Problema calculării vitezei pe străzi cu mai multe benzi de circulație

Pe orice stradă pe care este posibilă depășirea vehiculelor, calculul vitezei devine greu de realizat, deoarece acesta se realizează cu presupunerea că primul vehicul care intră într-o porțiune de drum va fi și primul care ieșe din ea. Aceasta este, în opinia mea, cea mai mare limitare a sistemului în starea sa actuală. În schimb, chiar și în cazul eliminării componentei de măsurat viteza, performanțele obținute ar fi, în continuare, satisfăcătoare din punct de vedere al consumului de energie electrică. Pentru studiul din [5], unde consumul pe durata unei luni a scăzut cu 40%, felinarele funcționează la intensitate maximă la detecția unui vehicul, indiferent de viteza acestuia.

## 5.3. Caz particular pietoni

Detectia pietonilor poate fi considerată un caz particular de stradă cu mai multe benzi de circulație. Principalele diferențe sunt dificultatea în a determina comportamentul acestora și faptul că viteza lor nu este la fel de relevantă ca cea a vehiculelor. Senzorii pentru prezență destinați detectării pietonilor trebuie să fie separați de cei pentru automobile, pentru a nu influența măsurarea vitezei acestora, ca în Figura 5.2.

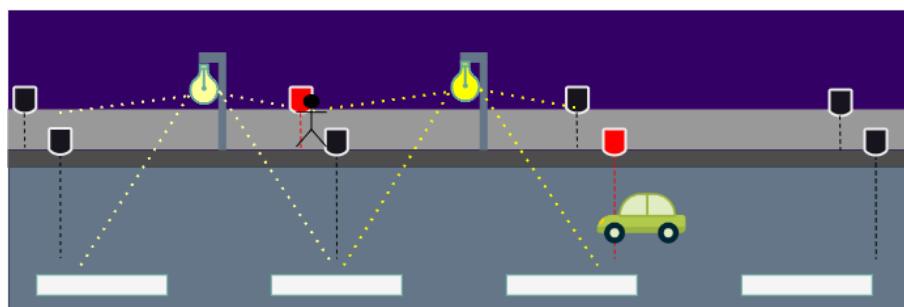


Figura 5.2.: Sugestie sistem pentru detectarea pietonilor

Direcția deplasării pietonilor nu poate fi prezisă cu certitudine. Aceștia se pot și opri sau părăsi drumul, de exemplu pentru a intra într-o clădire. De aceea, sunt de părere că trecerea pietonilor nu ar trebui să influențeze counter-ul ce determină funcționarea felinarelor, ci doar să aprindă cele două sau mai multe felinare din jurul senzorului activat, la intensitate medie, în cazul în care acestea nu erau deja aprinse. Pentru această metodă, este necesară adăugarea unui timer(5.4) , care să determine stingerea felinarelor la trecerea unei perioade de timp prestabilite.

## 5.4. Utilizare timere

Arduino IDE permite folosirea unor funcții specializate pentru realizarea operațiilor cu momente de timp, prin adăugarea de biblioteci ce se bazează pe funcțiile millis() sau

micros(). Atât folosirea unor timere ca unică metodă de control a stingerii felinarelelor, cât și îmbinarea unui astfel de element cu sistemul realizat, poate oferi performanțe mai bune în cazurile limită. Aceasta pare a fi chiar obligatorie pentru orice fel de astfel de sistem care are ca scop și detectarea pietonilor. În metoda mea de implementare am preferat să nu adaug timere, pentru a evidenția cât mai mult răspunsurile sistemului la stimuli externi, specifice unui sistem în timp real.

Un exemplu de posibilă îmbunătățire adusă de adăugarea unui timer este decrementarea automată a counter-ului **c** pentru o anumită porțiune de drum, la trecerea unei perioade de timp prestabilită de la ultima dată când a intrat un vehicul în ea. Aceasta ar duce la stingerea felinarului în cazul în care un vehicul se oprește între 2 senzori, sau dacă acesta, din posibile erori hardware, nu este detectat de senzorul de la ieșirea din porțiunea respectivă.

