Procesarea semnalelor primite de la senzori pe microcontroler

Student: Simina Dan-Marius

Proiect Structura Sistemelor de Calcul

Universitatea Tehnică din Cluj-Napoca

Cuprins

1. Introducere

1.1 Context

Scopul acestui proiect este eficientizarea implementărilor de procesare a datelor de la senzori pe microcontroler. Microcontrolerele sunt folosite în produse și dispozitive controlate automat, cum ar fi sistemele de control al motoarelor automobilelor, dispozitivele medicale implantabile, aparatele electrocasnice, uneltele electrice, jucării și alte sisteme încorporate. Prin reducerea dimensiunii și costului în comparație cu un design care utilizează un microprocesor separat, memorie și dispozitive de intrare/ieșire, microcontrolerele fac practic posibil controlul digital al unui număr mai mare de dispozitive și procese. În ceea ce privește microcontrolerele senzorii joacă un rol crucial, permițând microcontrolerelor să colecteze informații despre mediu și să răspundă în consecință.

Implementările eficiente ale procesării datelor de la senzori pe microcontrolere sunt importante din mai multe motive, două dintre ele fiind:

Constrângeri de Resurse: Microcontrolerele au adesea putere de procesare, memorie și sursă de energie limitate. Procesarea eficientă a datelor asigură utilizarea eficace a acestor resurse, maximizând performanța fără a supraîncărca sistemul.

Procesarea online: Multe aplicații, cum ar fi robotică sau sisteme auto, necesită procesare a datelor în timp real. Algoritmii eficienți pot ajuta la asigurarea unor răspunsuri rapide la intrările senzorilor, îmbunătățind fiabilitatea și performanța sistemului.

1.2 Obiective

1. Procesarea eficientă a datelor de la senzorii accelerometru și giroscop (MPU 6050) și de la senzorul de distanță (HC-SR04).
2. Transmiterea datelor procesate de la microcontroler către o aplicație de afișare, prin intermediul interfeței seriale.
3. Realizarea unei implementări inițiale orientată exclusiv pe funcționalitate.
4. Optimizarea implementării din punctul de vedere al timpului de execuție și al dimensiunii programului.
5. Compararea performanțelor dintre soluția inițială și cea optimizată, prezentând diferențele obținute.

2. Studiu bibliografic

2.1 Specificații Arduino UNO

Arduino UNO este o placă de dezvoltare populară, ideală pentru începători și profesioniști, bazată pe microcontrolerul ATmega328P. Designul său simplu și versatil o face potrivită pentru o gamă largă de proiecte electronice.

Specificații tehnice:

- Este un microcontroler bazat pe ATmega328P.
- Tensiunea de funcționare a Arduino-ului este de 5V.
- Tensiunea de intrare recomandată variază între 7V și 12V.
- Tensiunea de intrare (limita) este între 6V și 20V.
- Pinii de intrare și ieșire digitali - 14.
- Pinii de intrare și ieșire digitali (PWM) - 6.
- Pinii de intrare analogici sunt 6.
- Curentul continuu pentru fiecare pin I/O este de 20 mA.
- Curentul continuu utilizat pentru pinul de 3.3V este de 50 mA.
- Memoria Flash - 32 KB, iar 0.5 KB din memorie este utilizată de bootloader.
- SRAM este de 2 KB.
- EEPROM este de 1 KB.
- Viteza CLK este de 16 MHz.
- LED integrat.
- Dimensiunile Arduino-ului sunt 68.6 mm x 53.4 mm.

Nu există suport pentru multiprocesare sau multithreading pe Arduino.

2.2 Cum se poate realiza multitasking cu Arduino

1) **Implementarea ca o mașină cu stări finite:**

   O mașină de stare finită este un model de calcul care include stările în care poate fi o mașină și modul în care mașina trece de la o stare la alta; mașina poate fi doar într-o singură stare la un moment dat.

   Totodată ar trebui îndeplinite urmatoarele condiții:

   1. Menținerea timpul de execuție al tuturor funcțiilor foarte scurt.
   2. Nu folosim delay().
   3. Nu se așteptă pentru a primi ceva.

2) Folosirea librăriei Prothreads:

Protothreads este o bibliotecă pur C. Cu Protothreads poți „simula” multithreading-ul pentru sisteme bazate pe evenimente, astfel că este destul de utilă pentru programele Arduino mai complexe.

3) Multitasking cu millis():

Funcția millis returnează numărul de milisecunde de la momentul în care placa Arduino a început să ruleze programul curent, iar astfel o putem folosi pentru a gestiona mai multe sarcini fără a bloca programul. Aceasta permițând să se execute diferite funcții în funcție de timpul scurs.

4) Utilizarea întreruperilor externe:

Unii pini de pe Arduino suportă întreruperi hardware. Practic se creează o funcție care este declanșată de un buton sau alt actuator de pe un pin hardware. Când întreruperea este declanșată, programul va fi întrerupt, iar funcția va fi executată. Odată ce funcția s-a terminat, programul continuă de unde a fost întrerupt. Desigur, funcția ar trebui să fie foarte rapidă, pentru a nu opri execuția principală "thread"-ului prea mult timp.

5) Folosirea librăriei TaskScheduler

Suportă: execuția periodică a sarcinilor (cu perioadă de execuție dinamică în milisecunde sau microsecunde – frecvența execuției), număr de iterații (număr limitat sau infinit de iterații), execuția sarcinilor într-o secvență prestabilită, modificarea dinamică a parametrilor de execuție ai sarcinilor (frecvență, număr de iterații, metode callback), economisirea energiei prin intrarea în modul de repaus IDLE când sarcinile nu sunt programate să ruleze, invocarea sarcinilor bazată pe evenimente prin intermediul obiectului Status Request, ID-uri de sarcini și Puncte de Control pentru gestionarea erorilor și temporizatorul watchdog, pointerul Local Task Storage (care permite utilizarea aceluiași cod callback pentru mai multe sarcini), prioritizarea stratificată a sarcinilor, funcții std::functions (unde sunt suportate), timeout global al sarcinilor, legarea statică și dinamică a metodelor callback.

6) Folosirea librariei Timer One

Această bibliotecă permite utilizarea funcțiilor Timer pe plăcile Arduino. Cu ajutorul acestei biblioteci, se pot adăuga și gestiona evenimente bazate pe funcții de timer. Timer-ul este configurat cu un anumit număr de microsecunde, iar de fiecare dată când acest număr este atins, un contor este incrementat și, dacă este setată, o întrerupere este activată. Acest lucru înseamnă că poți programa întreruperi recurente, configurând frecvența acestora.

2.3 Cum funcționează senzorul de accelerometru si giroscop MPU-6050

Senzorul MPU6050 are integrat un accelerometru pe 3 axe și un giroscop pe 3 axe pe un singur cip.

Giroscopul măsoară viteza de rotație sau rata de schimbare a poziției unghiulare în timp, pe axele X, Y și Z. Acesta utilizează tehnologia MEMS(microfabricated systems comprising both electrical and mechanical components) și efectul Coriolis(forța Coriolis este o forță aparentă, de inerție, care acționează asupra unui corp când acesta este situat într-un sistem de referință aflat în mișcare de rotație, iar din punct de vedere fizic, ea este o urmare a conservării momentului cinetic a mișcării rotative) pentru măsurare. Ieșirile giroscopului sunt exprimate în grade pe secundă, așa că, pentru a obține poziția unghiulară, trebuie doar să integrăm viteza unghiulară.

2.4 Cum funcționează senzorul de distanță HC-SR04

Senzorul HC-SR04 emite un ultrasunet la 40.000 Hz care călătorește prin aer, iar dacă există un obiect sau un obstacol pe calea sa, acesta va reveni la modul. Luând în considerare timpul de călătorie și viteza sunetului se poate calcula distanța.

