Procesarea semnalelor primite de la senzori pe microcontroler

Student: Radu-Rares Ciobanu

Structura Sistemelor de Calcul

Universitatea Tehnica din Cluj-Napoca

Octombrie 17, 2024

Contents

Introduction

1.1 Context

Acest proiect implică măsurarea luminii şi a culorii folosind doi senzori conectaţi la un microcontroler Arduino Uno R3. Senzorul de culoare va detecta variaţiile cromatice din mediu, iar senzorul de luminozitate va măsura intensitatea luminii. Deşi Arduino Uno R3 nu dispune de paralelizare nativă, proiectul va utiliza tehnici de multitasking, pentru a colecta datele de la ambii senzori într-un mod rapid şi coordonat.

Datele obţinute vor fi procesate pentru a realiza diverse calcule şi interpretări ale valorilor măsurate. Deoarece Arduino Uno R3 are resurse limitate de memorie, un obiectiv important este optimizarea codului pentru a utiliza cât mai puţină memorie. Scopul proiectului este de a demonstra capacitatea sistemului de a prelucra şi analiza datele în mod eficient, chiar şi în condiţii de resurse hardware restrânse.

1.2 Objectives

Proiectul îşi propune să realizeze măsurători precise ale luminii şi culorii folosind senzori conectaţi la un microcontroler Arduino Uno R3. Obiectivele specifice ale proiectului includ:

Lista 1.2.1: Obiectivele Proiectului

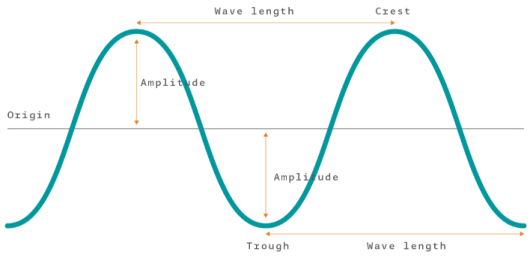
- **Analiza diferenţelor de măsurători:** Evaluarea variaţiilor de date obţinute de la senzori în urma schimbărilor bruşte ale condiţiilor de mediu.
- **Optimizarea memoriei utilizate:** Reducerea consumului de memorie al programului pentru a se adapta resurselor limitate ale Arduino Uno R3.
- **Realizarea multitaskingului eficient:** Implementarea unei gestionări alternative a citirilor de la senzori pentru a permite funcţionarea simultană a acestora, chiar şi în absenţa paralelizării native.

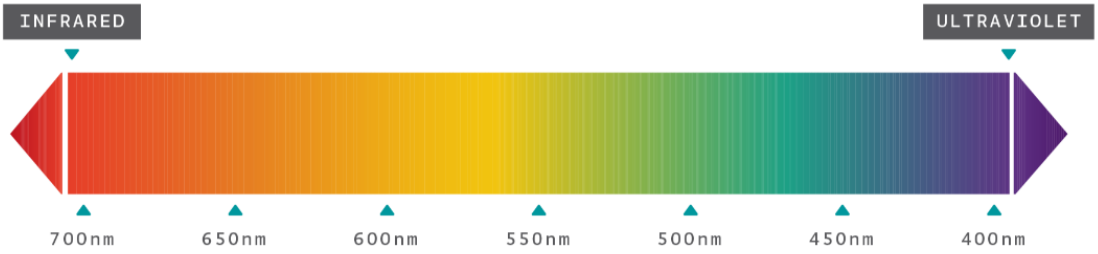
Studiu Bibliografic

2.1 Functionalitatea Senzorilor Alesi

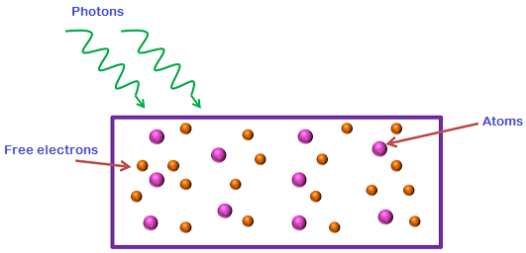
În cadrul proiectului, vor fi utilizaţi doi senzori: un senzor de culoare şi un senzor de luminozitate.