## 6. Concluzie

Pe parcursul lucrării nu am adus în evidență foarte mult potențialul câștig al sistemului din punct de vedere al reducerii consumului de energie electrică și a emisiilor de dioxid de carbon(CO<sub>2</sub>), ci am insistat mai mult pe implementarea în sine și pe limitările întâlnite în practică.

Am identificat elementele hardware necesare și am prezentat o metodă de implementare funcțională pentru un caz restrâns. Prin analiza realizată anterior am demonstrat că majoritatea acestor limitări sunt cauzate de calculul vitezei vehiculelor, pe tipuri de drum unde comportamentul acestora este imprevizibil. De aceea, o metodă de implementare similară și-ar putea avea locul în cazul străzilor cu sens unic, sau cu o singură bandă de circulație pentru fiecare sens, cu mențiunea că integrarea unor componente ce permit măsurarea distanței sau renunțarea în întregime la calculul vitezei ar putea deschide calea mai multor posibilități de utilizare.

Sunt de părere că implementarea unui astfel de sistem, în urma tratării cazurilor limită, oferă cel puțin condiții necesare siguranței rutiere, cu potențiale câștiguri la costurile operaționale, în funcție de densitatea traficului pe timpul nopții pe drumurile unde ar fi folosit.

Prin urmare, consider că un astfel de sistem ar putea fi implementat prin generalizarea soluției identificate.

## **Anexe**

# A. Fișiere sursă

## A.1. TaskProd

```
1 void TaskProd( void *pvParameters __attribute__((unused)) )
2 {
3     for (;;)
4     {
5         xSemaphoreTake(SemNOAPTE, portMAX_DELAY);
6
7         bc[0] = digitalRead(49);
8         bc[1] = digitalRead(50);
9         bc[2] = digitalRead(51);
10        bc[3] = digitalRead(52);
11        bc[4] = digitalRead(53);
12
13        xSemaphoreTake(SemPLIN, portMAX_DELAY);
14        xSemaphoreTake(SemSB, portMAX_DELAY);
15
16        for(int in = 0; in < 5; in ++){
17            if (f[in] == 1){
18                b[in]= bn[in];
19                bn[in] = 0;
20                f[in] = 2;
21            }
22            if (aux[in] - bc[in] == 1) {
23                if (in <= 3) {
24                    c[in]++;
25                }
26                if ((in >= 1) && c[in-1]>0){
27                    c[in - 1]--;
28                    if ((bn[in-1] != 0) &&(c[in - 1] == 1)){
29                        f[in-1] = 1;
30                    }
31                }
32                if (in == 4)
33                    b[in] = millis();
34                else if ((in <= 3) && (c[in] == 1)){
35                    b[in] = millis();
36                }
37                else if ((in <= 3) && (c[in] > 1)){
38                    bn[in] = millis();
```

```
39         }
40     }
41     aux[in] = bc[in];
42 }
43     xSemaphoreGive(SemSB);
44     xSemaphoreGive(SemGOL);
45
46     vTaskDelay(1);
47 }
48 }
```

