Sem. I 2024-2025

# Proiect PATR

## Sistemul de navigație al unui avion militar ce calculează viteza acestuia pe baza informațiilor primite de la accelerometru si giroscop

## Echipa: 7

## Data: 17 ianuarie 2025

Componența echipei și contribuția individuală (în % )

| Membrii | A | C | D | PR | CB | e-mail |
|---|---|---|---|---|---|---|
| Vasile Ioan Teodor | 30% | 30% | 40% | 50% | 37.5% | ioan_teodor.vasile@stud.acs.upb.ro |
| Lungulescu Raul Adrian | 40% | 5% | 40% | 25% | 27.5% | raul.lungulescu@stud.acs.upb.ro |
| Gandraman Iulian | 30% | 65% | 20% | 25% | 35% | iulian.gandraman@stud.acs.upb.ro |
| | 100% | 100% | 100% | 100% | | |

*A-analiză problemă și concepere soluție, C- implementare cod, D-editare documentație, PR-"proofreading", CB - contribuție individuală totală ([%])*

*Membrii echipei declară că lucrarea respectă toate regulile privind onestitatea academică. În caz de nerespectare a acestora tema va fi notată cu* ***0(zero) puncte***

# Cuprins

# 1 Introducere. Definire problemă

În cadrul acestui proiect, se dorește dezvoltarea unei aplicații în timp real care să calculeze și să afișeze viteza unui avion militar pe baza datelor primite de la doi senzori: accelerometrul și giroscopul. Sistemul va afișa viteza aeronavei pe un monitor la fiecare secundă, măsurând datele de la senzori la un interval fix.

Avionul militar are un sistem de navigație care se bazează pe datele colectate de cei 2 senzori:

- **Accelerometrul** - care furnizează accelerațiile ($\ddot{x}$,$\ddot{y}$,$\ddot{z}$) la fiecare 5 milisecunde
- **Giroscopul** - care furnizează vitezele unghiulare ($\dot{\theta}$,$\dot{\phi}$,$\dot{\psi}$), aferente unghiurilor de ruliu (roll, $\theta$), tangaj(pitch, $\phi$) și girație(yaw, $\psi$), la fiecare 40 milisecunde

Pentru a ajunge de la datele brute citite de senzor la viteze, este nevoie de câteva prelucrări matematice și numerice a datelor, cum ar fi integrarea numerică sau compunerea vectorială a vitezelor pe cele 3 planuri de deplasare ale avionului. În cadrul abordării problemei, un obstacol semnificativ îl constituie prezența accelerației gravitaționale. Gravitația introduce o componentă constantă în măsurătorile accelerometrului, care poate afecta acuratețea datelor brute privind accelerația liniară a avionului. Această componentă interferează cu procesul de determinare a accelerației reale, necesare pentru calculul vitezei relative la sol, deoarece senzorul de accelerare nu distinge între accelerația generată de mișcare și cea datorată atracției gravitaționale.

În special, pe axele verticale și în scenarii în care avionul se află în unghiuri de înclinare (roll, pitch), gravitația poate modifica percepția accelerometrului asupra axelor respective, rezultând erori în transformarea datelor din sistemul local al avionului în coordonate globale. Această problemă necesită utilizarea unor tehnici de filtrare, cum ar fi filtrul complementar sau Kalman, care combină datele de la giroscop și accelerometru pentru a separa accelerația gravitațională de accelerația specifică mișcării avionului.

# 2 Analiza problemei

Datele măsurate de accelerometru si giroscop furnizează accelerațiile și vitezele unghiulare în raport cu sistemul de referință (mobil) al avionului, pentru a găsi o legătura între acesta și sistemul de referință (fix) al pământului folosim 3 transformări (rotații) prin următoarele matrice:

**Roll Rotation ($\phi$):**

$$R_1(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}$$

**Pitch Rotation ($\theta$):**

$$R_2(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

**Yaw Rotation ($\psi$):**

$$R_3(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Pentru a putea trece de la sistemul de referință legat de avion la cel legat de pământ, matricea de trece între cele 2 sisteme de referința este $R_3(\psi)R_2(\theta)R_1(\phi)$. Aceasta face trecerea de la sistemul de referință fix la cel mobil, fixat pe avion. Pentru a face trecerea inversă vom folosi matricea $(R_3(\psi)R_2(\theta)R_1(\phi))^{-1}$. Așadar, pentru a calcula vitezele față de pământ vom folosi următoarea relație:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = (R_1(\varphi)R_2(\theta)R_3(\psi))^{-1} \begin{bmatrix} u \\ v \\ w \end{bmatrix},$$

unde

- $u$, $v$, $w$ sunt componentele vitezei în sistemul de referință al avionului
- $\dot{x}$, $\dot{y}$, $\dot{z}$ reprezintă componentele vitezei în sistemul de referință al pământului

Calculul matricei inversate:

$$(R_1(\varphi)R_2(\theta)R_3(\psi))^{-1} = R_3(\psi)^{-1}R_2(\theta)^{-1}R_1(\varphi)^{-1} = R_3(-\psi)R_2(-\theta)R_1(-\varphi)$$

$$= \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta\cos\psi & \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi \\ \cos\theta\sin\psi & \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix}$$

Pentru a calcula vitezele $u$, $v$, $w$ și unghiurile $\theta$, $\phi$, $\psi$ vom integra, folosind metoda Euler, datele măsurate și obținem:

$$\begin{aligned} u &= u_o + a_x\Delta t_1 & \theta &= \theta_o + \dot{\theta}\Delta t_2 \\ v &= v_o + a_y\Delta t_1 & \phi &= \phi_o + \dot{\phi}\Delta t_2 \\ w &= w_o + a_z\Delta t_1 & \psi &= \psi_o + \dot{\psi}\Delta t_2 \end{aligned}$$

După care vom aplica un filtru complementar pe unghiuri, pentru a elimina zgomotul produs de giroscop:

$$\theta = 0.98\theta_o + 0.02a_x$$
$$\phi = 0.98\phi_o + 0.02a_x$$

Analiza unor secvențe corecte de execuție pentru a confirma rularea corespunzătoare a aplicației:

- Secvența 1: Repaus: Toate unghiurile de zbor sunt nule, vitezele sunt nule
- Secvența 2: Zbor liniar cu viteză constantă pe orizontală
  - Avionul zboară pe o traiectorie dreaptă și orizontală, fără manevre sau înclinări semnificative.
  - Ruliu (Roll): Se menține constant la 0°. Avionul nu se rotește în jurul axei X, rămânând stabil pe direcția de zbor.
  - Tangaj (Pitch): Se menține constant la 0°. Avionul nu se înclină înainte sau înapoi.
  - Girație (Yaw): Poate varia foarte ușor în funcție de mișcările de ajustare a direcției, dar în general rămâne aproape constant, pentru că avionul zboară într-o linie dreaptă.
  - Accelerațiile pe cele 3 axe vor fi nule
  - Viteza pe X o să aibă variații mici, fiind aproape constantă.
  - Vitezele pe Y și Z vor fi nule.
- Secvența 3: Avion în urcare
  - Avionul efectuează o urcare treptată. Acesta se va înclina pe axa de tangaj în sus pentru a schimba altitudinea. Dacă avionul virează ușor, vor exista și modificări pe axa de ruliu.
  - Ruliu : Dacă avionul virează ușor spre stânga sau dreapta, valoarea de ruliu va începe să se modifice (se va înclina ușor pe axa X).