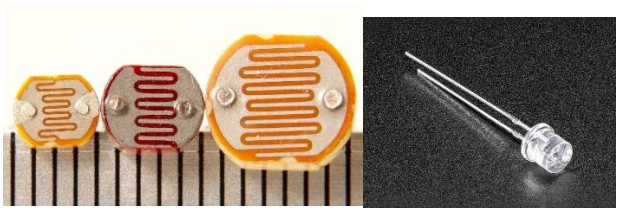
Senzorul de culoare detectează şi măsoară culorile din mediu prin captarea luminii reflectate de obiecte şi transformarea acesteia în semnale electrice, prin detectarea lungimii de undă (şi codificarea acesteia în RGB), ceea ce permite identificarea nuanţelor şi intensităţii culorilor.





Pe de altă parte senzorul de luminozitate utilizat în proiect este o fotot ranzistor, care este un dispozitiv ce converteşte lumina în semnale electrice. Acesta conţine materiale semiconductoare, cum ar fi siliciul, care absorb fotonii de lumină şi generează electroni. Când lumina loveşte fotodioda, aceasta provoacă eliberarea electronilor, creând un curent electric proporţional cu intensitatea luminii.





2.2 Flexibilitatea folosirii Arduino Uno

Arduino Uno oferă o flexibilitate remarcabilă pentru realizarea unor proiecte complexe, şi vine la pachet cu un microcontroler ***Atmega328P***, care oferă o frecvenţă de 16 MHz, 32 KB de memorie flash pentru stocarea codului, 2 KB de RAM (SRAM) pentru manipularea datelor şi multe alte caracteristici care pot fi de ajutor. Aceste specificaţii permit rularea codului în timp real şi gestionarea datelor senzorilor fără probleme. Microcontrolerul are 23 de pini de intrare-ieşire digitali şi analogici şi un convertor analog-digital (ADC) de 10 biţi, ideal pentru senzorii de lumină şi de culoare pe care îi vei folosi în proiect

2.3 Solutii pentru prelucrarea paralelă

Pentru a realiza rularea paralelă a proceselor pe Arduino Uno, se poate folosi o soluţie principală: utilizarea întreruperilor. Aceasta permite generarea de date de la senzori diferiţi după fiecare întrerupere, asigurând astfel o colectare eficientă a informaţiilor.

Prin configurarea timerelelor şi utilizarea întreruperilor, se pot obţine măsurători precise de la senzorii de lumină şi de culoare, fără a bloca execuţia programului principal. Această metodă maximizează utilizarea resurselor disponibile ale microcontrolerului şi îmbunătăţeşte performanţa proiectului.

2.4 Analiza şi procesarea masuratorilor

2.4.1 Detectarea Anomaliilor cu Z-Score

Pentru detectarea anomaliilor în intensităţile culorilor (Roşu, Verde, Albastru) şi în intensitatea luminii, se utilizează Z-Score. Acesta calculează cât de departe se află o valoare faţă de media anterioară, raportat la deviaţia standard. Formula Z-Score este:

$$Z - Score = \frac{X - \mu}{\sigma}$$

unde:

- X - este valoarea curentă,
- μ - este media valorilor anterioare,
- σ - este deviaţia standard a valorilor anterioare.

Dacă Z-Score-ul depăşeşte o valoare prestabilită (de obicei 2 sau 3), se consideră că valoarea este anormală şi semnalează o posibilă anomalie.

2.4.2 Detectarea Anomaliilor cu CUSUM

Pentru monitorizarea continuă a schimbărilor din intensitatea luminii şi culorii, se utilizează CUSUM (Cumulative Sum). Acesta calculează acumularea abaterilor faţă de media anterioară, semnalând schimbările semnificative din valorile măsurate. Formula CUSUM este:

$$CUSUM = max(0, CUSUM + (X - \mu - Prag))$$

unde:

- X este valoarea curentă,
- μ - este media valorilor anterioare,
- Pragul este valoarea de referinţă folosită pentru a detecta schimbările semnificative.

Dacă valoarea CUSUM depăşeşte un prag prestabilit, se semnalează o schimbare semnificativă în intensitatea luminii sau a culorii.

Analiza

3.1 Obiectivul proiectului

Acest proiect are ca scop colectarea şi prelucrarea datelor de la doi senzori - un senzor de culoare şi un senzor de lumină - utilizând platforma Arduino Uno şi microcontrolerul ATMega328P. Proiectul urmăreşte măsurarea intensităţii luminii şi analiza variaţiilor de culoare în funcţie de condiţiile de iluminare, date utile pentru aplicaţii precum controlul automat al iluminatului.