2.5 Filtrarea semnalelor primite de la senzori

Măsurările din lumea reală conțin adesea zgomot. În termeni generali, zgomotul este doar partea semnalului pe care nu o dorești. De exemplu, vibrațiile de la motor adaugă zgomot dacă măsori accelerația unui kart. Filtrarea este o metodă de a elimina o parte din semnalul nedorit pentru a obține un rezultat mai neted.

Una dintre cele mai simple modalități de a filtra datele zgomotoase este prin calcularea mediei.

Calculul mediei funcționează prin adunarea unui număr de măsurători și împărțirea totalului la numărul de măsurători adunate. Cu cât incluzi mai multe măsurători în medie, cu atât mai mult zgomot va fi eliminat. Totuși, există o returnare în scădere: media a 101 măsurători nu va fi cu mult mai puțin zgomotoasă decât media a 100 măsurători.

2.6 Metode de optimizare a programului

Optimizarea codului înseamnă îmbunătățirea codului pentru a produce un program eficient. Un cod bine optimizat trebuie să fie concis, să ocupe mai puțină memorie, să aibă un timp de execuție mai rapid, un randament mai mare, un consum redus de energie și, cel mai important, rezultatul trebuie să fie același ca și rezultatul codului neoptimizat.

Tehnici pentru optimizarea codului:

1) Eliminarea codului inutil

Codul inutil se referă la orice variabilă, funcție sau bibliotecă neutilizată pe care ai putea să o fi inclus în schița ta. Aceste „coduri moarte” provin adesea din etapa inițială a dezvoltării.

2) Folosirea tipurilor de date mai mici

Un tip de date indică ce poate conține o variabilă în programare. O variabilă este un nume pe care îl atribui unei porțiuni de date din memorie. Există câteva tipuri de date disponibile în programarea Arduino.

Arduino efectuează cele mai rapide operațiuni pe tipurile de date întregi (cum ar fi int și unsigned int), dar mai ales pe tipurile de date de dimensiune mică, precum byte și char. Aceasta se datorează faptului că procesorul din majoritatea plăcilor Arduino, cum ar fi ATmega328P (utilizat în Arduino Uno), are o arhitectură de 8 biți, ceea ce înseamnă că este optimizat pentru a lucra cu date de 8 biți (cum ar fi byte sau char).

3) Folosirea funcțiilor

Funcțiile sunt secțiuni denumite ale unui program care îndeplinesc o sarcină specifică. Ele previn repetarea codului și economisesc memorie, deoarece CPU-ul încarcă codul din memorie doar atunci când funcția este apelată.

4) Folosirea variabilelor locale în loc de variabile globale

Variabilele globale sunt declarate înainte de funcția void setup() și pot fi apelate în întreaga schiță, în timp ce variabilele locale sunt disponibile doar în funcția lor părinte. Variabilele globale sunt încărcate de fiecare dată când programul rulează pe Arduino-ul tău, ceea ce înseamnă că ocupă resurse și contribuie la timpul de execuție, chiar dacă bucla principală nu le folosește. În schimb, variabilele locale sunt încărcate doar când funcția lor părinte este apelată.

5) F() Strings

F() Strings oferă o altă modalitate de a afișa textul în monitorul serial sau pe un ecran. Imprimarea tradițională a șirurilor consumă o cantitate semnificativă de RAM. O modalitate de a remedia acest lucru este să salvăm șirurile în memoria flash. Pentru a face acest lucru, adăugăm F() la începutul variabilei șir.

6) **Mutarea constantelor în PROGMEM**

PROGMEM este un cuvânt cheie în Arduino IDE care stochează datele în memoria programului sau în memoria flash, în loc de RAM. Este recomandat să stochezi date care nu se schimbă, precum constantele, în memoria flash, deoarece aceasta are o capacitate mai mare. Totuși, trebuie menționat că memoria flash este mai lentă la încărcare.

7) **Direct Port Manipulation**

Porturile sunt coduri care reprezintă registrele dintr-un microcontroller. Acestea îți permit să controlezi și să citești direct starea pinilor.

8) **Eliminarea bootloader-ului**

În final, poți elimina bootloader-ul Arduino pentru a elibera spațiu. Microcontrolerele sunt de obicei programate printr-un programator, cu excepția cazului în care ai un firmware în microcontroler care permite instalarea de firmware nou fără a folosi un programator extern. Acest firmware se numește bootloader, conform site-ului oficial Arduino.

Din păcate, acest software ocupă aproximativ 2000 de octeți din memoria flash. Ia în considerare eliminarea bootloader-ului și programarea cu un programator extern sau prin ISP dacă ești limitat de memorie.

2.7 Algoritmi pentru procesarea datelor

Pentru procesarea datelor se pot aplica algoritmi cum ar fii:

1) **Cusum:**

În controlul statistic al calității, CUSUM (sau diagrama de control a sumei cumulative) este o tehnică de analiză secvențială dezvoltată de E. S. Page de la Universitatea din Cambridge. Este utilizată în mod obișnuit pentru monitorizarea detectării schimbărilor.

2) **Z-Score:**

În statistică, scorul standard reprezintă numărul de abateri standard prin care valoarea unui scor brut (adică o valoare observată sau un punct de date) se află deasupra sau dedesubtul valorii medii a ceea ce este observat sau măsurat. Scorurile brute care sunt deasupra mediei au scoruri standard pozitive, în timp ce cele care sunt sub medie au scoruri standard negative.

2.8 Implementare aplicație pentru preluarea și afișarea datelor

Pentru implementarea aplicație care va afișa datele preluate de la microcontroler se va folosi Windows Forms. Windows Forms este un framework UI pentru crearea de aplicații desktop pe Windows. Acesta oferă una dintre cele mai productive metode de a crea aplicații desktop, bazându-se pe designerul vizual furnizat în Visual Studio. Funcționalitatea, precum plasarea prin drag-and-drop a controalelor vizuale, face ușor procesul de construire a aplicațiilor desktop.

Cu Windows Forms, dezvolți aplicații grafic bogate, ușor de distribuit, actualizat și care funcționează atât offline, cât și conectate la internet. Aplicațiile Windows Forms pot accesa hardware-ul local și sistemul de fișiere al calculatorului pe care rulează.

Comunicarea serială poate fi stabilită și în procesul de proiectare a aplicației, făcând astfel mai ușoară și rapidă interacțiunea cu Arduino printr-un PORT COM specificat.

3. Analiză

3.1 Propunere de Proiect

Implementarea unui program pe Arduino UNO care procesează datele de la senzorul de accelerometru și giroscop MPU-6050 și senzorul pentru măsurarea distanței HC-SR04.

Din punct de vedere al procesării datelor, pentru datele primite de la MPU-6050 se va calcula Z-Score care este o măsură statistică ce descrie relația unei valori față de media unui grup de valori, iar datele de la HC-SR04 vor fi trecute printr-un filtru de medie mobilă. După aceea se vor fi trimise unei aplicații pentru afișare.

Din punct de vedere al implementării, mai întâi se va realiza o implementare bazată doar pe funcționalitate, iar mai apoi se va încerca optimizarea codului și se vor compara soluțiile din punct de vedere al performanței.

3.2 Analiza Proiectului

3.2.1 MPU-6050

Modulul MPU6050 este un senzor de mișcare pe 6 axe, care integrează un giroscop și un accelerometru, fiecare pe 3 axe, oferind astfel date precise despre rotație și accelerație. Acesta comunică cu microcontrolerele prin interfața I2C.

Componentele principale ale MPU6050:

1. Giroscop pe 3 axe (tehnologie MEMS):
   - Detectează viteza de rotație în jurul axelor X, Y și Z.
   - Se bazează pe efectul Coriolis pentru a sesiza mișcările de rotație; vibrațiile rezultate sunt transformate în semnale electrice.
   - Semnalele sunt amplificate și convertite în digital cu ajutorul unui ADC pe 16 biți, permițând măsurarea vitezei unghiulare în grade pe secundă.
   - Domeniile de măsurare configurabile sunt de ±250, ±500, ±1000, și ±2000 grade/s.