  - Tangaj : Va crește treptat (poate ajunge la 5-10° sau mai mult), pentru că avionul se înclină în sus pe axa Y în timpul urcării.
  - Girație : Poate rămâne constant dacă avionul urcă pe o traiectorie dreaptă, dar va varia ușor dacă avionul virează în timpul urcării.
  - Viteza pe axa x tinde să scadă ușor, deoarece o parte din tracțiune este utilizată pentru a câștiga altitudine.
  - Viteza pe axa y crește treptat, pe măsură ce avionul urcă. Este responsabilă pentru modificarea altitudinii.
  - Viteza pe axa z Dacă urcarea este pe o traiectorie dreaptă, această viteză rămâne aproximativ zero.
  - Accelerația pe x tinde să fie negativă, deoarece viteza pe x scade ușor în timpul urcării.
  - Accelerația pe y este pozitivă, deoarece avionul urcă. Aceasta reflectă forța netă care învinge gravitația și permite creșterea altitudinii
  - Accelerația pe z rămâne aproape zero dacă traiectoria este rectilinie în plan orizontal.

- Secvența 4: Viraj abrupt sau manevră de rotire
  - Avionul efectuează un viraj abrupt sau o manevră de rotire (de exemplu, un viraj de 90 de grade). În timpul manevrei, avionul va experimenta modificări semnificative ale unghiurilor de ruliu, tangaj și giratie.
  - Ruliu va ajunge la valori mari (20-40°), în funcție de cât de abrupt este virajul.
  - Tangaj va varia mai puțin, dar va fi afectat de manevra de viraj
  - Girație va crește rapid (sau va scădea rapid), reflectând schimbarea de direcție.
  - Este negativă în timpul virajului, deoarece viteza orizontală scade ușor.
  - Rămâne aproape zero dacă altitudinea este constantă, dar crește sau scade în funcție de tangaj, dacă avionul urcă sau coboară în timpul virajului.
  - Este foarte mare în timpul virajului abrupt, reflectând forțele centrifuge care acționează asupra avionului. Aceasta este componenta dominantă a accelerației în această manevră.
  - Scade pe parcursul virajului, deoarece o parte din energia motorului este utilizată pentru a menține rotația și contracararea forțelor laterale.
  - Rămâne relativ constantă dacă altitudinea nu se schimbă. Dacă virajul implică și o urcare sau coborâre, viteza verticală se va modifica corespunzător.
  - Devine semnificativă, deoarece avionul se deplasează lateral în timpul virajului. Această componentă reflectă schimbarea direcției și devine dominantă pentru un viraj brusc.

Evaluarea unor secvente eronate de executie:

- Secvența 1: Erorile de măsurare ale senzorului
  - Senzorul MPU6050 este unul destul de slab si poate prezenta erori în furnizarea corectă a vitezelor unghiulare și accelerațiilor, cum ar fi zgomotul sau alunecarea. Aceste erori se cumuleaza treptat și afectează performanța sistemului.
  - Ruliu (Roll) și Tangaj (Pitch) pot avea valori incorecte, iar unghiul de Girație (Yaw) poate devine eronat.
  - Calculul vitezei față de Pământ ar fi afectat semnificativ, deoarece va integra accelerațiile incorecte, ducând la o estimare eronată a mișcării avionului.
- Secvența 2: Eroare matematică în procesul de integrare numerica (erori cumulative de calcul)
  - O altă sursă de erori poate fi legată de procesul de integrare al accelerațiilor și al vitezelor unghiulare
  - Dacă valorile accelerației nu sunt corect integrate (de exemplu, folosind intervalul de timp prea mare), integrarea accelerațiilor poate conduce la un calcul eronat al vitezei.
  - Dacă eroarea se cumulează prea mult, vitezometrul va indica valori eronate ale vitezei.
  - Această eroare ar putea afecta controlul avionului, deoarece unghiurile de ruliu și tangaj ar putea să nu reflecte corect mișcarea reală a avionului.
- Secvența 3: Eroare de sincronizare (sincronizarea greșită a task-urilor)
  - O eroare de sincronizare poate apărea dacă task-urile care citesc datele de la senzori și cele care le integrează sau le folosesc pentru calcule nu sunt corect sincronizate.
  - Dacă elementele de sincronizare nu sunt gestionate corect între task-uri, există riscul ca un task să acceseze datele înainte ca alte task-uri să le actualizeze complet.
  - Unghiurile de ruliu, tangaj, girație și accelerațiile pentru fiecare axă vor fi incorecte.

# 3 Aplicația. Structura și soluția de implementare propusă

## 3.1 Definirea structurii aplicației

Aplicația va fi împărțită pe mai multe task-uri corespunzătoare diferitelor entități care vor executa anumite operații: CitireAccelerometru, CitireGiroscop, IntegrareAccelerometru, IntegrareGiroscop, CalculareViteza, AfisareViteza

#### Task de citire date accelerometru

Acest task gestionează citirea datelor de la accelerometrul senzorului MPU-6050. În timpul inițializării, senzorul este calibrat pentru a elimina efectele gravitației și a compensa abaterile hardware (bias). Datele brute de accelerare pe axele X, Y și Z sunt obținute și ajustate folosind valorile de bias calculate în timpul calibrarii. Task-ul utilizează un semafor pentru a asigura accesul sincronizat la structura globală accelData, astfel încât alte task-uri să poată utiliza aceste date fără a apărea conflicte.

Logic, acest task reprezintă "ochiul" sistemului, captând datele brute de mișcare ale dispozitivului. Prin calibrare, elimină erorile statice și oferă un flux de informații precise. Datele obținute de accelerometru sunt esențiale pentru calcularea vitezelor.

#### Task de citire date giroscop

Task-ul gestionează citirea datelor de la giroscopul senzorului al doilea MPU-6050. Similar cu accelerometrul, inițial senzorul este calibrat pentru a elimina erorile de offset pe cele trei axe (ruliu, tangaj și girație). Datele de rotație sunt apoi stocate într-o structură globală protejată de mutex (gyroData), asigurând accesul controlat pentru task-urile care vor utiliza aceste informații.