3.2. Cerinţe şi specificaţii

- **Cerinţe funcţionale**: Sistemul trebuie să capteze şi să stocheze măsurători de la senzori, să calculeze medii, variaţii şi alte rapoarte, şi să prezinte aceste date într-o formă intuitivă.
- **Cerinţe nefuncţionale**: Optimizarea memoriei şi a resurselor procesorului, având în vedere limitările hardware ale microcontrolerului ATMega328P, pentru a asigura viteza de procesare şi utilizarea eficientă a energie

3.4 Implementarea întreruperilor şi timerelor

Pentru a prelua măsurătorile în paralel, se folosesc timer-ele şi întreruperile. Timerul configurat în modul CTC (Clear Timer on Compare Match) generează o întrerupere la intervale precise, stabilite prin registrele TCCRn şi OCRn. Intreruperile

declanşează funcţia ISR care colectează valorile de la senzori, minimizând blocajele în execuţia codului principal.

3.6 Limitări

ATMega328P are limitări de memorie şi putere de calcul, astfel că sunt necesare optimizări pentru a gestiona datele în timp real. De asemenea, calculele complexe în virgulă mobilă pot consuma timp de procesare, afectând viteza de răspuns.

3.7 Soluții propuse pentru optimizare

Pentru a reduce impactul limitărilor, calculele sunt implementate într-un format pe întregi acolo unde este posibil, iar operaţiile în virgulă mobilă sunt minimizate. De asemenea, sunt utilizate tehnici de normalizare şi simplificare a calculelor pentru a asigura o execuţie rapidă şi eficientă.

Design

4.1 Modul de afisare a datelor

Pentru afisarea datelor se va realiza o interfata grafica unde vor fi afisate rezultatele calculelor astfel:

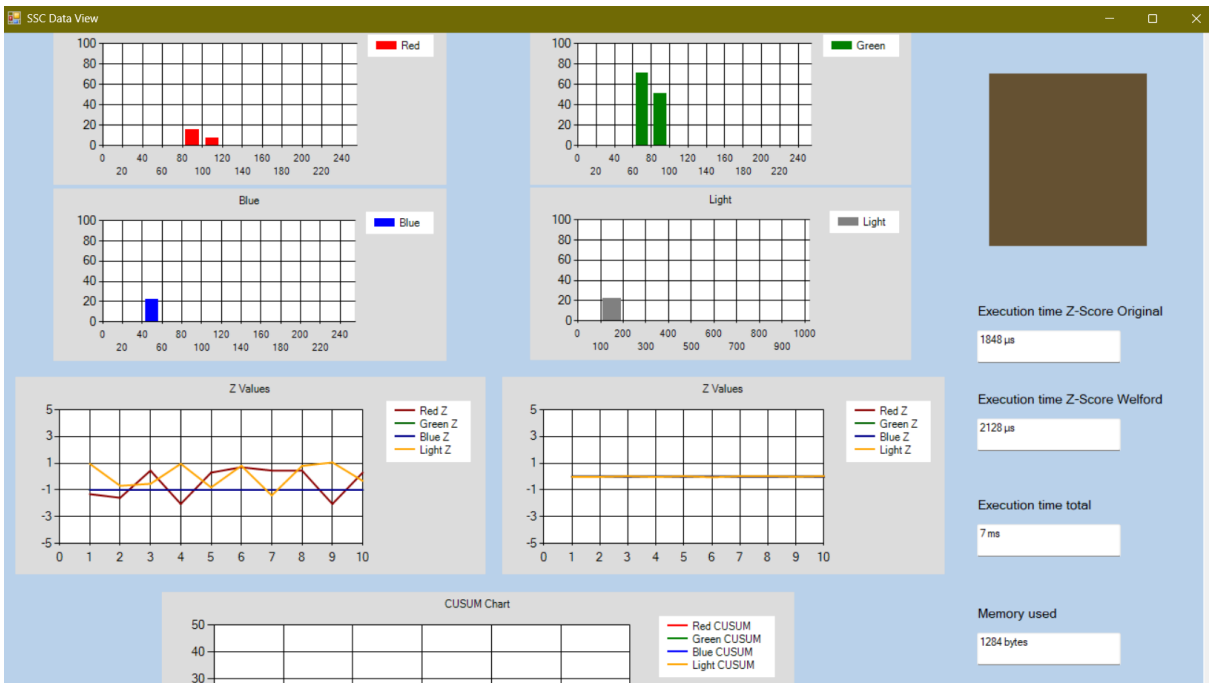


Fig 4.1

4.2 Diagrama de circuit

Imaginea (*Fig 4.2*) prezintă un circuit cu un Arduino UNO conectat la un senzor de culoare TCS34725 şi un fototranzistor. Pinul **Vin** al senzorului este conectat la **5V** pe Arduino pentru alimentare, iar **GND** la masă. Comunicarea între senzor şi Arduino se face prin conexiunile I2C: **SDA** la pinul **A4** şi **SCL** la pinul **A5**.

Fototranzistorul are colectorul conectat la **5V** printr-un rezistor de **10 kΩ**, iar emitorul la **GND**. Acesta detectează intensitatea luminii ambientale, iar senzorul trimite datele de culoare către Arduino pentru prelucrare.