2. Accelerometru pe 3 axe (tehnologie MEMS):
   - Măsoară accelerația liniară pe axele X, Y și Z, oferind informații despre înclinare și mișcare.
   - Funcționează pe baza unei mase mobile care generează un semnal electric proporțional cu accelerația pe fiecare axă.
   - Utilizează un ADC pe 16 biți pentru digitalizarea datelor, cu domenii de măsurare configurabile de ±2g, ±4g, ±8g, și ±16g.
   - În absența mișcării și plasat pe o suprafață orizontală, senzorul indică 0g pe X și Y și +1g pe Z.

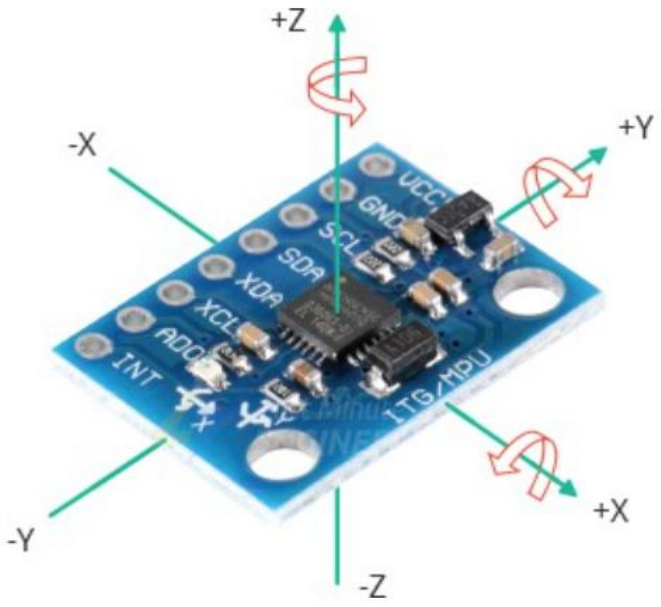


Figura 3.2.1.1: Axe MPU-6050

Dacă accelerometrul are un interval de măsură de +/- 2g cu un factor de sensibilitate de 16.384 LSB (Count)/g, iar giroscopul are un interval de măsură de +/- 250 °/s cu un factor de sensibilitate de 131 LSB (Count)/°/s atunci:

Valorile accelerometrului în g (forța g):

Accelerația pe axa X = (Datele brute ale accelerometrului pe axa X / 16384) g.

Accelerația pe axa Y = (Datele brute ale accelerometrului pe axa Y / 16384) g.

Accelerația pe axa Z = (Datele brute ale accelerometrului pe axa Z / 16384) g.

Valorile giroscopului în °/s (grade pe secundă):

Viteza unghiulară pe axa X = (Datele brute ale giroscopului pe axa X / 131) °/s.

Viteza unghiulară pe axa Y = (Datele brute ale giroscopului pe axa Y / 131) °/s.

Viteza unghiulară pe axa Z = (Datele brute ale giroscopului pe axa Z / 131) °/s.

Pentru calcularea unghiurilor în grade folosind datele de la accelerometru se folosește formula:

$$\theta = atan2(A_x, A_y) \cdot \frac{180}{\pi}$$

Unde:

- θ este unghiul în grade,
- A_x este accelerația pe direcția X (m/s²),
- $A_y$este accelerația pe direcția Y (m/s²).

Pentru a calcula unghiurilor în grade folosind datele de la giroscop se folosește formula:

$$\theta = \theta_0 + \omega \cdot t$$

Unde:

- θ este unghiul final,
- θ_0 este unghiul inițial,
- ω este viteza unghiulară,
- t este timpul.

Pentru a avea o estimare precisă și stabilă a unghiului se poate folosi un filtru complementar, care poate fi considerat ca o combinație a două filtre diferite: un filtru trece-sus pentru giroscop și un filtru trece-jos pentru accelerometru. Primul permite trecerea doar a valorilor deasupra unui anumit prag, spre deosebire de filtrul trece-jos, care permite trecerea doar a valorilor de sub acest prag.

Accelerometrul oferă un indicator bun al orientării în condiții statice, iar giroscopul oferă un indicator bun al înclinării în condiții dinamice. Formula rezultată din combinarea celor două filtre este:

$$unghi = (1 - \alpha) * (unghi + unghi\ giroscop * dt) + \alpha * unghi\ accelerometru$$

3.2.2 HC-SR04

Senzorul ultrasonic HC-SR04 funcționează prin emiterea și recepționarea ultrasunetelor pentru a măsura distanța până la un obiect, folosind un principiu similar cu sistemele SONAR și RADAR. Modulul are un emițător de ultrasunete, un receptor și un circuit de control.

Cum funcționează HC-SR04:

1. Inițierea măsurătorii: Se trimite un impuls de trigger către modul, ceea ce determină emițătorul să genereze 8 pulsuri de ultrasunete la 40 kHz.

2. Măsurarea ecoului: După emitere, pinul de echo devine activ (HIGH) și rămâne astfel până când receptorul detectează ecoul reflectat de obiect.
3. Calculul distanței: Durata cât pinul de echo este activ reprezintă timpul de călătorie al ultrasunetelor dus-întors. Folosind acest timp și viteza sunetului, se poate calcula distanța până la obiect.

HC-SR04 măsoară distanțe între 2 cm și 400 cm.

Pentru a măsura distanța utilizând senzorul ultrasonic HC-SR04, se folosește formula:

$$Distanta = \frac{Timp \times Viteza\ Sunetului}{2}$$

3.2.3 Algoritmul Z-Score

Z-Score este o măsură statistică ce descrie relația unei valori față de media unui grup de valori. Z-Score este exprimat în unități de abatere standard față de medie.

Scorul Z este o măsură statistică ce cuantifică distanța dintre un punct de date și media unui set de date. Acesta este exprimat în unități de abatere standard și indică de câte abateri standard se află un punct de date față de media distribuției.

Dacă un Z-Score este 0, înseamnă că punctul de date este identic cu media. Un Z-Score de 1,0 ar indica o valoare care se află la o abatere standard deasupra mediei. Z-Score pot fi pozitive sau negative: o valoare pozitivă arată că scorul este deasupra mediei, iar o valoare negativă indică faptul că este sub media.

Pentru a calcula Z-Score pentru un vector de date de la senzori, se folosește următoarea formulă pentru fiecare element x din vector:

$$z = \frac{x - \mu}{\sigma}$$

Unde:

- x este valoarea individuală din vector,
- μ este media vectorului,
- σ este abaterea standard a vectorului.

Pași pentru calculul Z-Score pentru fiecare valoare dintr-un vector de date:

1. **Calcularea mediei (μ)** valorilor din vector:

$$\mu = \frac{\sum_{i=1}^{n} x_i}{n}$$

   unde n este numărul total de valori din vector.

2. **Calcularea abaterii standard (σ)**:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \mu)^2}{n}}$$

3. **Calculerea Z-Score** pentru fiecare valoare x_i folosind formula:

$$z_i = \frac{x_i - \mu}{\sigma}$$

3.2.4 Medie mobilă

Mediea mobilă calculează media unui set de date pentru o perioadă specificată.

O medie mobilă, cunoscută și sub denumirea de medie curentă sau medie cumulativă, este un calcul al mediei unui set de date luat pe o „fereastră glisantă" de dimensiune specificată. Această fereastră glisantă se deplasează de-a lungul punctelor de date, astfel încât, la fiecare pas, se calculează media celor mai recente puncte de date din cadrul ferestrei.