Din punct de vedere fizic, acest task captează schimbările de orientare ale dispozitivului, măsurând viteza unghiulară. Vitezele unghiulare masurate sunt complementare informațiilor furnizate de accelerometru pentru a oferi o imagine completă a mișcării dispozitivului.

#### Task de integrare date accelerometru

Acest task calculează vitezele instantanee pe axele X, Y și Z prin integrarea valorilor de accelerare obținute de la TaskReadAccelerometer. După ce preia accesul la datele globale cu ajutorul unui

mutex, accelerația este integrată în timp folosind o metoda de integrare numerica (metoda Euler), iar rezultatul este actualizat în variabilele globale pentru viteze.

Din perspectivă fizică, acest task transformă datele de accelerație brute în informații despre mișcare, cum ar fi viteza. Este o etapă critică pentru a determina cât de rapid și în ce direcție se deplasează avionul.

#### Task de integrare date giroscop

Task-ul calculează unghiurile de ruliu, tangaj și girație prin integrarea datelor de la giroscop. După ce obține datele protejate prin mutex, acestea sunt integrate numeric pentru a furniza pozițiile unghiulare ale avionului. Valorile sunt actualizate în variabile globale pentru a fi folosite de alte task-uri. Acest task primește de asemenea și accelerații, protejate prin semafor, de la task-ul de citire.

Acest task transformă viteza unghiulară în unghiuri, oferind o descriere detaliată a orientării dispozitivului, cu ajutorul integrării și a filtrării complementare.

#### Task de calculare a vitezei fața de sistemul de referință al pământului

Acest task combină informațiile de la accelerometru și giroscop pentru a calcula viteza în raport cu sistemul de referință al Pamantului. Astfel, folosind datele integrate de cele 2 taskuri și aplicând 3 rotații obținem viteza pe fiecare axă. În plus, se calculează și viteza totală.

Din perspectivă fizică, acest task realizează transformarea datelor dintr-un sistem de referință mobil într-un sistem de referință fix. Este esențial pentru a înțelege cum se mișcă dispozitivul într-un spațiu tridimensional relativ la un cadru de referință fix, cum ar fi Pământul.

#### Task de afisare

Scopul acestui task este să afișarea vitezelor calculate pentru fiecare axă (X, Y, Z) și viteza totală, folosind Serial Monitor. Acesta funcționează într-un ciclu periodic, sincronizat cu celelalte task-uri, și afișează datele la intervale regulate pentru a monitoriza performanța și funcționarea sistemului.

Rolul fizic al acestui task este de a oferi feedback utilizatorului sau operatorului despre starea sistemului în timp real.

## 3.2 Definirea soluției în vederea implementării

Soluţia a fost implementată pe o plăcută Arduino Mega 2560, folosind FreeRTOS.h.

### Mecanisme de sincronizare între task-uri

Proiectul utilizează mutex-uri, timere și semafoare din FreeRTOS pentru a proteja accesul la resursele partajate, asigurând integritatea datelor în medii concurente.

- **mutexCitireA** și **mutexCitireG**: Protejează datele brute de accelerometru și giroscop împotriva accesului simultan de către alte task-uri.
- **mutexIntG**: Coordonează integrarea datelor citite de la giroscop, prevenind conflictele între task-urile care citesc și cele care integrează.
- **mutexIntA**: Contrar numelui, acesta este un semafor cu valoarea maximă 2, care protejează datele citite de accelerometru, și care sunt accesate simultan atât de task-ul pentru integrarea vitezelor unghiulare, cât și cea de integrare a accelerațiilor.
- **mutexCalcA** și **mutexCalcG**: Asigură exclusivitatea în calcularea vitezelor și a altor transformări bazate pe datele integrate.
- **Timer** de afișare a vitezelor: întrucât vitezele sunt calculate la intervale foarte mici, aproximativ 5ms, ar fi foarte obositor pentru ochiul uman să urmărească toate datele afișate. Drept pentru care acest timer afișează vitezele la un interval prestabilit (5000ms) pentru a fi mai ușor de citit.

Mutex-urile sunt create cu blocare limitată prin portMAXDELAY, ceea ce asigură că un task nu rămâne blocat permanent.

### Mecanisme de comunicare între task-uri

Task-ul de citire accelerometru (**TaskReadAccelerometer**) semnalează integrarea accelerometrului prin **xSemaphoreGive(mutexIntA)**. Acest lucru permite ca **TaskIntegrateAcceleration** să preia datele procesate și să continue calculele.

Fiecare task se poate concentra pe o singură responsabilitate, comunicând rezultate prin mecanisme bine definite, asigurand că procesarea datelor este secvențială, respectând dependențele între task-uri

### Planificarea task-urilor pe condiții de timp

FreeRTOS permite planificarea task-urilor utilizând priorități și întârzieri temporale **(vTaskDelay)**.

- **TaskReadAccelerometer** rulează cu o întârziere de 5 ms, sincronizându-se cu frecvența de citire a senzorului.
- **TaskReadGyroscope** rulează cu o întârziere de 40 ms, respectând necesitatea de a obține date mai rare de la giroscop.
- **Task-uri de integrare și calcul** sunt sincronizate cu task-urile de citire prin mutex,

evitând problemele de concurență.

Task-urile inactive intră în starea „blocată” (Blocked), eliberând procesorul pentru alte operațiuni