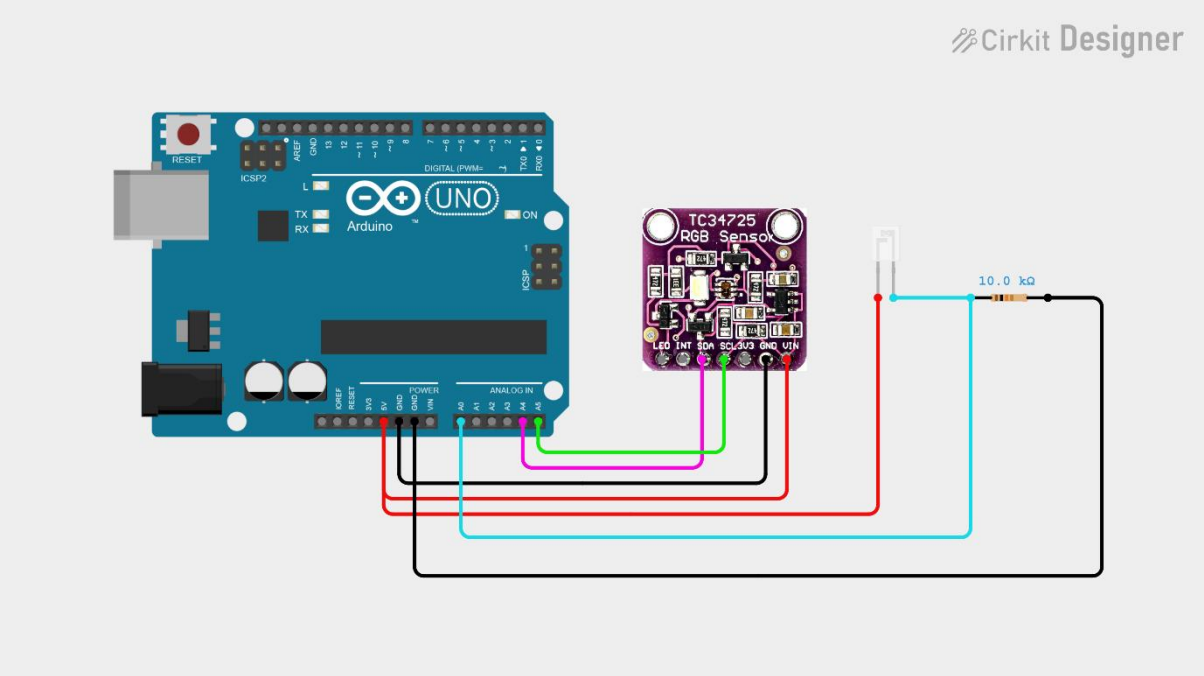


Fig 4.2

Implementare

Codul foloseşte un senzor de culoare TCS34725 şi un fototranzistor pentru a măsura valorile RGB şi intensitatea luminii. La început, sunt importate bibliotecile necesare pentru comunicaţia I2C şi pentru a controla senzorul TCS34725. Apoi, este definit pinul pentru fototranzistor şi alte variabile necesare, cum ar fi „bufferCount", care se foloseste pentru a indexa prin ultimele 10 valori preluate de la senzori.

```
#include <Wire.h>
#include "Adafruit_TCS34725.h"
#include <math.h>

#define BUFFER_SIZE 10
#define THRESHOLD 2.0

Adafruit_TCS34725 tcs = Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_50MS, TCS34725_GAIN_4X);

float redBuffer[BUFFER_SIZE];
float greenBuffer[BUFFER_SIZE];
float blueBuffer[BUFFER_SIZE];
float lightBuffer[BUFFER_SIZE];

int bufferIndex = 0;
int msSum = 0;
int wfdSum = 0;
int orgSum = 0;
int bufferCount = 0;
int ten = 10;
int defect = 10;
float redCUSUM = 0, greenCUSUM = 0, blueCUSUM = 0, lightCUSUM = 0;
float redMean = 0, greenMean = 0, blueMean = 0, lightMean = 0;
float redM2 = 0, greenM2 = 0, blueM2 = 0, lightM2 = 0;
float redStdDev = 0, greenStdDev = 0, blueStdDev = 0, lightStdDev = 0;
float redZScore, greenZScore, blueZScore, lightZScore, redZScoreWelford, greenZScoreWelford, blueZScoreWelford, lightZScoreWelford;
```

Fig 5.1

Urmează iniţializarea obiectului pentru senzorul de culoare, cu un timp de integrare de 50 ms şi un câştig de 4x pentru sensibilitate. Sunt definite variabile globale pentru a stoca mediile, deviaţiile standard şi Z-score-urile, pentru fiecare canal de culoare (roşu, verde, albastru) şi pentru luminozitate.

Codul include funcţii pentru a calcula media, deviaţia standard(***Fig 5.2***) si Z-score-ul, atat pentru implementarea clasica cat si pentru implementarea Welford(***Fig 5.3***). Aceste funcţii sunt folosite pentru a analiza valorile de culoare şi lumină detectate de senzor şi fototranzistor.

```
61  void updateBufferMeanAndVariance(float newValue, float &mean, float &stdDev, float buffer[], int count) {
62    float sum = 0;
63    for (int i = 0; i < count; i++) {
64      sum += buffer[i];
65    }
66    mean = sum / count;
67
68    float variance = 0;
69    for (int i = 0; i < count; i++) {
70      variance += pow(buffer[i] - mean, 2);
71    }
72    stdDev = sqrt(variance / count);
73  }
74
75  void updateMeanAndVariance(float newValue, float &mean, float &M2, int count) {
76    float delta = newValue - mean;
77    mean += delta / count;
78    M2 += delta * (newValue - mean);
79  }
80
81  float calculateCUSUM(float value, float mean, float threshold, float& cusum) {
82    cusum = max(0.0f, cusum + (value - mean - threshold));
83    return cusum;
84  }
85
86  float calculateZScore(float value, float mean, float stdDev) {
87    return (value - mean) / stdDev;
88  }
```

Fig 5.2

```
float calculateZScore(float value, float mean, float stdDev) {
  return (value - mean) / stdDev;
}
```

Fig 5.3

Acest cod (***Fig 5.7***)configurează şi utilizează ADC-ul (Analog-to-Digital Converter) şi Timer1 al microcontrollerului **ATmega328P**. Funcţia read_adc permite citirea unei valori analogice de pe un canal specific, configurând registrele **ADMUX** (***Fig 5.4***) şi **ADCSRA** (***Fig 5.5***) pentru selectarea canalului şi pornirea conversiei.

23.9.1 ADMUX – ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Fig 5.4

23.9.2 ADCSRA – ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Fig 5.5

Timer1 este configurat pentru a genera o întrerupere periodică folosind **OCR1A** în mod **CTC** (Clear Timer on Compare Match), iar în rutina de întrerupere ISR asociată, valoarea citită de la ADC este salvată într-un buffer. Codul setează şi prescalerul ADC-ului pentru o frecvenţă corectă şi activează întreruperile pentru a citi automat valorile la intervale precise.

14.9.2 TCCR0B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Fig 5.6

```
81  uint16_t read_adc(uint8_t channel) {
82    ADMUX &= 0xE0;
83    ADMUX |= channel&0x07;
84    ADCSRB = channel&(1<<3);
85    ADCSRA |= (1<<ADSC);
86    while(ADCSRA & (1<<ADSC));
87    return ADCW;
88  }
89
90  ISR(TIMER1_COMPA_vect) {
91    lightBuffer[bufferIndex] = read_adc(0);
92  }
93
94  void setup() {
95    TCCR1A = 0;
96    TCCR1B = 0;
97    OCR1A = 7812;
98    TCCR1B |= (1 << WGM12);
99    TCCR1B |= (1 << CS12) | (1 << CS10);
100   TIMSK1 |= (1 << OCIE1A);
101
102   ADMUX = (1 << REFS0);
103   ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);
104
105   sei();
106   Serial.begin(9600);
107 }
```

Fig 5.7

Această porţiune de cod este folosită pentru a determina memoria SRAM folosită. Aşa putem afla şi memoria rămasă.(***Fig 5.8***).

```
109 extern int __heap_start, *__brkval;
110
111 int usedStackSpace() {
112   int stackPointer;
113   return (int)&stackPointer - (__brkval == 0 ? (int)&__heap_start : (int)__brkval);
114 }
```

Fig 5.8

Prin portul serial, se pot citi date de la microcontroler şi transmite către interfaţa grafică, unde acestea pot fi preluate şi afişate. Aceasta permite o comunicare eficientă între microcontroler şi aplicaţia care vizualizează informaţiile. (***Fig 5.9***).

```
174     Serial.print(redValue, 2); Serial.print(",");
175     Serial.print(greenValue, 2); Serial.print(",");
176     Serial.print(blueValue, 2); Serial.print(",");
177     Serial.print(lightValue, 2); Serial.print(",");
178     Serial.print(redZScore, 2); Serial.print(",");
179     Serial.print(greenZScore, 2); Serial.print(",");
180     Serial.print(blueZScore, 2); Serial.print(",");
181     Serial.print(lightZScore, 2); Serial.print(",");
182     Serial.print(redZScoreWelford, 2); Serial.print(",");
183     Serial.print(greenZScoreWelford, 2); Serial.print(",");
184     Serial.print(blueZScoreWelford, 2); Serial.print(",");
185     Serial.print(lightZScoreWelford, 2); Serial.print(",");
186     Serial.print(redCUSUM, 2); Serial.print(",");
187     Serial.print(greenCUSUM, 2); Serial.print(",");
188     Serial.print(blueCUSUM, 2); Serial.print(",");
189     Serial.println(lightCUSUM, 2);
190     ten--;
191     if (ten == 0){
192       Serial.print(orgSum); Serial.print(",");
193       Serial.print(wfdSum); Serial.print(",");
194       Serial.print(msSum); Serial.print(",");
195       Serial.println(usedStackSpace());
196       msSum = 0;
197       wfdSum = 0;
198       orgSum = 0;
199       ten = 10;
200     }
```

Fig 5.9

Testare si validare

Algoritmul Welford diferă de implementarea originală prin faptul că calculează media şi varianţa incremental, actualizând aceste valori pe măsură ce fiecare nou punct de date este introdus. Această abordare permite obţinerea unui rezultat mai stabil în cazul fluxurilor continue de date, fără a fi necesară păstrarea întregii serii de date. În schimb, implementarea originală foloseşte un buffer pentru a stoca valorile şi calculează media şi varianţa pe întreaga serie de date într-un singur pas. Deşi Welford introduce un cost suplimentar de calcul pe fiecare actualizare, ambele metode furnizează rezultate corecte şi consistente.(***Fig 6.1, Fig 6.2***)