3.2.5 Algoritmul online al lui Welford

Este adesea util să poți calcula varianța într-o singură trecere, inspectând fiecare valoare x_i doar o singură dată; de exemplu, atunci când datele sunt colectate fără suficient spațiu de stocare pentru a păstra toate valorile sau atunci când costurile accesului la memorie domină costurile calculului. Pentru un astfel de algoritm online, este necesară o relație de recurență între cantități din care statisticile necesare pot fi calculate într-un mod numeric stabil.

Următoarele formule pot fi utilizate pentru a actualiza media și varianța (estimată) a unei secvențe, atunci când se adaugă un nou element x_n.

Media pentru primele n mostre:

$$\overline{x_n} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Varianța eșantionului părtinitoare:

$$\sigma_n^2 = \frac{1}{n}\sum_{i=1}^{n} (x_i - \overline{x_n})^2$$

Varianța eșantionului nepărtinitoare:

$$s_n^2 = \frac{1}{n-1}\sum_{i=1}^{n} (x_i - \overline{x_n})^2$$

Calcularea folosind algoritmul lui Welford:

$$M_{2,n} = M_{2,n-1} + (x_n - \overline{x_{n-1}})(x_n - \overline{x_n})$$

$$\sigma_n^2 = \frac{M_{2,n}}{n}$$

$$s_n^2 = \frac{M_{2,n}}{n-1}$$

3.2.6 Aproximarea timpului de execuție

Arduino Uno se bazează pe microcontrolerul ATmega328P, care funcționează la o frecvență de ceas de 16 MHz. Această frecvență de ceas determină cât de repede poate microcontrolerul să execute instrucțiuni.

Cicluri de Ceas:

Microcontrolerul execută instrucțiuni în cicluri de ceas. Numărul de cicluri de ceas necesare pentru a executa o instrucțiune variază în funcție de instrucțiunea în sine.

ATmega328P de pe plăcile Arduino rulează la 16MHz – adică 16 milioane de cicluri pe secundă. Instrucțiunile ATmega328P durează între 1 și 3 cicluri de ceas (cu excepția instrucțiunilor legate de subrutine, care durează 4 sau 5 cicluri). Media este undeva între 1 și 2 pentru majoritatea codului C compilat.

Timp pe Ciclu de Ceas:

Timpul pentru fiecare ciclu de ceas se calculează astfel:

$$\text{Timp pe ciclu de ceas} = \frac{1}{\text{Frecvența de ceas}} = \frac{1}{16\,\text{MHz}} = \frac{1}{16 \times 10^6\,\text{Hz}} \approx 62.5\,\text{nanosecunde}$$

Timp de Execuție:

Pentru a estima timpul de execuție pentru o linie specifică de cod, ar trebui să se înmulțească numărul de cicluri de ceas necesare cu timpul pe ciclu de ceas. De exemplu:

- Dacă o instrucțiune necesită 1 ciclu de ceas:

$$\text{Timp de execuție} = 1\,\text{ciclu} \times 62.5\,\text{ns} = 62.5\,\text{ns}$$

- Dacă o instrucțiune necesită 2 cicluri de ceas:

$$\text{Timp de execuție} = 2\,\text{cicluri} \times 62.5\,\text{ns} = 125\,\text{ns}$$

3.2.7 Multitasking

Pentru a realiza multitasking-ul am creat o structură de date numită ”Task” care conține un pointer la funcția corespunzătoare task-ului, timpul la care a fost executat task-ul ultima dată și intervalul la care trebuie executat. Apoi la funcția ”loop” funcționează ca un programator pentru task-uri, iar la fiecare iterație se ia task-ul care așteaptă de o perioadă de timp mai mare decât intervalul lui de execuție și totodată care așteaptă de cel mai mult timp. Totodată rezultatele obținute în task-urile care realizează operații de citire se memorează într-un buffer circular, pentru ca mai apoi task-urile care realizează procesarea datelor să poată preluat datele din ele, astfel se evită eventualele probleme care apar dacă funcțiile task-urile se execută cu o frecvență diferită în diferite momente ale rulării programului.

4. Design

Schema de montaj:

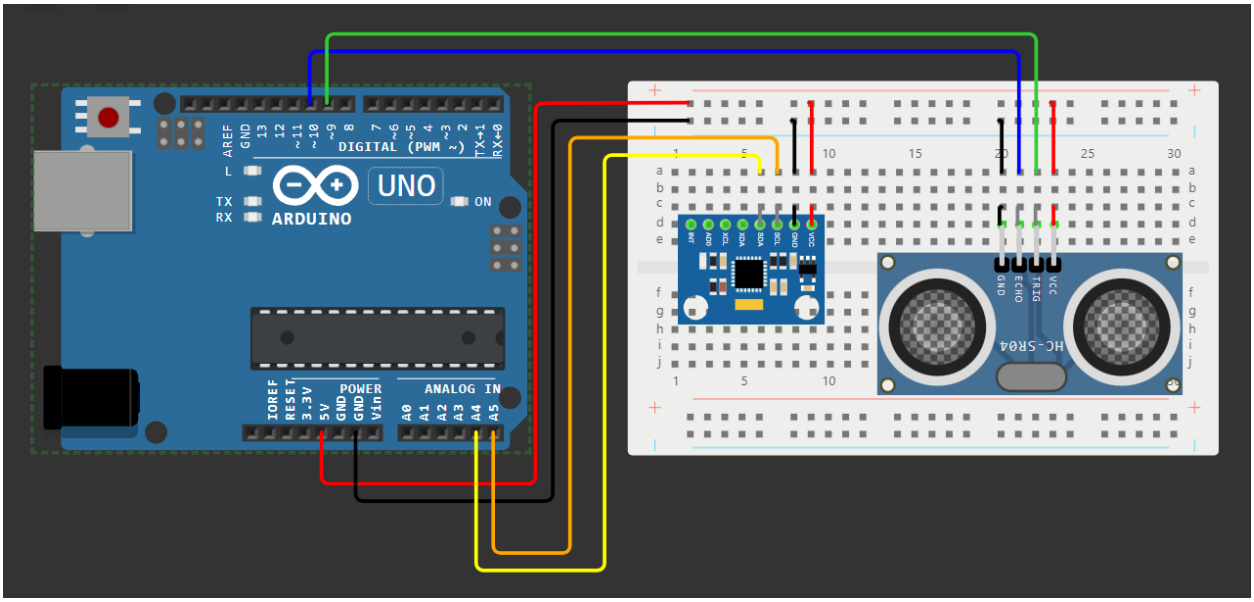


Figura 4.1.1.1: Schemă de montaj

Interfața grafică:

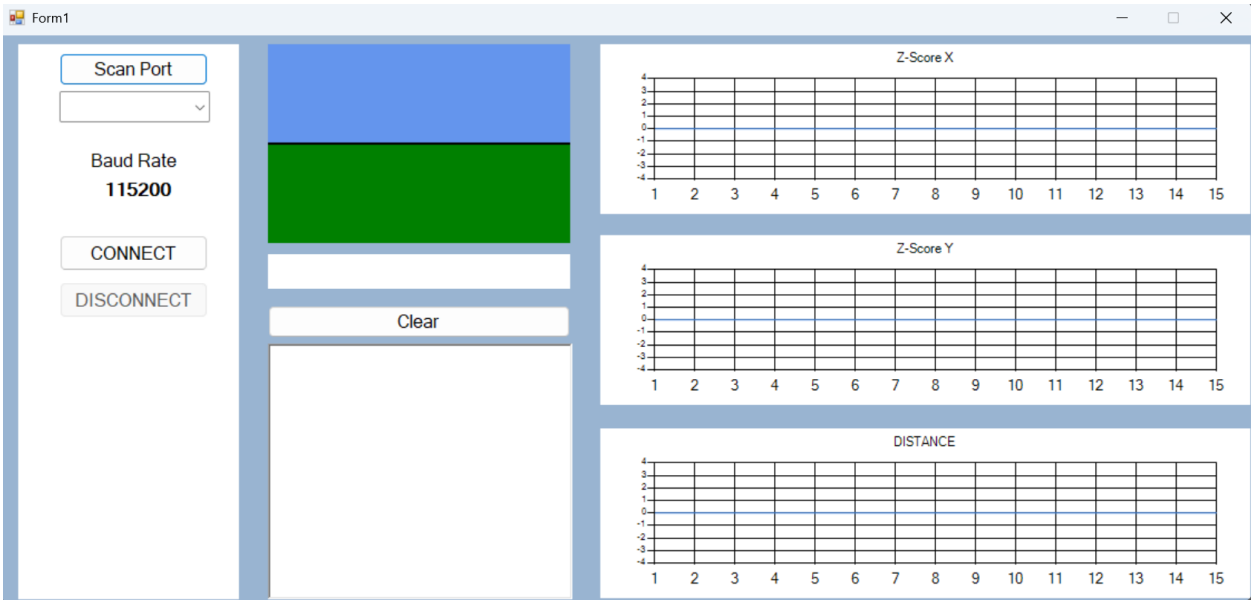


Figura 4.1.1.2: Interfață grafică

5. Implementare

Am implementat:

- Aplicație care să comunice cu Arduino
- Varianta cod în care se folosește metoda simplă de calcul, single-tasking
- Varianta cod în care se folosește algoritmul lui Welfort, single-tasking
- Varianta cod în care se folosește metoda simplă de calcul, multitasking
- Varianta cod în care se folosește algoritmul lui Welfort, multitasking

6. Testare

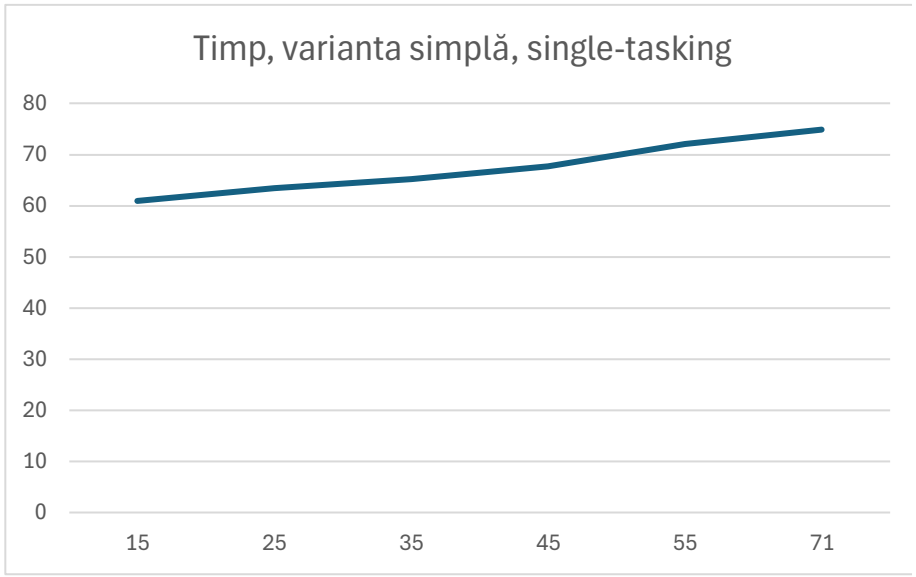
6.1 Variația timpului de execuție și a memoriei pentru diferite dimensiuni ale fereastrei de timp

Pentru determinarea duratei de execuție a funcției „loop”, se realizează un set de 500 de rulări consecutive, iar timpul de execuție corespunzător fiecărei rulări este înregistrat. Ulterior, se calculează media aritmetică a valorilor obținute, această medie și valoarea maximă de memorie folosite fiind transmise aplicației pentru a fi afișată utilizatorului. Procesul este repetat pentru diverse dimensiuni ale ferestrei de timp, permițând astfel analiza performanței funcției în contexte diferite.

6.1.1 Rezultate varianta simplă, single-tasking

Programul folosește 12872 bytes, din memoria flash.

Fereastra	Timp (ms)	Memorie (bytes)
15	61	989
25	63.48	1148
35	65.22	1311
45	67.67	1468
55	72.07	1629
71	74.93	1887



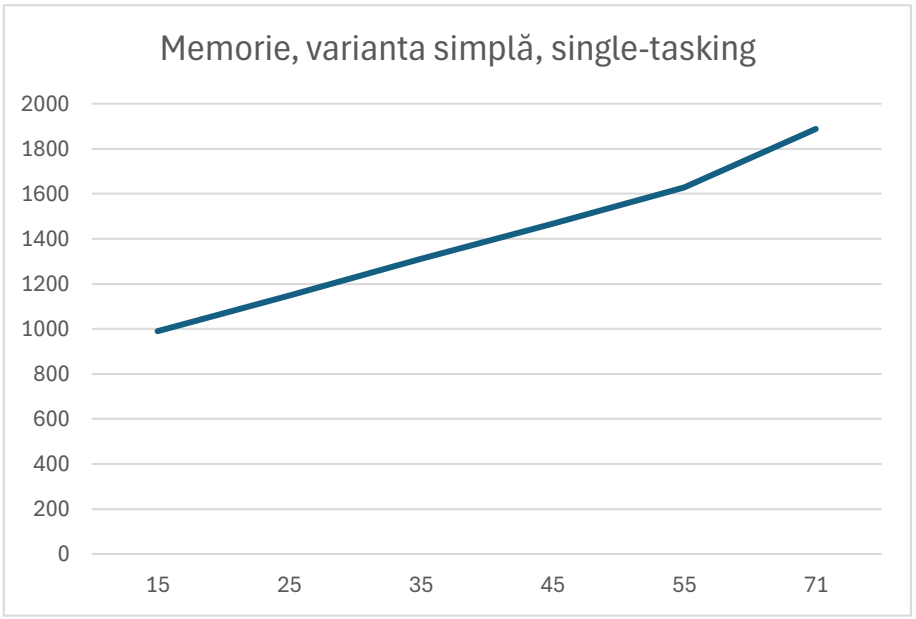


Figura 6.1.1.1: Varianta simplă single-tasking

6.1.2 Rezultate varianta în care se folosește algoritmul lui Welfort, single-tasking

Programul folosește 13382 bytes, din memoria flash.

Fereastra	Timp (ms)	Memorie (bytes)
15	62.63	1029
25	63.1	1181
35	64.11	1339
45	65.38	1499
55	64.79	1662
69	67.49	1886

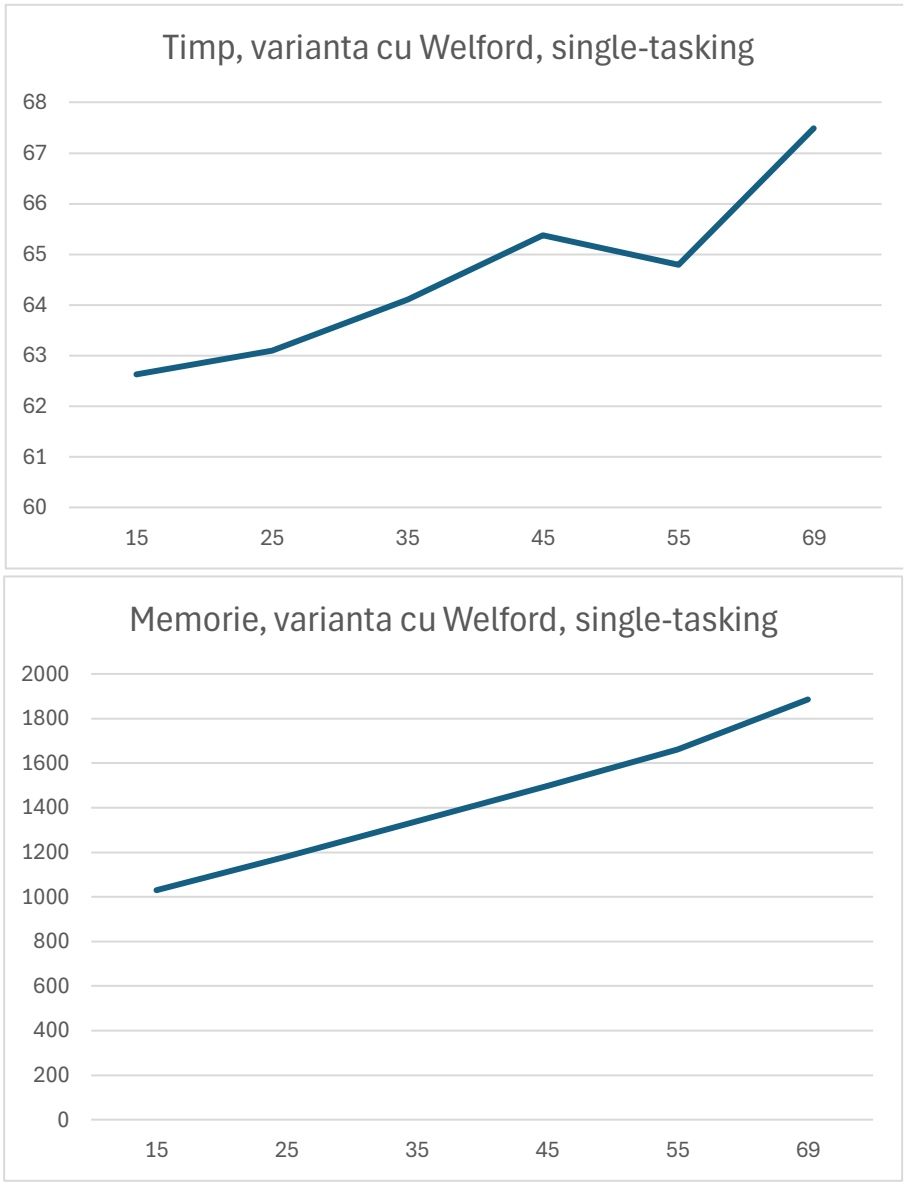
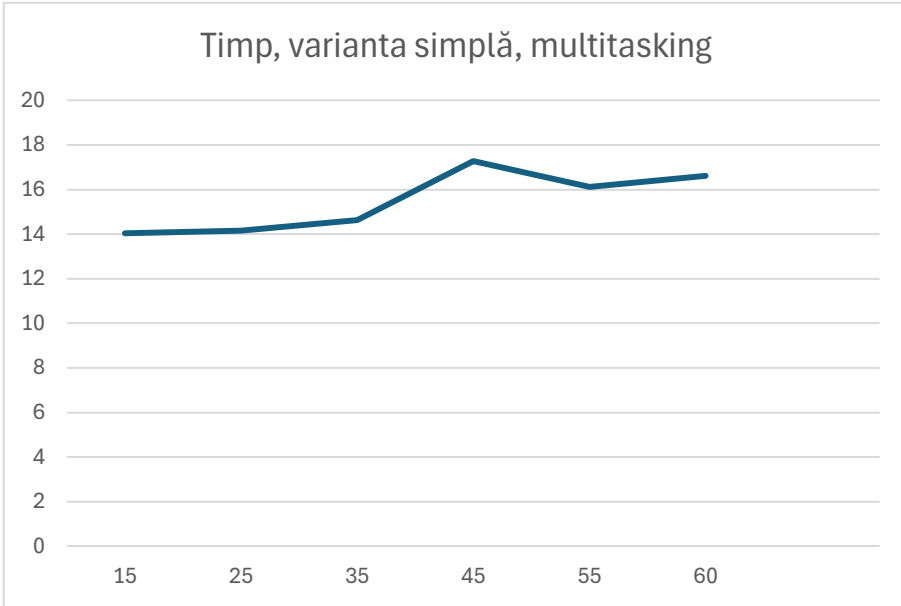


Figura 6.1.2.1: Varianta cu Welford single-tasking

6.1.3 Rezultate varianta simplă, multitasking

Programul folosește 13920 bytes, din memoria flash.

Fereastra	Timp (ms)	Memorie (bytes)
15	14.03	1174
25	14.16	1336
35	14.64	1494
45	17.29	1657
55	16.13	1818
60	16.61	1892



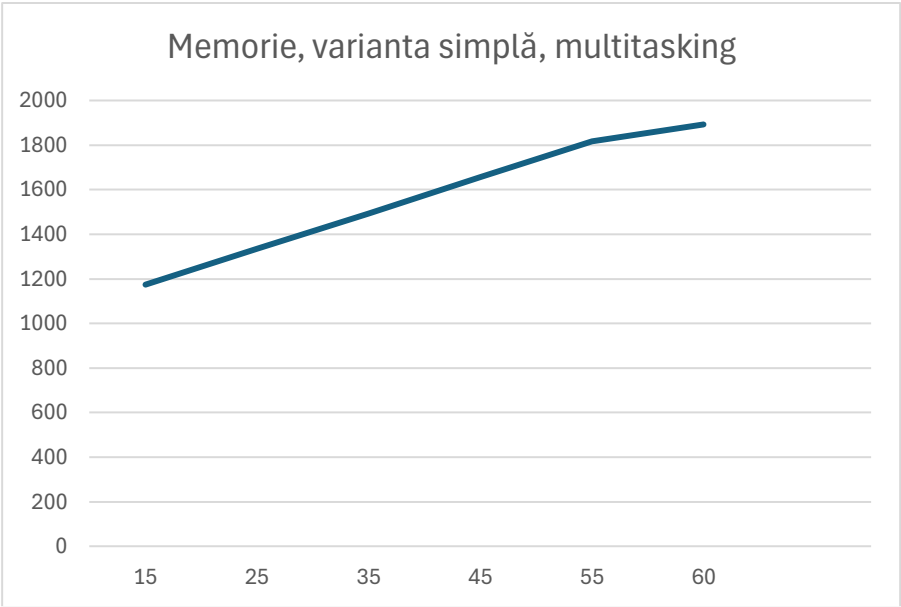
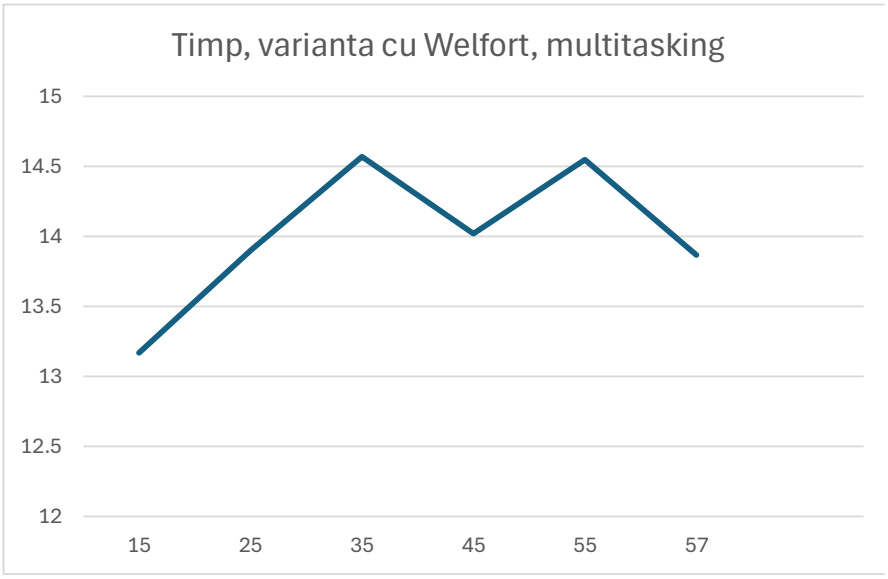


Figura 6.1.3.1: Varianta simplă multitasking

6.1.4 Rezultate varianta în care se folosește algoritmul lui Welfort, multitasking

Programul folosește 14474 bytes, din memoria flash.

Fereastra	Timp (ms)	Memorie (bytes)
15	13.17	1213
25	13.9	1374
35	14.57	1536
45	14.02	1695
55	14.55	1854
57	13.87	1886



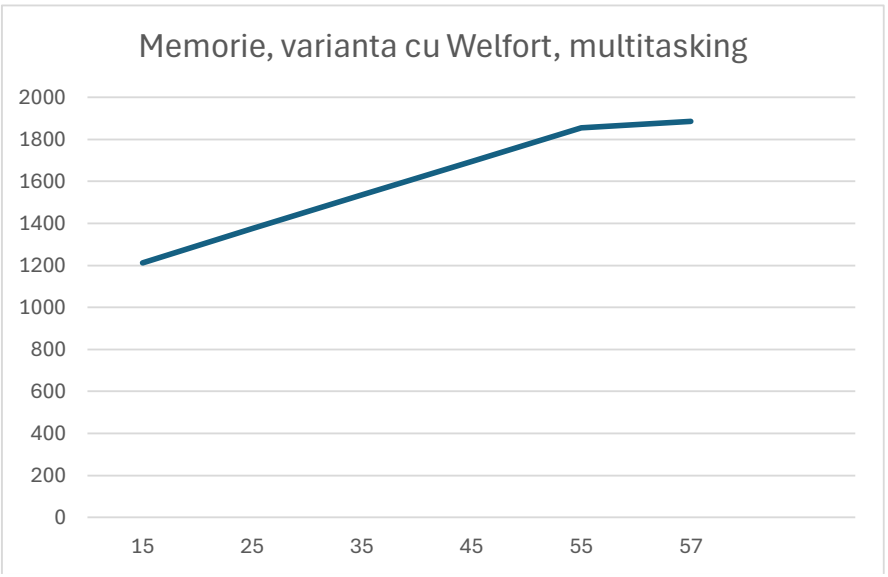


Figura 6.1.4.1: Varianta cu Welford multitasking

6.2 Comparare rezultate

6.2.1 Comparare din punct de vedere al timpului de execuție

Dimensiune maximă fereastră: varianta simplă single-threading 71, varianta cu Welford single-threading 69, varianta simpla multithreading 60, iar varianta cu Welford multithreading 57.

Fereastra	Timp varianta simplă single-tasking (ms)	Timp varianta Welford single-tasking (ms)	Timp varianta simplă multi-tasking (ms)	Timp varianta Welford multi-tasking (ms)
15	61	62.63	14.03	13.17
25	63.48	63.1	14.16	13.9
35	65.22	64.11	14.64	14.57
45	67.67	65.38	17.29	14.02
55	72.07	64.79	16.13	14.55
75	74.93	67.49	16.61	13.87

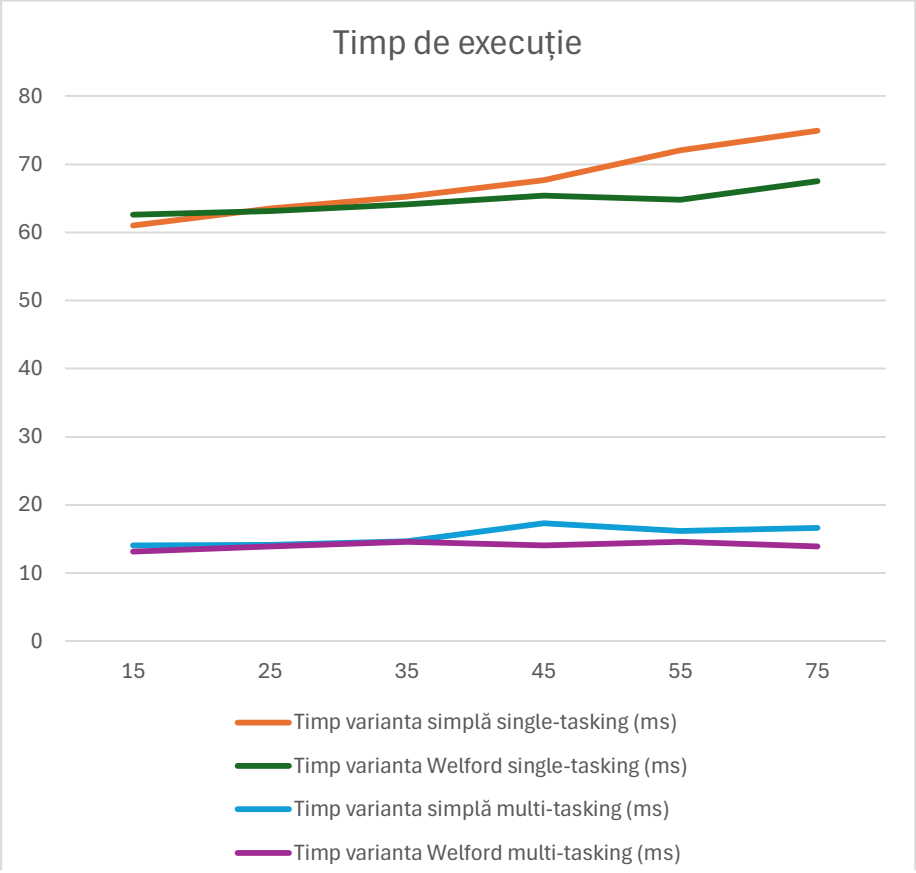


Figura 6.2.1.1: Compatație d.p.d.v. timp execuție

Din perspectiva timpului de execuție, implementările multitasking prezintă o eficiență semnificativ superioară, fiind de cel puțin trei ori mai rapide în comparație cu variantele single-tasking, atât în cazul utilizării algoritmului lui Welford, cât și în cazul abordării simple. Această superioritate devine tot mai evidentă pe măsură ce dimensiunea ferestrei de date pe care se realizează calculele crește. În mod particular, în ambele scenarii – multitasking și single-tasking – algoritmul lui Welford se distinge prin performanțe superioare, demonstrând o rapiditate mai mare decât varianta simplă. Deși diferențele de timp de execuție sunt neglijabile atunci când se lucrează cu ferestre de date de dimensiuni reduse, acestea devin semnificative și ușor observabile odată cu creșterea volumului de date procesate. Astfel varianta cu Welford multitasking e cea mai eficientă din punct de vedere al timpului de execuție.

6.2.2 Comparare din punct de vedere al memoriei

Dimensiune maximă fereastră: varianta simplă single-threading 71, varianta cu Welford single-threading 69, varianta simpla multithreading 60, iar varianta cu Welford multithreading 57.

Fereastra	Memorie varianta simplă single-tasking (bytes)	Memorie varianta Welford single-tasking (bytes)	Memorie varianta simplă multi-tasking (bytes)	Memorie varianta Welford multi-tasking (bytes)
15	989	1029	1174	1213
25	1148	1181	1336	1374
35	1311	1339	1494	1536
45	1468	1499	1657	1695
55	1629	1662	1818	1854
75	1887	1886	1892	1886

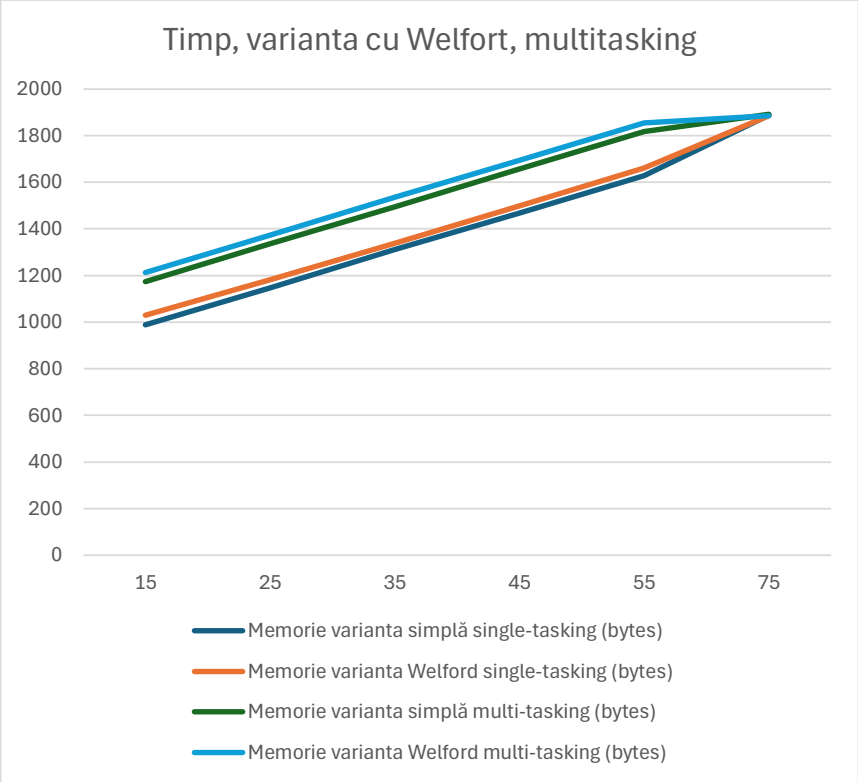


Figura 6.2.2.1: Comparație d.p.d.v. memorie folosită

Din perspectiva consumului de memorie, implementările single-tasking se remarcă printr-o cerință redusă de resurse comparativ cu cele multitasking, ceea ce permite alocarea unui spațiu mai generos pentru dimensiunea ferestrei de date. În plus, algoritmul simplu demonstrează un consum de memorie mai mic în ambele tipuri de implementări, fie că este vorba despre single-tasking sau multitasking. Prin urmare, varianta simplă în modul single-tasking se poziționează drept cea mai eficientă opțiune din punct de vedere al utilizării memoriei. Această caracteristică aduce un avantaj important, deoarece dimensiunea mai mare a ferestrei de date permite obținerea unor rezultate mai precise, îmbunătățind astfel calitatea și relevanța analizelor efectuate.

6.3 Concluzie

Fiecare soluție adusă în discuție prezintă atât avantaje, cât și dezavantaje specifice, însă, în opinia mea, beneficiul obținut prin reducerea timpului de execuție este mult mai semnificativ decât economiile de memorie realizate atunci când se acceptă un sacrificiu al performanței temporale. Astfel, o implementare care prioritizează eficiența temporală devine mai valoroasă decât una care se concentrează pe optimizarea utilizării memoriei, având în vedere că, în majoritatea cazurilor, viteza de procesare reprezintă un factor mult mai important decât economiile de resurse.

7. Bibliografie

1. https://en.wikipedia.org/wiki/Microcontroller
   -Data vizitării: 19.10.2024

2. https://roboticsbackend.com/how-to-do-multitasking-with-arduino/
   - Autor: Edouard Renard, Data vizitării: 24.10.2024

3. https://roboticsbackend.com/arduino-protothreads-tutorial/
   - Autor: Edouard Renard, Data vizitării: 24.10.2024

4. https://medium.com/@araffin/simple-and-robust-computer-arduino-serial-communication-f91b95596788
   - Autor: Antonin RAFFIN, Data vizitării: 24.10.2024

5. https://docs.arduino.cc/hardware/uno-rev3/
   - Data vizitării: 24.10.2024

6. https://howtomechatronics.com/tutorials/arduino/arduino-and-mpu6050-accelerometer-and-gyroscope-tutorial/
   - Autor: Dejan, Data vizitării: 24.10.2024

7. https://ro.wikipedia.org/wiki/For%C8%9Ba_Coriolis
   - Autori: Petre Pascu, 2A02:8071:3E90:6200:18E7:5F2C:16DA:988F, 2A02:8071:3E90:6200:844B:A5F4:B612:9D89, 2A02:8071:3E90:6200:78EC:74BA:AA43:7079, FoxBot, 2A02:8071:3E90:6200:58A6:C44A:D055:E50E, Miehs, 92.230.230.246, Falseufals Nicolae Coman, Data vizitării: 24.10.2024

8. https://howtomechatronics.com/tutorials/arduino/ultrasonic-sensor-hc-sr04/
   - Autor: Dejan, Data vizitării: 24.10.2024

9. https://www.megunolink.com/articles/coding/3-methods-filter-noisy-arduino-measurements/
   - Data vizitării: 24.10.2024

10. https://roboticsbackend.com/how-to-do-multitasking-with-arduino/#Lets_start_multitasking
    - Autor: Edouard Renard, Data vizitării: 24.10.2024

11. https://jonblack.me/blog/2015/arduino-multitasking-using-finite-state-machines/

-Autor: Jon Black, Data vizitării: 24.10.2024

12. https://docs.arduino.cc/libraries/taskscheduler/
    -Data vizitării: 24.10.2024

13. https://learn.circuit.rocks/nine-techniques-to-optimize-your-arduino-code
    -Data vizitării: 24.10.2024

14. https://www.electronicwings.com/sensors-modules/mpu6050-gyroscope-accelerometer-temperature-sensor-module
    - Autor: ElectronicWings, Data vizitării: 03.11.2024

15. https://www.electronicwings.com/sensors-modules/ultrasonic-module-hc-sr04
    - Autor: ElectronicWings, Data vizitării: 03.11.2024

16. https://docs.arduino.cc/retired/archived-libraries/CurieTimerOne/
    -Data vizitării 03.11.2024

17. https://www.investopedia.com/terms/z/zscore.asp
    -Autor: Scott Nevil, Data vizitării: 03.11.2024

18. https://maker.pro/arduino/tutorial/how-to-clean-up-noisy-sensor-data-with-a-moving-average-filter
    -Autor: Brett Garberman, Data vizitării: 07.11.2024

19. https://calculator.academy/gyroscope-angle-calculator/
    -Autor: Calculator Academy Team, Data vizitării: 07.11.2024

20. https://www.dofbot.com/post/angle-measure-using-mpu-6050-3-axis-accelerometer-and-gyroscope-sensor
    -Autor Ramesh G, Data vizitării: 07.11.2024

21. https://dsdojo.medium.com/running-totals-and-moving-averages-eb72f0738b16
    -Autor: Debanjan Saha, Data vizitării: 08.11.2024

22. https://en.wikipedia.org/wiki/Algorithms_for_calculating_variance
    -Autor:, Data vizitării: 21.11.2024

23. https://www.quora.com/How-much-time-does-Arduino-Uno-take-to-execute-each-line-of-the-code-known-that-its-microcontroller-clock-s-speed-is-16-MHz
    -Autor: Rod Nussbaumer, Data vizitării: 21.11.2024

24. https://cybergibbons.com/uncategorized/arduino-misconceptions-2-arduino-is-slow/
-Autor: Cybergibbons, Data vizitării: 21.11.2024

Plan de lucru

Întâlnire proiect 1:

- Alegere proiect

Întâlnire proiect 2:

- Scris partea de introducere si studio bibliografic

Întâlnire proiect 3:

- Scris partea de analiză și design
- Schema interfață grafică
- Implementare parțială program pentru Arduino și aplicație

Întâlnire proiect 4:

- Scris partea de implementare și testare + validare
- Terminat interfață grafică și prima variantă de program pentru Arduino

Întâlnire proiect 5:

- Optimizat program arduino
- Modificări documentație(dacă apar pentru implementarea nouă) și scris partea de concluzii

Întâlnire proiect 6:

- Testare și eventuale modificări

Întâlnire proiect 7:

- Prezentare