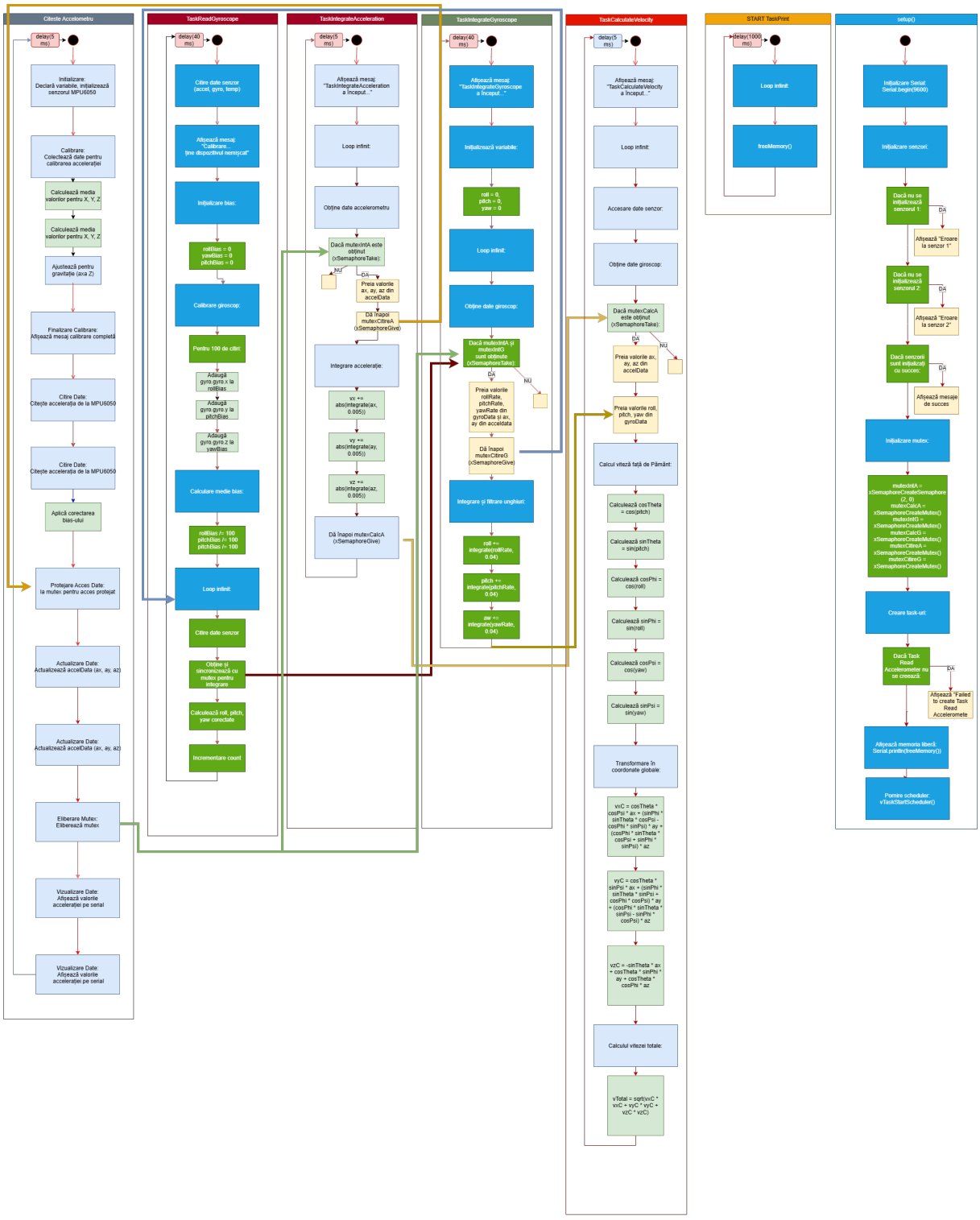


Figura 3.1: Organigrama Task-uri

### Aspecte legate de limbaj

Au fost utilizate urmatoarele biblioteci:

- **FreeRTOS** pentru multitasking
- **Biblioteca Adafruit pentru MPU6050** aceasta abstractizează comunicarea cu senzorii prin I2C
- **semphr.h** gestioneaza sincronizarea prin mutex
- **timers.h** gestioneaza planificarea prin timere.
- **Wire.h** eficientizeaza conexiunea cu placuta si senzorii utilizati.

### Schema circuitului

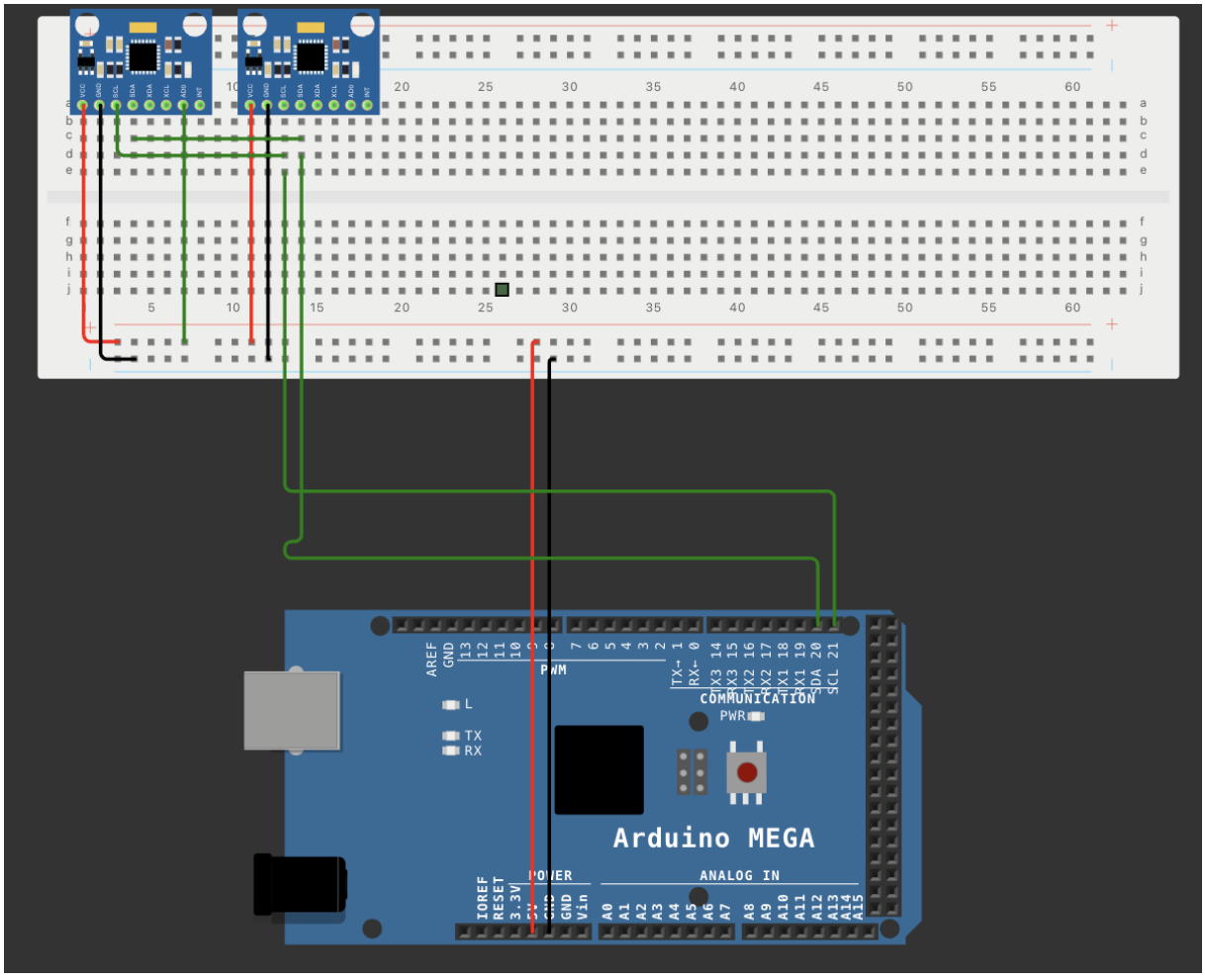


Figura 3.2: Soluţia implementată - circuitul in wokwi

Această schemă reprezintă conexiunea dintre două module MPU-6050 și un Arduino Mega, uti-