## A.2. TaskCons

```
1 void TaskCons( void *pvParameters __attribute__((unused)) )
2 {
3     for (;;)
4     {
5
6         xSemaphoreTake(SemGOL, portMAX_DELAY);
// Transfer date buffer
7         xSemaphoreTake(SemSB, portMAX_DELAY);
8         for (int i = 0; i < 5; i++) {
9             z[i] = b[i];
10            zn[i] = bn[i];
11        }
12
13
14         xSemaphoreGive(SemSB);
15         xSemaphoreGive(SemPLIN);
16 // CALCULE
17         for (int i = 0; i < 4; i++) {
18             if (i==3)
19                 t[i] = z[i + 1] - z[i];
20             else if (c[i + 1] == 1)
21                 t[i] = z[i + 1] - z[i];
22             else if (c[i + 1] == 2)
23                 t[i] = zn[i + 1] - z[i];
24
25             if (t[i] != 0)
26                 vint[i] = 10*d * 3.6/((float)t[i]);
27             else if (t[i] == 0)
28                 vint[i] = 0;
29             if (auxv[i] != vint[i]){
30                 if ((f[i] != 2)&&(vint[i]>=0)){
31                     v[i] = vint[i];
32                     Serial.print(" t[");Serial.print(i);Serial.print("] = ");
33                     Serial.print(t[i]);
34                     Serial.print(" |||");
35                     for (int j = 0; j < 4; j++) {
```

```
36         if (j==i){
37             Serial.print("<v[");Serial.print(j);Serial.print("] = ");
38             Serial.print(v[j]);
39             Serial.print(">");
40         }
41         else {
42             Serial.print(" v[");Serial.print(j);Serial.print("] = ");
43             Serial.print(v[j]);
44             Serial.print(" ");
45         }
46     }
47     Serial.println("");
48 }
49 else {Serial.print(" z[");Serial.print(i);Serial.print("] = ");
50     Serial.print(z[i]);
51     Serial.print(" zn[");Serial.print(i);Serial.print("] = ");
52     Serial.print(zn[i]);
53     Serial.print(" z[");Serial.print(i+1);Serial.print("] = ");
54     Serial.print(z[i+1]);
55     Serial.print(" zn[");Serial.print(i+1);Serial.print("] = ");
56     Serial.print(zn[i+1]);
57     Serial.print(" t[");Serial.print(i);Serial.print("] = ");
58     Serial.print(t[i]);
59     Serial.print(" ***");
60     Serial.println("");
61     f[i] = 0;
62   }
63 }
64 auxv[i] = vint[i];
65
66 if (i > 0){
67   if (lum[i - 1] < 3)
68     lum[i - 1] = 3;
69   if ((c[i] == 0) && ( lum[i - 1] > 3)) {
70     if (timp == 0) {
71       if (lum[i - 1] > 8)
72         lum[i - 1] = lum[i - 1] * 0.7;
73       else
74         lum[i - 1] --;
75     }
76     timp++;
77     timp = timp % 5;
78   }
79   else if (c[i] == 0) {
80   }
81   else if (c[i] == 1){
82     if (v[i-1] > 4){
83       lum[i - 1] = 255;
84     }

```

```
85         else if (v[i-1] > 1){
86             if (lum[i - 1] > 150)
87                 lum[i - 1] = lum[i - 1] - 2;
88             else
89                 lum[i - 1] = 150;
90         }
91         else if (v[i-1] > 0){
92             if (lum[i - 1] > 40)
93                 lum[i - 1] = lum[i - 1] - 2;
94             else
95                 lum[i - 1] = 40;
96         }
97     }
98     else if (c[i] == 2){
99         if ((v[i-1] > 4)&& (lum[i - 1] < 150)){
100             lum[i - 1] = 255;
101         }
102         else if ((v[i - 1] > 1) && (lum[i - 1] < 150)){
103             lum[i - 1] = 150;
104         }
105     }
106     analogWrite(4 + i, lum[i - 1]);
107 }
108 }
109 }
110 }
```

### A.3. TaskLightRead

```
1 void TaskLightRead( void *pvParameters __attribute__((unused)) )
2 {
3
4     for (;;)
5     {
6         int FotoRez = analogRead(A0);
7         if (FotoRez>80)
8             xSemaphoreGive( SemNOAPTE );
9         else {
10            Serial.println(FotoRez);
11            analogWrite (5, 0);
12            analogWrite (6, 0);
13            analogWrite (7, 0);
14            timp = 0;
15            b[0] = 0; b[1] = 0; b[2] = 0; b[3] = 0; b[4] = 0;
16            f[0] = 0; f[1] = 0; f[2] = 0; f[3] = 0; f[4] = 0;
17            bn[0] = 0; bn[1] = 0; bn[2] = 0; bn[3] = 0; bn[4] = 0;
18            c[0] = 0; c[1] = 0; c[2] = 0; c[3] = 0;
19            lum[0] = 0; lum[1] = 0; lum[2] = 0;
```

```
20         v[0] = 0; v[1] = 0; v[2] = 0; v[3] = 0;
21         vint[0] = 0; vint[1] = 0; vint[2] = 0; vint[3] = 0;
22         t[0] = 0; t[1] = 0; t[2] = 0; t[3] = 0;
23         aux[0] = 0; aux[1] = 0; aux[2] = 0; aux[3] = 0; aux[4] = 0;
24         auxv[0] = 0; auxv[1] = 0; auxv[2] = 0; auxv[3] = 0;
25     }
26 }
27 }
```

#### A.4. TaskLCD

```
1 void TaskLCD( void *pvParameters __attribute__((unused)) )
2 {
3     for (;;)
4     {
5         if (p == 0) {
6             lcd.begin(16, 2);
7             lcd.setBacklight(255);
8             lcd.home();
9             lcd.clear();
10            p = p + 1;
11        }
12        xSemaphoreTake(SemLCD, portMAX_DELAY);
13        xSemaphoreTake(SemSB, portMAX_DELAY);
14        lcd.clear();
15        for (int i = 0; i < 4; i ++){
16            if (i == 0)
17                lcd.setCursor(0, 0);
18            else if (i == 1)
19                lcd.setCursor(7, 0);
20            else if (i == 2)
21                lcd.setCursor(0, 1);
22            else if (i == 3)
23                lcd.setCursor(7, 1);
24            lcd.print(v[i]);
25        }
26    }
27    lcd.print(" km/h");
28    xSemaphoreGive(SemSB);
29 }
30 }
```

# Bibliografie

- [1] Aziera Abdullah și alții. „Smart Street Light Using Intensity Controller“. În: *2018 7th International Conference on Computer and Communication Engineering (ICCCE)*. 2018.
- [2] B. Krishna Chaitanya și alții. „Automation of street lights using Arduino & NI Lab VIEW“. În: *2015 IEEE UP Section Conference on Electrical Computer and Electronics (UPCON)*. 2015.
- [3] Monica Drăgoicea. *Proiectarea aplicațiilor în timp real*. Universitatea "Politehnica" București, 2022.
- [4] Monica Drăgoicea. *Sisteme în timp real*. Universitatea "Politehnica" București, 2023.
- [5] Marko Kuusik, Toivo Varjas și Argo Rosin. „Case study of smart city lighting system with motion detector and remote control“. În: *2016 IEEE International Energy Conference (ENERGYCON)*. 2016.
- [6] Richard J. Smythe. „Counting Events and Timing“. În: *Arduino in Science: Collecting, Displaying, and Manipulating Sensor Data*. Berkeley, CA: Apress, 2021, pag. 239–304.
- [7] Richard J. Smythe. „Variable Intensity and Power Control“. În: *Arduino in Science: Collecting, Displaying, and Manipulating Sensor Data*. Berkeley, CA: Apress, 2021, pag. 209–238.

## **Lista contribuțiilor personale**

<b>Sistem în timp real pentru controlul iluminatului public</b>		
<b>Durbală Cristian</b>		
<b>Prof. dr. ing. Monica Drăgoicea</b>		
	Activitate	Durată [zile]
1	Documentare	5
2	Testare componente hardware	0.5
3	Alegere și implementare soluție hardware	3
4	Implementare cod - aprindere felinare pe baza counter-ului de obiecte	1.5
5	Redactare lucrare - Introducere și elemente hardware	2
6	Implementare cod - realizare calcul viteza pentru 1 vehicul și aprinderea felinarelor în funcție de aceasta	0.5
7	Implementare cod - realizare calcul viteza pentru 2 vehicule	3
8	Redactare lucrare - restul lucrării	4