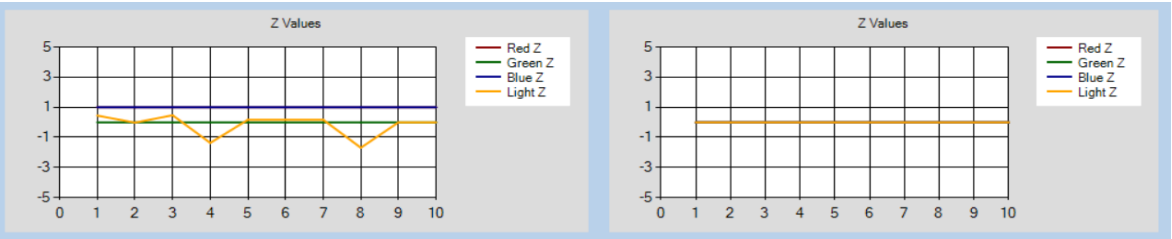


Fig 6.1

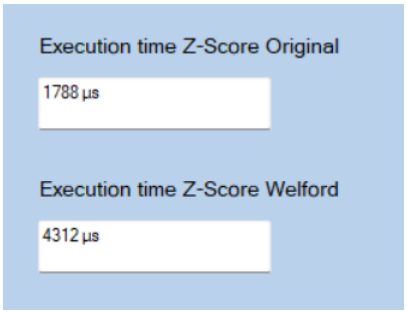


Fig 6.2

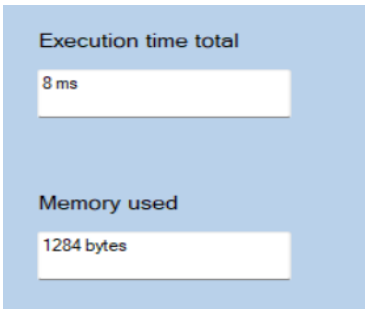


Fig 6.3

Algoritmii implementați funcționează perfect, iar testele efectuate au confirmat că histograma generată reflectă corect distribuția valorilor de culoare și lumină. Performanța sistemului a fost evaluată și utilizarea spațiului de memorie a fost menținută sub 50%, ceea ce asigură o eficiență bună în cadrul resurselor disponibile. Algoritmii sunt rapizi și eficienți, iar implementarea acestora îndeplinește cerințele proiectului fără probleme majore. (***Fig 6.3, Fig 6.4***)

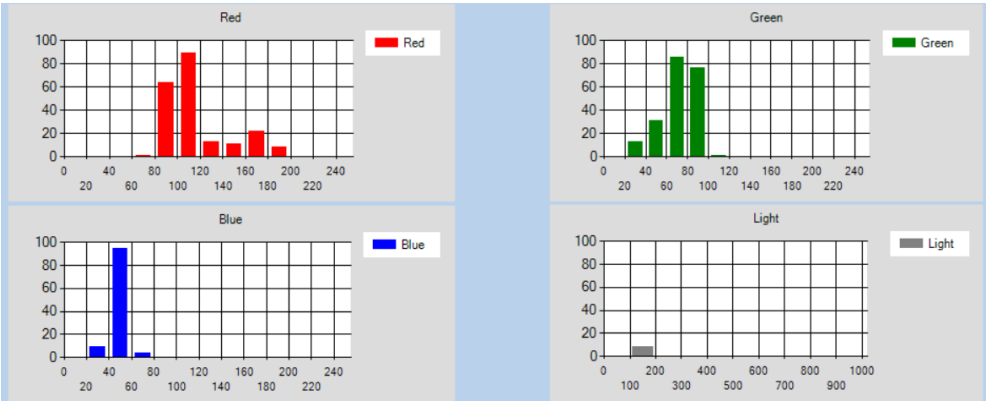


Fig 6.4

Concluzii

În concluzie, proiectul a fost o experiență foarte interesantă și satisfăcătoare, deoarece am reușit să implementez și să compar două abordări diferite pentru procesarea datelor de la senzorul de culoare și iluminare. Ambele metode, fie că vorbim despre abordarea buffer-based, fie despre algoritmul Welford, au demonstrat rezultate corecte și performanță solidă. M-a încântat să văd cum algoritmii au reușit să răspundă cerințelor, iar histogramă generată a fost exactă, ceea ce a validat întregul proces. În plus, utilizarea eficientă a resurselor și performanța optimă mi-au oferit încredere în stabilitatea și fiabilitatea implementării. Această experiență mi-a oferit ocazia să învăț și să apreciez detaliile tehnice ale procesării semnalului și ale optimizării algoritmicelor, iar rezultatele obținute mă fac să fiu foarte mulțumit de cum a evoluat proiectul.

Bibliografie

[1] The Arduino Team, https://studentkit.arduino.cc/resources, 2021

[2] Anatoli Arkhipenko, https://docs.arduino.cc/libraries/taskscheduler/, 2022

[3] Muskan Shaik, https://www.physics-and-radio-electronics.com/electronic-devices-and-circuits/passive-components/resistors/photoresistor.html, 2014

[4] Rui Santos și Sara Santos, https://randomnerdtutorials.com/arduino-color-sensor-tcs230-tcs3200/, 2017

[5] Banzi Massimo, "Getting Started with Arduino: The Open Source Electronics Prototyping Platform", 2009

[6] Răzvan Itu, "Proiectare cu microprocesoare", 2018

[7] Atmel Corporation, "ATmega328P Datasheet", 2015