lizând o placă de test (breadboard) pentru organizarea circuitului. Modulele MPU-6050 sunt alimentate prin pinii VCC și GND, conectați la sursa de alimentare de pe Arduino Mega. Comunicarea între senzori și Arduino se realizează prin magistrala I2C, folosind pinii SDA (pin 20) și SCL (pin 21) de pe Arduino Mega. Cele două module împart aceeași magistrală, dar pentru a evita conflictele, adresa I2C a fiecărui modul este diferențiată: unul are pinul AD0 conectat la GND (adresa implicită 0x68), iar celălalt la VCC (adresa 0x69). Aceasta permite Arduino să interacționeze cu ambele module simultan. Breadboard-ul facilitează realizarea circuitului, oferind o modalitate ordonată și sigură de conectare a componentelor.

## 3.3 Implementarea soluției

```
#include <Arduino_FreeRTOS.h>
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <Wire.h>
#include <semphr.h>
#include <SimpleTimer.h>

TaskHandle_t Task1;
TaskHandle_t Task2;
TaskHandle_t Task3;
TaskHandle_t Task4;
TaskHandle_t Task5;
SimpleTimer timer;

SemaphoreHandle_t mutexIntA;
SemaphoreHandle_t mutexCalcA;
SemaphoreHandle_t mutexIntG;
SemaphoreHandle_t mutexCalcG;
SemaphoreHandle_t mutexCitireA;
SemaphoreHandle_t mutexCitireG;
// Obiecte pentru senzori MPU-6050
Adafruit_MPU6050 mpu1; // Senzor 1
Adafruit_MPU6050 mpu2; // Senzor 2

// Structuri pentru stocarea datelor senzorilor
typedef struct {
  float ax, ay, az;
} AccelerometerData;

typedef struct {
  float roll, pitch, yaw;
} GyroscopeData;

// Variabile globale protejate
AccelerometerData accelData = { 0, 0, 0 };
GyroscopeData gyroData = { 0, 0, 0 };
float vx = 0, vy = 0, vz = 0;

// Functii de integrare
float integrate(float value, float deltaT) {
  return value * deltaT; // Simpla integrare folosind valoarea curenta si timpul
}
float a = 0, b = 0, c = 0, count = 0;
// Task 1: Citire accelerometru de la senzor 1
void TaskReadAccelerometer(void *pvParameters) {
  (void)pvParameters;
  sensors_event_t accel, gyro, temp;
  mpu1.getEvent(&accel, &gyro, &temp);
  Serial.println("Accelerometer a nceput...");
  float axBias = 0, ayBias = 0, azBias = 0;
  Serial.println("Calibrare... tine dispozitivul nemiscat.");
  // Variabile pentru bias si calibrare
```

```
  const float g = 9.81; // Acceleratia gravitationala
  for (int i = 0; i < 1000; i++) {

    axBias += accel.acceleration.x;
    ayBias += accel.acceleration.y;
    azBias += accel.acceleration.z;
  }
  axBias /= 1000;
  ayBias /= 1000;
  azBias /= 1000 + g; // Adaugam gravitatia (pe axa verticala)

  Serial.println("Calibrare completa.");
  while (1) {

    sensors_event_t accel, gyro, temp;
    mpu1.getEvent(&accel, &gyro, &temp);

    // Protejam accesul la accelData
    if (xSemaphoreTake(mutexCitireA, portMAX_DELAY)) // primeste mutex de la functia
        de integrare, care trimite mutex dupa ce termina de integrat
    {
      accelData.ax = accel.acceleration.x - axBias; //- 1.7096;
      accelData.ay = accel.acceleration.y - ayBias; // 0.191407;
      accelData.az = accel.acceleration.z - azBias; //- 11.2816;

      xSemaphoreGive(mutexIntA);
      xSemaphoreGive(mutexIntA);
    }  // Activeaza functia de integrare aceleratie

    vTaskDelay(pdMS_TO_TICKS(5)); // Ruleaza la fiecare 5 ms
  }
}


float rollBias, yawBias, pitchBias;

// Task 2: Citire giroscop de la senzor 2
void TaskReadGyroscope(void *pvParameters) {
  (void)pvParameters;
  sensors_event_t accel, gyro, temp;
  mpu2.getEvent(&accel, &gyro, &temp);

  Serial.println("Gyroscope a nceput...");
  Serial.println("Calibrare... tine dispozitivul nemiscat.");
  // Variabile pentru bias si calibrare
  rollBias = 0;
  yawBias = 0;
  pitchBias = 0;
  const float g = 9.8; // Acceleratia gravitationala
  for (int i = 0; i < 1000; i++) {

    rollBias += gyro.gyro.x;
    pitchBias += gyro.gyro.y;
    yawBias += gyro.gyro.z;
  }
  rollBias /= 1000;
```

```
  pitchBias /= 1000;
  yawBias /= 1000;

  Serial.println("Calibrare completa.");
  while (1) {

    sensors_event_t accel, gyro, temp;
    mpu2.getEvent(&accel, &gyro, &temp);

    // Calculam unghiurile de ruliu, tangaj si giratie
    if (xSemaphoreTake(mutexCitireG, portMAX_DELAY)) // primeste mutex de la functia
        de integrare, care trimite mutex dupa ce termina de integrat
    {

      gyroData.roll = gyro.gyro.x - rollBias;
      gyroData.pitch = gyro.gyro.y - pitchBias;
      gyroData.yaw = gyro.gyro.z - yawBias;
      xSemaphoreGive(mutexIntG); // Activeaza functia de integrare giroscop
    }

    vTaskDelay(pdMS_TO_TICKS(40)); // Ruleaza la fiecare 40 ms
  }
}

// Parametrii filtrului complementar
const float alpha = 0.98; // Pondere pentru giroscop (ajustabil ntre 0 si 1)
float g = 9.81;
// Variabile pentru filtrul complementar
float pitchFiltered = 0;
float rollFiltered = 0;
float yaw;

// Task 4: Integrare giroscop senzor 2 (cu filtru complementar)
void TaskIntegrateGyroscope(void *pvParameters) {
  (void)pvParameters;

  Serial.println("TaskIntegrateGyroscope a nceput...");

  while (1) {
    float rollRate, pitchRate, yawRate, ax, ay, az, pitch, roll;

    if (xSemaphoreTake(mutexIntG, portMAX_DELAY)) {
      rollRate = gyroData.roll;
      pitchRate = gyroData.pitch;
      yawRate = gyroData.yaw;
      xSemaphoreGive(mutexCitireG); // Activeaza functia de citire gyro

      // Integrare giroscop (angulare)
      float pitchAcc, rollAcc; // Calcul din accelerometru
      if (xSemaphoreTake(mutexIntA, portMAX_DELAY)) {
        ax = accelData.ax;
        ay = accelData.ay;
        az = accelData.az;
      }

      pitch += integrate(pitchRate, 0.04);
```

```
      roll += integrate(rollRate, 0.04);

      // Filtrul complementar
      pitchFiltered = alpha * pitch + (1 - alpha) * ax;
      rollFiltered = alpha * roll + (1 - alpha) * ay;

      // Integrare yaw fara corectii
      yaw += integrate(yawRate, 0.04);
      xSemaphoreGive(mutexCalcG);  // Activeaza functia de adunare acceleratie
      vTaskDelay(pdMS_TO_TICKS(40)); // Ruleaza sincronizat cu citirea giroscopului
    }
  }
}

void TaskIntegrateAcceleration(void *pvParameters) {
  (void)pvParameters;

  Serial.println("TaskIntegrateAcceleration a nceput...");
  while (1) {
    float ax, ay, az;

    if (xSemaphoreTake(mutexIntA, portMAX_DELAY)) // primeste mutex de la functia de
        citire, care trimite mutex dupa ce termina de citit
    {

      ax = accelData.ax;
      ay = accelData.ay;
      az = accelData.az;
      xSemaphoreGive(mutexCitireA); // Activeaza functia de citire acceleratie
      // ax -= sin(pitchFiltered) * g;
      // ay -= sin(rollFiltered) * g;
      // az -= cos(pitchFiltered) * cos(rollFiltered) * g;
      if (abs(ax) < 0.1) ax = 0;
      if (abs(ay) < 0.1) ay = 0;
      if (abs(az) < 0.1) az = 0;

      vx += integrate(ax, 0.005); // Integrare acceleratie
      vy += integrate(ay, 0.005);
      vz += integrate(az, 0.005);
      if (abs(vx) < 0.1) vx = 0;
      if (abs(vy) < 0.1) vy = 0;
      if (abs(vz) < 0.1) vz = 0;


      xSemaphoreGive(mutexCalcA); // Activeaza functia de adunare acceleratie
      //Serial.println("TaskIntegrateAcceleration a terminat...");
      vTaskDelay(pdMS_TO_TICKS(5)); // Ruleaza sincronizat cu citirea accelerometrului
    }
  }
}

float vTotal, vxC, vyC, vzC;
// Task 5: Calculare si afisare viteza fata de Pamnt (corectata)
void TaskCalculateVelocity(void *pvParameters) {
  (void)pvParameters;
  const float g = 9.81;
```

```
  float gint = integrate(g, 0.005);

  Serial.println("TaskCalculateVelocity a nceput...");
  while (1) {
    float ax, ay, az, aroll, ayaw, apitch;

    if (xSemaphoreTake(mutexCalcA, portMAX_DELAY)) {
      ax = vx;
      ay = vy;
      az = vz;
    }
    if (xSemaphoreTake(mutexCalcG, portMAX_DELAY)) {
      aroll = rollFiltered;
      ayaw = yaw;
      apitch = pitchFiltered;
    }

    float cosTheta = cos(aroll);
    float sinTheta = sin(aroll);
    float cosPhi = cos(ayaw);
    float sinPhi = sin(ayaw);
    float cosPsi = cos(apitch);
    float sinPsi = sin(apitch);

    // Transformare n coordonate globale
    vxC = cosTheta * cosPsi * ax + (sinPhi * sinTheta * cosPsi - cosPhi * sinPsi) * ay
        + (cosPhi * sinTheta * cosPsi + sinPhi * sinPsi) * az - (cosPhi * sinTheta *
        cosPsi + sinPhi * sinPsi) * gint;

    vyC = cosTheta * sinPsi * ax + (sinPhi * sinTheta * sinPsi + cosPhi * cosPsi) * ay
        + (cosPhi * sinTheta * sinPsi - sinPhi * cosPsi) * az - (cosPhi * sinTheta *
        sinPsi - sinPhi * cosPsi) * gint;

    vzC = -sinTheta * ax + cosTheta * sinPhi * ay + cosTheta * cosPhi * az - cosPsi *
        cosTheta * gint;

    // Calculul vitezei totale
    vTotal = sqrt(vxC * vxC + vyC * vyC + vzC * vzC);
    timer.run();
    vTaskDelay(pdMS_TO_TICKS(5)); // Ruleaza la fiecare 5 ms
  }
}


void TaskPrint() {
  // Afisare viteze
  Serial.print("Viteza: vx = ");
  Serial.print(vxC, 3);
  Serial.print(", vy = ");
  Serial.print(vyC, 3);
  Serial.print(", vz = ");
  Serial.print(vzC, 3);
  Serial.print(", vTotal = ");
  Serial.println(vTotal, 3);
  Serial.print("Roll = ");
  Serial.print(rollFiltered, 3);
```

```
  Serial.print(", yaw = ");
  Serial.print(yaw, 3);
  Serial.print(", [pitch] = ");
  Serial.println(pitchFiltered, 3);
}
extern unsigned int __heap_start, *__brkval;
int freeMemory() {
  int v;
  return (int)&v - (__brkval == 0 ? (int)&__heap_start : (int)__brkval);
}

void setup() {
  Serial.begin(9600);

  // Mesaje de debug pentru initializarea senzorilor
  if (!mpu1.begin(0x69)) {
    Serial.println("Eroare la senzor 1");
    // Opriti programul daca senzorul 1 nu poate fi initializat
  } else {
    Serial.println("Senzor 1 initializat cu succes");
  }

  if (!mpu2.begin(0x68)) {
    Serial.println("Eroare la senzor 2");
    while (1)
      ;  // Opriti programul daca senzorul 2 nu poate fi initializat
  } else {
    Serial.println("Senzor 2 initializat cu succes");
  }

  // Initializare mutex
  mutexIntA = xSemaphoreCreateCounting(2, 0);
  mutexCalcA = xSemaphoreCreateMutex();
  mutexIntG = xSemaphoreCreateMutex();
  mutexCalcG = xSemaphoreCreateMutex();
  mutexCitireA = xSemaphoreCreateMutex();
  mutexCitireG = xSemaphoreCreateMutex();

  // Creare task-uri
  if (xTaskCreate(TaskReadAccelerometer, "ReadAccel", 512, NULL, 2, &Task1) !=
      pdPASS) {
    Serial.println("Failed to create TaskReadAccelerometer");
  }

  if (xTaskCreate(TaskReadGyroscope, "ReadGyro", 512, NULL, 2, &Task2) != pdPASS) {
    Serial.println("Failed to create TaskReadGyroscope");
  }

  if (xTaskCreate(TaskIntegrateAcceleration, "IntegrateAccel", 512, NULL, 2, &Task3)
      != pdPASS) {
    Serial.println("Failed to create TaskIntegrateAcceleration");
  }

  if (xTaskCreate(TaskIntegrateGyroscope, "IntegrateGyro", 512, NULL, 2, &Task4) !=
      pdPASS) {
    Serial.println("Failed to create TaskIntegrateGyroscope");
```

```
  }

  if (xTaskCreate(TaskCalculateVelocity, "CalculateVelocity", 512, NULL, 2, &Task5)
      != pdPASS) {
    Serial.println("Failed to create TaskCalculateVelocity");
  }
  timer.setInterval(500, TaskPrint);
  Serial.println(freeMemory());

  // Pornire scheduler
  vTaskStartScheduler();
}

void loop() {

  // Bucla principala este goala - rularea este gestionata de FreeRTOS
}
```

# 4 Testarea aplicației și validarea soluției propuse

În cadrul testării aplicației, am recurs la 3 scenarii replicabile pentru a evalua acuratețea și performanța acesteia în diferite condiții. Aceste scenarii au fost concepute astfel încât să acopere atât situații obișnuite, cât și situații limită, relevante pentru scopul aplicației.

- **Scenariul static** - În cadrul acestui scenariu s-a urmărit comportarea sistemului în condiții de stabilitate. În aceste condiții, aplicația afișează viteze 0 și unghiuri apropiate de 0.
- **Scenariul cu variații lente de accelerație și rotație** - În cadrul acestui scenariu s-a urmărit comportarea sistemului în condiții de accelerație și rotație lente, cum ar fi zborul rectiliniu al unui avion. În acest scenariu se observă o calculare și afișare stabilă a vitezei, conform mișcării imprimate, și apoi se stabilizează, cu mici erori. La imprimarea unei decelerări, viteza afișată scade încet. De asemenea, este nevoie de o mișcare bruscă pentru a începe calculul vitezelor. Acest fapt poate fi datorat metodei de calibrare ce prezintă erori și nu este precisă în timp.
- **Scenariul cu variații bruște de accelerație și rotație** - În cadrul acestui scenariu s-a urmărit comportarea sistemului în condiții de accelerație și rotație, precum în cazul decolării, accelerării sau alte manevre acrobatice efectuate în special de avioanele militare. se observă că datele furnizate de giroscop și apoi integrate prezintă erori grosiere, de tipul $\theta \geq 3\pi$, ceea ce nu este posibil, întrucât accelerația unghiulară furnizată de giroscop se găsește în intervalul $[-\pi, \pi]$. De asemenea, vitezele pe diferite axe variază foarte mult, inclusiv semnele, însă acest lucru poate fi datorat de asemenea integrării ce prezintă erori, și apoi funcțiilor trigonometrice ce își schimbă semnele și care au un impact direct asupra calculului vitezelor.

În urma testării aplicatiei, se pot observa câteva concluzii importante cu privire la comportamentul sistemului în cele trei scenarii. În scenariul static, unde sistemul operează în condiții de stabilitate cu viteze și unghiuri aproape de 0, performanța acestuia este adecvată, iar comportamentul este previzibil și corect. Sistemul răspunde bine în aceste condiții stabile, fără fluctuații semnificative, ceea ce sugerează că funcționează eficient în medii stabile.

În scenariul cu variații lente de accelerare și rotație, cum ar fi zborul rectiliniu al unui avion, sistemul afișează o calculare și o afișare stabilă a vitezei, chiar dacă se observă mici erori care se stabilizează treptat. Aceste erori pot fi atribuite metodei de calibrare, care nu este suficient de precisă pe termen lung. În plus, în momentul în care se imprimă o decelerare, viteza scade încet, iar sistemul are o reacție mai lentă la schimbările de viteză, ceea ce sugerează că ar fi necesară o îmbunătățire a procesării acestor variabile.

În ceea ce privește scenariul cu variații bruște de accelerare și rotație, specifice manevrelor de decolare sau acrobațiilor avioanelor militare, se observă erori semnificative, cum ar fi unghiuri de

rotație care depășesc intervalul permis de giroscop. Aceste erori indică o problemă cu integrarea datelor de la giroscop și cu utilizarea funcțiilor trigonometrice pentru calcularea unghiurilor. În acest caz, sistemul prezintă variații nejustificate ale semnelor vitezelor pe diferite axe, ceea ce sugerează erori atât la nivelul semnalelor de intrare, cât și la nivelul algoritmilor de integrare și trigonometrie. Aceste erori pot fi corectate prin îmbunătățirea acestor algoritmi, pentru a fi mai rezilienți la schimbările bruște de viteză și rotație.

În ansamblu, sistemul funcționează bine în condiții de stabilitate și cu variații lente, dar are nevoie de ajustări semnificative pentru a face față condițiilor extreme, unde erorile devin mai evidente.

```
Viteza: vx = 0.000, vy = -0.000, vz = -0.049, vTotal = 0.049
Roll = -0.002, yaw = -0.000, [pitch] = -0.001
Viteza: vx = 0.000, vy = -0.000, vz = -0.049, vTotal = 0.049
Roll = -0.002, yaw = -0.001, [pitch] = -0.000
Viteza: vx = 0.000, vy = -0.000, vz = -0.049, vTotal = 0.049
Roll = -0.004, yaw = -0.000, [pitch] = -0.001
Viteza: vx = 0.000, vy = -0.000, vz = -0.049, vTotal = 0.049
Roll = -0.003, yaw = -0.000, [pitch] = -0.000
Viteza: vx = 0.000, vy = -0.000, vz = -0.049, vTotal = 0.049
Roll = -0.005, yaw = -0.000, [pitch] = 0.002
Viteza: vx = 0.000, vy = 0.000, vz = -0.049, vTotal = 0.049
Roll = -0.006, yaw = 0.000, [pitch] = 0.002
Viteza: vx = 0.000, vy = 0.000, vz = -0.049, vTotal = 0.049
Roll = -0.007, yaw = 0.000, [pitch] = 0.004
Viteza: vx = 0.000, vy = 0.000, vz = -0.049, vTotal = 0.049
Roll = -0.008, yaw = 0.000, [pitch] = 0.004
Viteza: vx = 0.000, vy = 0.000, vz = -0.049, vTotal = 0.049
Roll = -0.009, yaw = 0.001, [pitch] = 0.005
Viteza: vx = 0.001, vy = 0.000, vz = -0.049, vTotal = 0.049
Roll = -0.010, yaw = 0.000, [pitch] = 0.006
Viteza: vx = 0.001, vy = 0.000, vz = -0.049, vTotal = 0.049
```

Figura 4.1: Scenariu 1 - output

```
Viteza: vx = 0.019, vy = -0.025, vz = -0.044, vTotal = 0.054
Roll = -0.459, yaw = -0.538, [pitch] = 0.018
Viteza: vx = 0.023, vy = -0.017, vz = -0.043, vTotal = 0.051
Roll = -0.497, yaw = -0.372, [pitch] = 0.030
Viteza: vx = -0.005, vy = -0.028, vz = -0.042, vTotal = 0.051
Roll = 0.373, yaw = -0.507, [pitch] = 0.363
Viteza: vx = 0.022, vy = -0.015, vz = -0.044, vTotal = 0.051
Roll = -0.463, yaw = -0.326, [pitch] = 0.036
Viteza: vx = 0.017, vy = -0.014, vz = -0.046, vTotal = 0.051
Roll = -0.377, yaw = -0.295, [pitch] = -0.013
Viteza: vx = 0.016, vy = -0.014, vz = -0.046, vTotal = 0.051
Roll = -0.350, yaw = -0.276, [pitch] = -0.033
Viteza: vx = 0.017, vy = -0.039, vz = -0.039, vTotal = 0.058
Roll = -0.651, yaw = -0.910, [pitch] = -0.068
Viteza: vx = 6.845, vy = -5.653, vz = 4.981, vTotal = 10.179
Roll = -0.166, yaw = 0.877, [pitch] = -0.173
Viteza: vx = 7.273, vy = -1.259, vz = 0.964, vTotal = 7.444
Roll = -0.136, yaw = 0.875, [pitch] = -0.177
Viteza: vx = 6.338, vy = -1.116, vz = 0.639, vTotal = 6.467
Roll = -0.106, yaw = 0.873, [pitch] = -0.180
Viteza: vx = 5.398, vy = -0.965, vz = 0.371, vTotal = 5.496
Roll = -0.076, yaw = 0.872, [pitch] = -0.184
Viteza: vx = 4.450, vy = -0.805, vz = 0.161, vTotal = 4.525
Roll = -0.046, yaw = 0.870, [pitch] = -0.188
Viteza: vx = 3.490, vy = -0.636, vz = 0.010, vTotal = 3.547
Roll = -0.016, yaw = 0.868, [pitch] = -0.191
Viteza: vx = 2.527, vy = -0.460, vz = -0.083, vTotal = 2.570
Roll = 0.014, yaw = 0.866, [pitch] = -0.195
Viteza: vx = 1.571, vy = -0.277, vz = -0.118, vTotal = 1.600
Roll = 0.044, yaw = 0.864, [pitch] = -0.198
Viteza: vx = 0.611, vy = -0.087, vz = -0.094, vTotal = 0.624
Roll = 0.074, yaw = 0.862, [pitch] = -0.202
Viteza: vx = 0.004, vy = 0.037, vz = -0.048, vTotal = 0.061
Roll = 0.104, yaw = 0.861, [pitch] = -0.206
Viteza: vx = 0.004, vy = 0.037, vz = -0.048, vTotal = 0.060
Roll = 0.134, yaw = 0.859, [pitch] = -0.209
```

Figura 4.2: Scenariu 2 - output

```
Viteza: vx = 2.801, vy = -2.412, vz = -5.636, vTotal = 6.740
Roll = 0.984, yaw = -0.411, [pitch] = -0.837
Viteza: vx = 5.569, vy = 8.442, vz = -2.668, vTotal = 10.460
Roll = 2.499, yaw = -0.857, [pitch] = -2.603
Viteza: vx = 4.636, vy = 18.660, vz = 6.194, vTotal = 20.201
Roll = 2.690, yaw = -0.900, [pitch] = -2.626
Viteza: vx = -2.043, vy = 29.328, vz = 12.722, vTotal = 32.033
Roll = 2.727, yaw = -0.955, [pitch] = -2.509
Viteza: vx = -11.216, vy = 37.912, vz = 19.898, vTotal = 44.261
Roll = 2.790, yaw = -0.976, [pitch] = -2.431
Viteza: vx = -22.493, vy = 44.689, vz = 26.354, vTotal = 56.547
Roll = 2.802, yaw = -0.986, [pitch] = -2.295
Viteza: vx = -35.889, vy = 49.032, vz = 32.475, vTotal = 68.897
Roll = 2.797, yaw = -0.992, [pitch] = -2.163
Viteza: vx = -50.880, vy = 50.505, vz = 38.075, vTotal = 81.174
Roll = 2.799, yaw = -1.006, [pitch] = -2.034
Viteza: vx = -65.817, vy = 51.062, vz = 43.315, vTotal = 93.890
Roll = 2.810, yaw = -1.025, [pitch] = -1.941
Viteza: vx = -82.965, vy = 45.457, vz = 49.134, vTotal = 106.600
Roll = 2.830, yaw = -1.037, [pitch] = -1.799
Viteza: vx = -105.439, vy = 14.696, vz = 54.076, vTotal = 119.405
Roll = 2.898, yaw = -1.064, [pitch] = -1.494
Viteza: vx = -118.608, vy = 3.019, vz = 58.814, vTotal = 132.423
Roll = 2.930, yaw = -1.081, [pitch] = -1.396
Viteza: vx = -129.684, vy = -13.963, vz = 64.094, vTotal = 145.331
Roll = 2.945, yaw = -1.087, [pitch] = -1.280
Viteza: vx = -138.306, vy = -32.394, vz = 69.127, vTotal = 157.976
Roll = 2.955, yaw = -1.094, [pitch] = -1.157
Viteza: vx = -144.162, vy = -54.987, vz = 73.372, vTotal = 170.850
Roll = 2.967, yaw = -1.104, [pitch] = -1.034
Viteza: vx = -148.041, vy = -76.731, vz = 77.570, vTotal = 183.905
Roll = 2.994, yaw = -1.120, [pitch] = -0.947
Viteza: vx = -152.364, vy = -95.459, vz = 80.064, vTotal = 196.818
Roll = 3.034, yaw = -1.139, [pitch] = -0.873
```

Figura 4.3: Scenariu 3 - output

# Bibliografie

[1] Arizona State University. *Lecture 9 - Aircraft Dynamics*. Disponibil la: https://control.asu.edu/Classes/MMAE441/Aircraft/441Lecture9.pdf.

[2] Smithsonian National Air and Space Museum. *Roll, Pitch, and Yaw*. Disponibil la: https://howthingsfly.si.edu/flight-dynamics/roll-pitch-and-yaw.

[3] Hibit. *Complementary Filter and Relative Orientation with MPU6050*. Disponibil la: https://www.hibit.dev/posts/92/complementary-filter-and-relative-orientation-with-mpu6050.

[4] Invensense. *MPU-6000 Datasheet*. Disponibil la: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf.