**Proiect Măsurători și Traductoare**

**Sistem de suspensie dinamic**

Dradici Leon

Anul II

Grupa 7

Facultatea de Automatică și Calculatoare

Secția Automatică și Informatică Aplicată

**Contact: 0741156469**

**Cuprins**

**1** **Introducere**  
1.1 Contextul proiectului  
1.2 Obiectivele proiectului

**2** **Descrierea realizării practice**  
2.1 Schema electrică  
2.2 Conexiuni  
2.3 Hardware utilizat  
2.3.1 ESP32 (ESP-WROOM-32)  
2.3.2 MPU 6050  
2.3.3 Servomotoare MG90 cu dinți de metal  
2.3.4 Alte componente  
2.4 Software utilizat  
2.4.1 Platforma de programare  
2.4.2 Biblioteci software utilizate  
2.4.3 Algoritmi implementați  
2.4.3.1 Achiziția datelor de la MPU 6050  
2.4.3.2 Filtrarea datelor de la MPU 6050  
2.4.3.3 Controlul poziției servomotoarelor  
2.4.3.4 Tranziția lină a pozițiilor servomotoarelor  
2.4.3.5 Scrierea pozițiilor către servomotoare

**3** **Asamblarea mecanismelor**  
3.1 Prototiparea suspensiilor  
3.2 Construirea șasiului și integrarea suspensiilor  
3.3 Testarea inițială și identificarea problemelor  
3.4 Ajustări și îmbunătățiri  
3.5 Montarea finală

**4** **Rezultate**

**5** **Concluzii**

**6** **Bibliografie**

**7 Anexe**

**Abrevieri si simboluri**

**Abrevieri**

* **ESP32**: Microcontroler dezvoltat de Espressif Systems
* **MPU 6050**: Senzor MEMS care include giroscop și accelerometru
* **PWM**: Pulse Width Modulation (Modulație în lățime de impuls)
* **I2C**: Inter-Integrated Circuit, protocol de comunicare serială
* **BLE**: Bluetooth Low Energy
* **Servo Horn**: Piesa din plastic care se ataseaza la servomotor

**Simboluri și termeni tehnici**

* **GND**: Ground (masă electrică)
* **VIN**: Voltage Input (alimentare)
* **SDA**: Serial Data (linie de date pentru I2C)
* **SCL**: Serial Clock (linie de ceas pentru I2C)
* **X, Y, Z**: Axele de mișcare ale senzorului MPU 6050
* **°/s**: Grade pe secundă, unitate de măsură pentru viteza unghiulară
* **g**: Accelerație gravitațională (aproximativ 9.81 m/s²)
* **PWM signal**: Semnal folosit pentru controlul servomotoarelor

**1. Introducere**

**1.1. Contextul proiectului**

Într-o lume în continuă dezvoltare tehnologică, sistemele de suspensii electrice joacă un rol esențial în diverse domenii, cum ar fi industria auto, robotică, aplicații de recunoaștere și salvare sau chiar vehicule off-road. Aceste sisteme oferă numeroase avantaje, printre care stabilitatea îmbunătățită, adaptabilitatea la terenuri dificile și eficiența energetică sporită.

Proiectul de față constă în realizarea unui prototip de mașină mică, echipată exclusiv cu un sistem de suspensii electrice. Acest prototip reprezintă o platformă modulară, care poate fi extinsă și adaptată pentru utilizarea în vehicule mai mari, roboți autonomi sau alte aplicații specializate. De asemenea, el demonstrează potențialul implementării unui astfel de sistem în vehicule electrice moderne sau în echipamente destinate utilizării în condiții extreme precum salvarea persoanelor din zone greu accesibile.

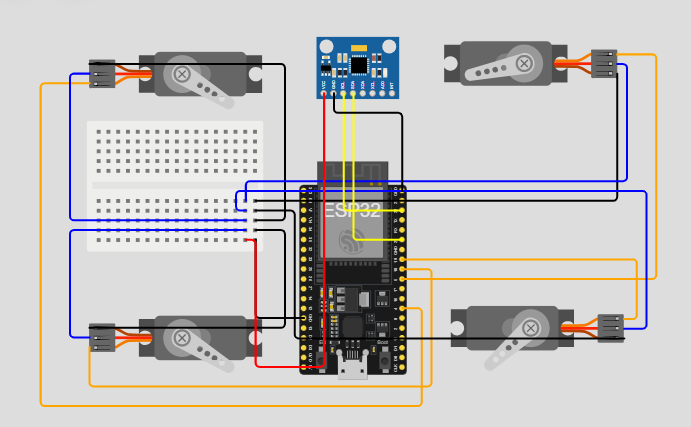
**1.2. Obiectivele proiectului**

Obiectivul principal al acestui proiect este proiectarea și implementarea unui sistem funcțional de suspensii electrice bazat pe un microcontroler ESP32, utilizând senzori MPU 6050 și servomotoare pentru controlul poziției. Mai specific, se urmărește:

* Construirea unui prototip care să demonstreze principiul de funcționare al suspensiilor electrice.
* Integrarea unui sistem de control precis, capabil să răspundă rapid la variațiile de teren.
* Testarea și evaluarea performanței sistemului în scenarii practice, relevante pentru aplicațiile reale.

**2. Descrierea realizarii practice**

**2.1. Schema electrica**

****

**2.2. Conectiuni :**

Servomotoarele sunt denumite: FL – Front Left conectat la pinul D4

FR – Front Right conectat la pinul D5

RL – Rear Leftconectat la pinul D18

RR – Rear Rightconectat la pinul D19

Pentru fiecare dintre acestea intrarile pentru alimentare si GND au fost conectate pe un breadboard unde fac legatura cu pinii de GND si VIN de la microcontrolerul ESP32.

Pentru MPU6050 legaturile la GND respectiv VIN sunt realizate la fel ca si la servomotoare. Pinii D21 si D22 au fost utilizati pentru conectiunea la SDA si SCL care permit transmiterea de informatii intre ESP32 si MPU6050.

Alimentarea se face prin cablu Micro USB pentru a putea fi vazuta transmiterea de date si schimbarea lor in timp real.

**2.3. Hardware utilizat**

#### ****2.3.1. ESP32 (ESP-WROOM-32)****

ESP32 este un microcontroler extrem de versatil și performant, dezvoltat de Espressif Systems, ideal pentru aplicații IoT și embedded. Modelul ESP-WROOM-32 utilizat în acest proiect oferă următoarele caracteristici principale:

* **Procesor**:
  + Dual-core Tensilica Xtensa LX6, cu o frecvență de până la 240 MHz.
  + Capacitatea de procesare permite execuția rapidă a algoritmilor de control și achiziție de date.
* **Memorie**:
  + Memorie RAM de 520 KB, potrivită pentru stocarea temporară a datelor achiziționate.
  + Memorie flash integrată de 4 MB pentru stocarea codului și configurărilor.
* **Conectivitate**:
  + Wi-Fi și Bluetooth Dual Mode (BLE și Classic), utile pentru control și monitorizare de la distanță.
* **GPIO-uri (General Purpose Input/Output)**:
  + Dispune de 34 de pini GPIO, dintre care mulți sunt multifuncționali, suportând PWM, I2C, SPI, UART și ADC.
  + Pinii PWM sunt folosiți pentru controlul servomotoarelor.
  + Interfața I2C este utilizată pentru comunicarea cu MPU 6050.

#### ****2.3.2. MPU 6050****

MPU 6050 este un senzor MEMS integrat care combină un giroscop cu trei axe și un accelerometru cu trei axe. Este utilizat pentru a detecta mișcarea și orientarea platformei, fiind esențial în sistemul de suspensii pentru stabilizarea automată.

* **Principiul de funcționare**:
  + **Accelerometru**: Măsoară accelerațiile pe cele trei axe (X, Y, Z), permițând determinarea poziției relative și a unghiurilor de înclinare.
  + **Giroscop**: Detectează viteza unghiulară pe cele trei axe, oferind informații despre rotație și oscilație.
* **Comunicare**:
  + Utilizează protocolul I2C pentru transferul datelor către ESP32.
  + Adresa I2C implicită este 0x68.
* **Alte caracteristici**:
  + Interval de măsurare configurabil pentru accelerometru (±2g, ±4g, ±8g, ±16g).
  + Interval de măsurare configurabil pentru giroscop (±250°/s, ±500°/s, ±1000°/s, ±2000°/s).

#### ****2.3.3. Servomotoare MG90 cu dinți de metal****

Având dinți de metal pentru o rezistență crescută, ceea ce le face ideale pentru aplicații cu încărcări mecanice sporite, precum cele din suspensiile electrice.

* **Caracteristici principale**:
  + Cuplu maxim: ~2.5 kg·cm la 5V, suficient pentru controlul precis al suspensiilor prototipului.
  + Unghi de rotație: 0° - 180°, permițând ajustări fine ale poziției suspensiilor.
  + Timp de răspuns: Aproximativ 0.1s/60° la 5V.
* **Rol în sistem**:
  + Servomotoarele sunt folosite pentru a modifica poziția suspensiilor, adaptându-se la schimbările terenului detectate de MPU 6050.
  + Controlate prin semnale PWM generate de ESP32.

#### ****2.3.4. Alte componente****

* **Fire de conexiune**:
  + Asigură conexiunile electrice între ESP32, MPU 6050, servomotoare și sursa de alimentare.
  + Firele sunt de calitate superioară pentru a preveni pierderile de tensiune și problemele de contact.
* **Breadboard**:
  + Utilizat pentru realizarea rapidă și flexibilă a conexiunilor în timpul dezvoltării prototipului.
* **Sursa de alimentare**:
  + Oferă alimentare stabilă și suficientă pentru toate componentele.
  + Tensiunea de 5V este utilizată pentru ESP32, MPU 6050 și servomotoare, fiind esențială pentru funcționarea corectă a întregului sistem.

Această selecție de componente a fost realizată pentru a asigura un echilibru optim între performanță, fiabilitate și cost, contribuind astfel la succesul proiectului.

**2.4. Software utilizat**

#### ****2.4.1. Platforma de programare****

Pentru dezvoltarea software-ului proiectului, a fost utilizată platforma **Arduino IDE**, datorită interfeței sale intuitive, suportului extins pentru microcontrolere și a bibliotecilor preexistente pentru componentele hardware utilizate. Arduino IDE oferă compatibilitate cu ESP32 prin instalarea extensiei corespunzătoare, permițând programarea și monitorizarea ușoară a dispozitivului.

#### ****2.4.2. Biblioteci software utilizate****

1. **Wire.h**:
   * Biblioteca standard I2C utilizată pentru comunicarea cu senzorul MPU 6050.
   * Facilitează transferul de date între microcontroler și senzor prin magistrala I2C.
2. **ESP32Servo.h**:
   * Oferă suport pentru controlul servomotoarelor SG90 pe ESP32.
   * Permite generarea semnalelor PWM pentru poziționarea precisă a servomotoarelor.

**2.4.3. Algoritmi implementați**

#### ****2.4.3.1. Achiziția datelor de la MPU 6050****

MPU 6050 furnizează datele accelerometrului prin magistrala I2C. Datele sunt citite, interpretate și convertite în valori utilizabile.

void readMpu6050Data() {

Wire.beginTransmission(0x68); // Adresa MPU-6050

Wire.write(0x3B); // Registrul de început pentru datele accelerometrului

Wire.endTransmission(false);

Wire.requestFrom(0x68, 6, true); // Cerere pentru 6 octeți de date (X, Y, Z)

int16\_t accelX = (Wire.read() << 8 | Wire.read()); // Date pe axa X

int16\_t accelY = (Wire.read() << 8 | Wire.read()); // Date pe axa Y

int16\_t accelZ = (Wire.read() << 8 | Wire.read()); // Date pe axa Z

along = accelX / 16384.0; // Conversie în unități de "g"

across = accelY / 16384.0;

vertical = accelZ / 16384.0;

#ifdef DEBUG

Serial.print("Accel X: "); Serial.print(along);

Serial.print(" Y: "); Serial.print(across);

Serial.print(" Z: "); Serial.println(vertical);

#endif

}

* Citirea datelor de la MPU-6050 folosind **Wire**.
* Conversia valorilor brute în accelerație (unități de "g").
* **Serial.print** este utilizat pentru debug, oferind o vizualizare a valorilor achiziționate.

#### ****2.4.3.2. Filtrarea datelor de la MPU 6050****

Pentru a elimina zgomotul, se aplică un filtru trece-jos (low-pass).

filteredAlong = (smoothingFactor \* along) + ((1 - smoothingFactor) \* filteredAlong);

filteredAcross = (smoothingFactor \* across) + ((1 - smoothingFactor) \* filteredAcross);

filteredVertical = (smoothingFactor \* vertical) + ((1 - smoothingFactor) \* filteredVertical);

* **smoothingFactor** controlează cât de repede se adaptează valorile filtrate la cele noi.
* Valorile brute sunt ponderate cu cele anterioare pentru o tranziție lină.

#### ****2.4.3.3. Controlul poziției servomotoarelor****

Pozițiile servomotoarelor sunt calculate pe baza valorilor filtrate și normalizate.

int targetFL = constrain(neutralFL + normalizedAlong \* maxServoRange \* sensitivityFactor - normalizedAcross \* maxServoRange \* sensitivityFactor -

normalizedVertical \* maxServoRange \* sensitivityFactor, 0, 180);

* Poziția țintă pentru fiecare servo este calculată pe baza valorilor normalizate ale MPU.
* **neutralFL** reprezintă poziția neutră a servomotorului (când nu există înclinare).
* **maxServoRange** și **sensitivityFactor** ajustează sensibilitatea și deplasarea maximă.
* **constrain** asigură ca pozițiile să fie între 0 și 180 de grade.

#### ****2.4.3.4. Tranziția lină a pozițiilor servomotoarelor****

Pentru a preveni mișcările bruște, pozițiile curente sunt interpolate către pozițiile țintă.

int moveTowards(int current, int target, int step) {

if (current < target) {

return min(current + step, target);

} else if (current > target) {

return max(current - step, target);

} else {

return current;

}

}

* **moveTowards** apropie poziția curentă de cea țintă în pași mici, controlați de **step**.
* Asigură tranziții netede între pozițiile succesive.

#### ****2.4.3.5. Scrierea pozițiilor către servomotoare****

Pozițiile calculate sunt trimise către servomotoare.

servoFL.write(currentServoFL);

servoFR.write(currentServoFR);

servoRL.write(currentServoRL);

servoRR.write(currentServoRR);

* **servo.write** setează unghiul pentru fiecare servomotor.
* Pozițiile sunt actualizate gradual, folosind valorile calculate și interpolate anterior.

Acest flux de algoritmi asigură că servomotoarele reacționează lin și precis la modificările terenului detectate de senzorul MPU 6050.

1. **Asamblarea Mecanismelor(Pentru fiecare modificare majora am schimbat culoarea piesei)**

**3.1 Prototiparea suspensiilor**

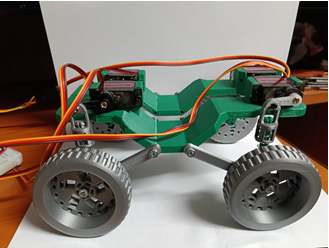
Întreaga idee a început cu proiectarea suspensiilor. Înainte de a începe construcția șasiului și a celorlalte componente, am considerat că ar fi o idee bună să absorb șocurile vehiculului prin suspensii cu arc, care să facă legătura între brațele servomotoarelor și roți.

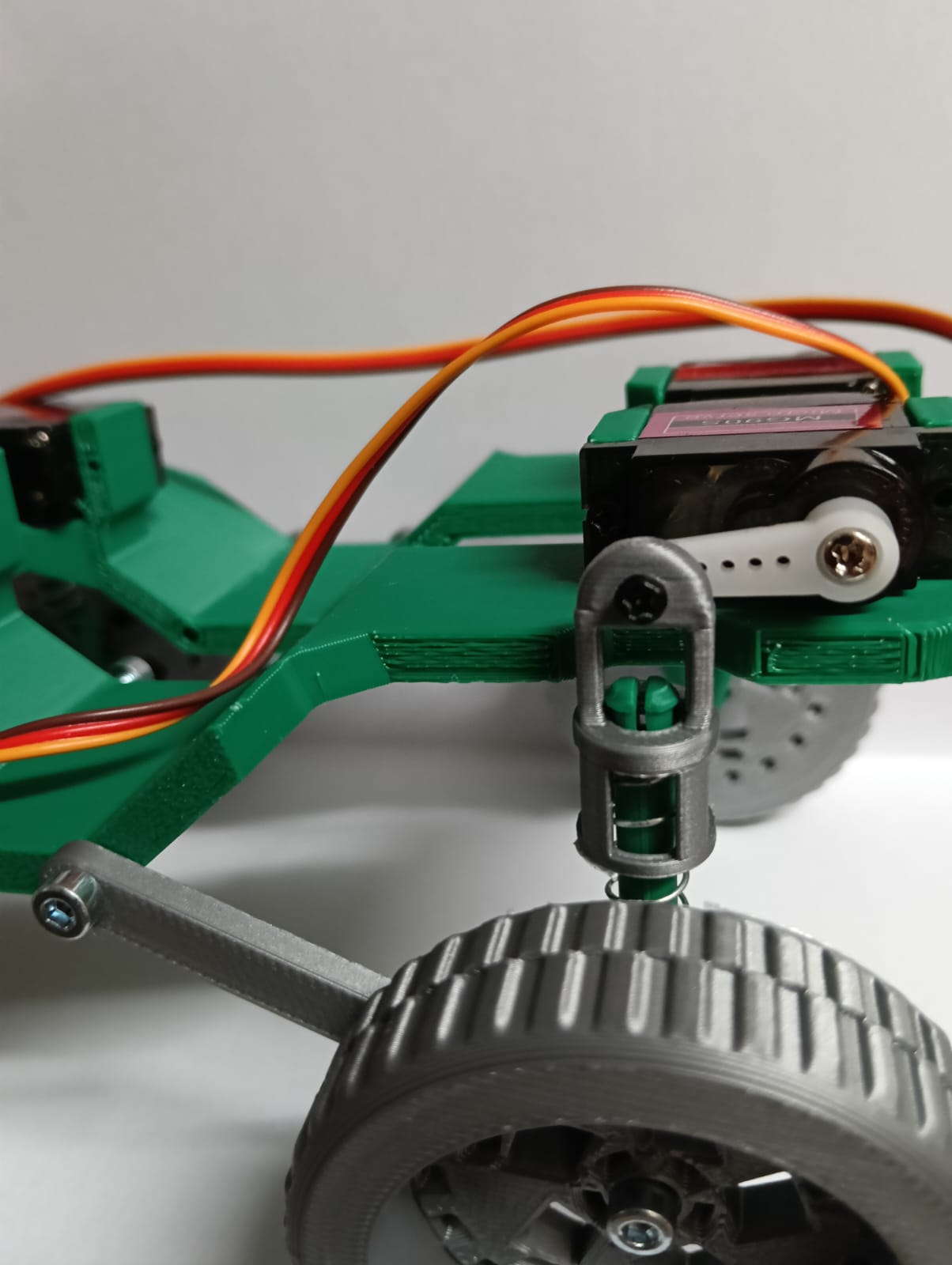
****Am început prin realizarea unor prototipuri, modelate in Onshape si printate 3D, pentru a verifica dacă ideea este realizabilă.

De la dreapta spre stânga, primul model a fost creat doar ca un concept. Ulterior, am hotărât să îl micșorez pentru a economisi spațiu, iar în cele din urmă am stilizat designul. Am ajuns la varianta finală după ce am rezolvat o problemă legată de faptul că arcul se agăța în partea inferioară a componentei glisante.

**3.2 Construirea șasiului și integrarea suspensiilor**

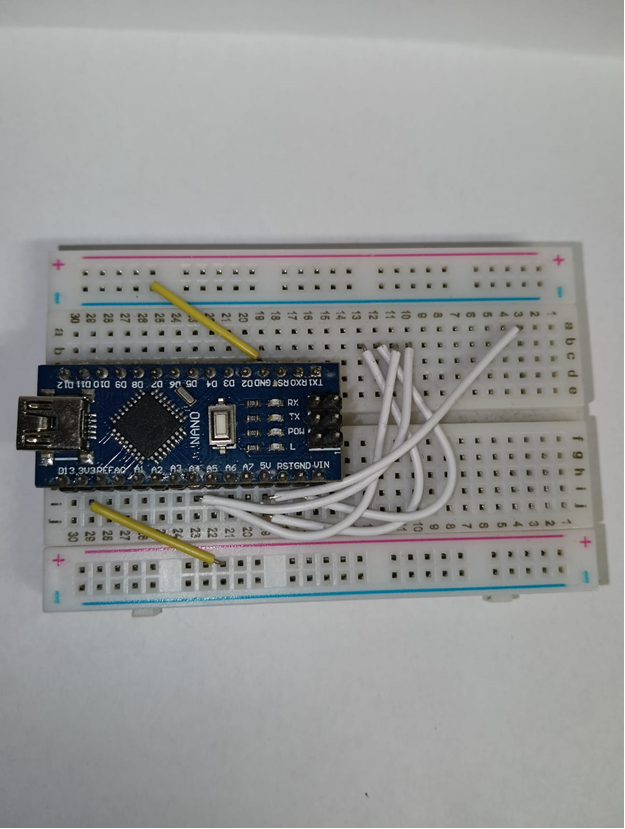
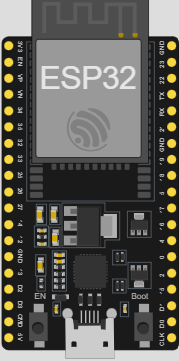
În continuare, înainte de a integra această variantă în întregul sistem, am realizat șasiul, pe care am montat servomotoarele, roțile și patru brațe, câte două pentru fiecare pereche de roți, care să le stabilizeze și să împiedice mișcările necorespunzătoare.



De asemenea, am adaptat suspensiile pentru acest prim sistem. Am adăugat o extensie în partea superioară a suspensiilor pentru a permite fixarea cu un șurub de brațele servomotoarelor. În partea inferioară, am utilizat un singur șurub pentru a fixa roata, brațul și a doua componentă a suspensiei.

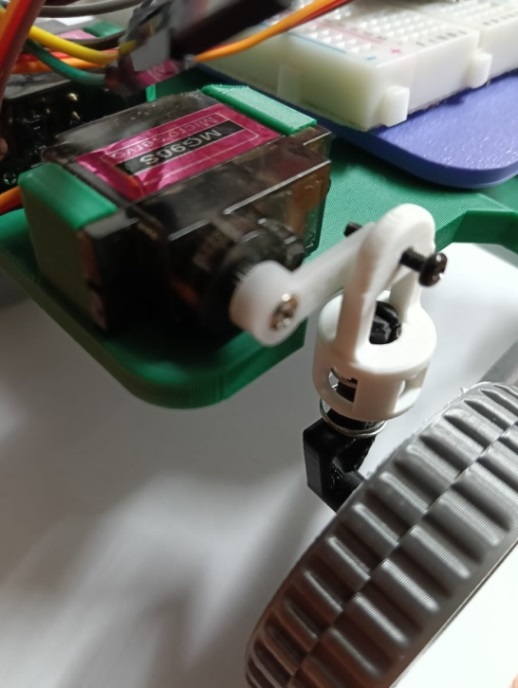
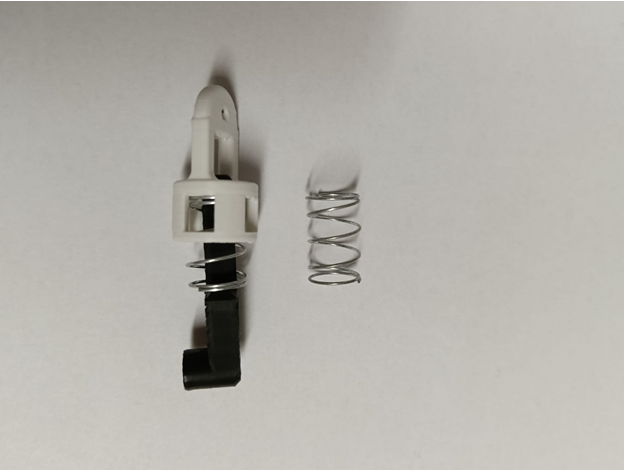
**3.3 Testarea inițială și identificarea problemelor**

Ulterior, am montat circuitul inițial, care conținea un Arduino Nano conectat la cele patru servomotoare și un senzor MPU6050. După ce am scris prima versiune a codului și am efectuat câteva teste, în care modulul MPU6050 nu era montat pe mecanism, am observat mai multe probleme.



Prima problemă, cea mai evidentă, era că servomotoarele vibraseră haotic, chiar și atunci când nu se mișcau. Soluția a fost să trec de la Arduino Nano la un microcontroler mai puternic, precum ESP32.

A doua problemă era de natură software: am constatat că era necesară amplificarea mișcărilor servomotoarelor, deoarece acestea se deplasau foarte puțin în raport cu mișcările senzorului MPU. De asemenea, a fost necesară filtrarea semnalului de intrare și implementarea unei viteze reglabile de tranziție pentru a evita situațiile în care servomotoarele ajustau excesiv poziția vehiculului, determinând senzorul să interpreteze că este mereu în dezechilibru.

Ultima problemă a acestui design inițial era rigiditatea excesivă a arcurilor suspensiilor, deoarece nu am luat în calcul greutatea redusă pe care acestea trebuiau să o susțină. Am modificat designul suspensiilor pentru a putea acomoda arcuri mai flexibile și le-am reasamblat pe șasiu.

Toate aceste ajustări au condus la soluționarea problemelor inițiale și la identificarea altor aspecte care afectau funcționalitatea sistemului.

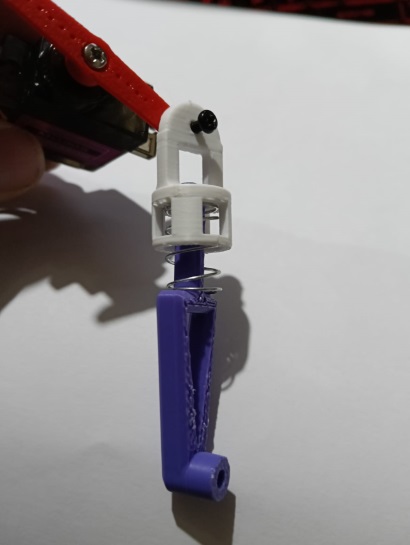
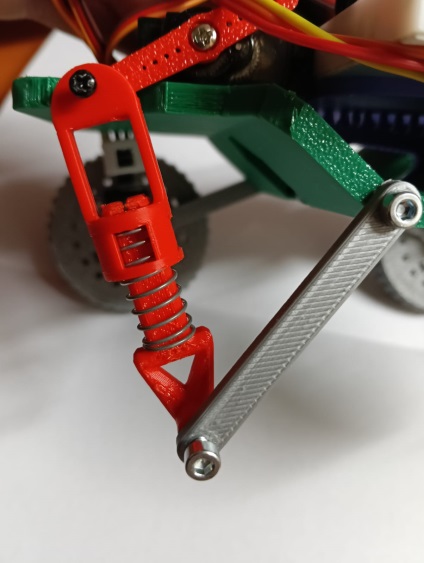
**3.4 Ajustări și îmbunătățiri**

 După ce am amplasat temporar tot circuitul pe mașină și am mai efectuat câteva teste, am observat că nu eram mulțumit de cât de mult se mișcau suspensiile în raport cu modulul MPU6050.

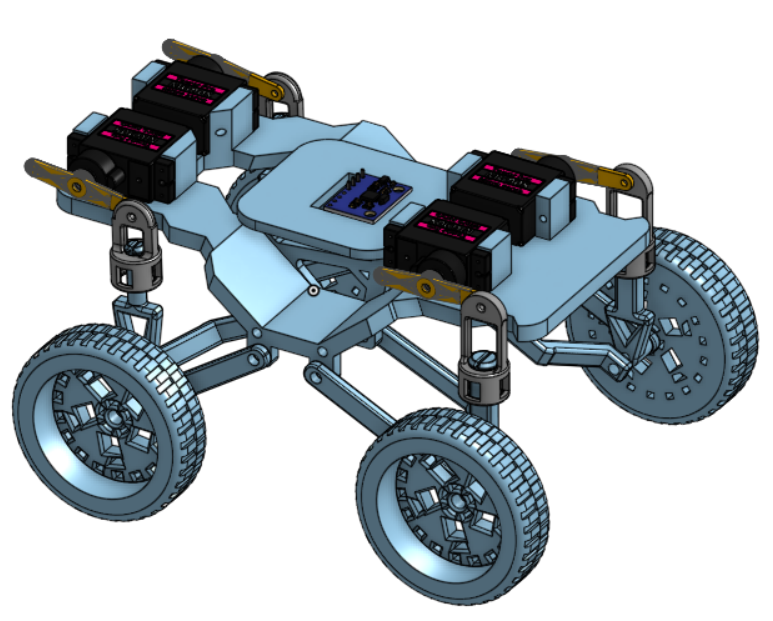


Din software, această problemă nu putea fi rezolvată, așa că următoarea soluție a fost să refac de la zero brațele servomotoarelor (servo horn-urile) care veneau inițial cu servourile. Am creat un nou design și le-am printat 3D, prelungindu-le. Această modificare a funcționat perfect.

După această schimbare, am constatat că suspensiile erau puțin prea scurte pentru întregul sistem, iar arcurile aveau o mișcare insuficientă. Am decis să utilizez un arc mai mare. Totuși, acest arc s-a dovedit a fi mai rigid decât cel utilizat anterior, lucru care a generat noi dificultăți. Am rezolvat această problemă prin înlocuirea arcului rigid cu două arcuri mai flexibile, utilizate pe același nou model de suspensie.

Noile modele de suspensie (colorate în roșu) au fost montate în partea din față a mașinii. În partea din spate, am prelungit partea inferioară a suspensiei, păstrând aceeași configurație veche a arcului, deoarece șocurile din spate erau mai mici decât cele din față.

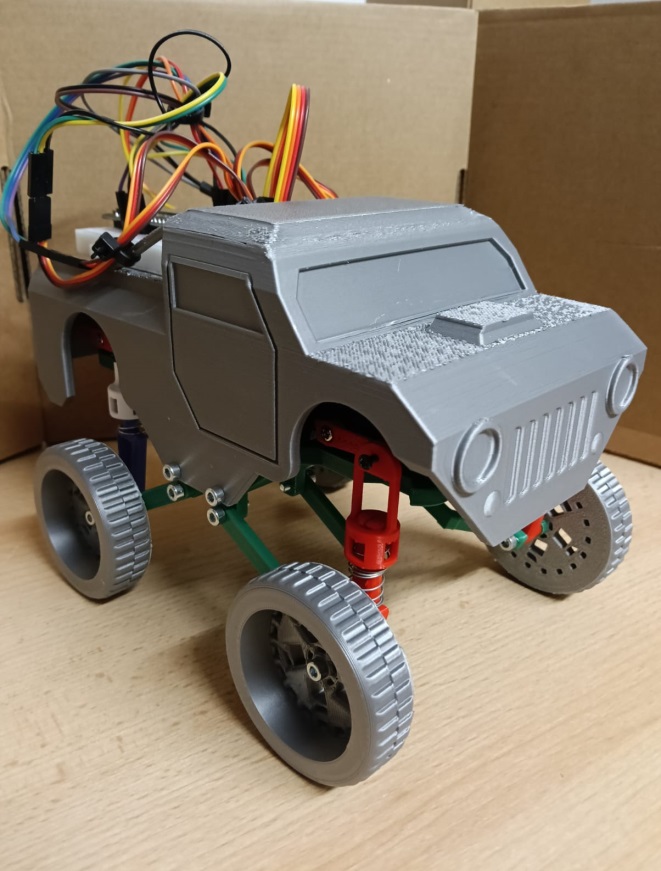
După multiple teste ale noilor implementări, brațele care susțineau suspensiile și roțile au început să cedeze. Am proiectat un nou set de brațe, pe care le-am montat în interiorul șasiului. Totodată, am ajustat lungimile acestor brațe pentru a susține noile dimensiuni ale suspensiilor.

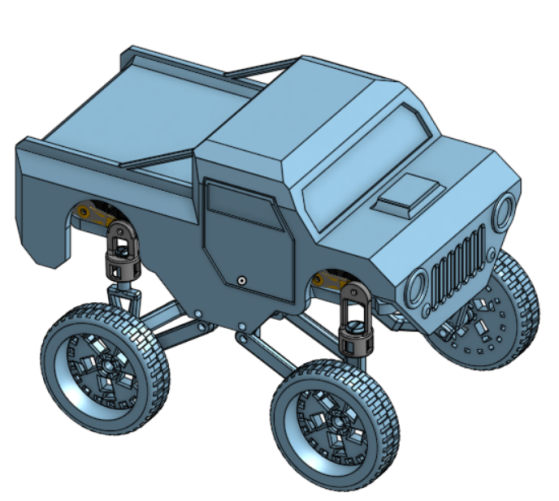


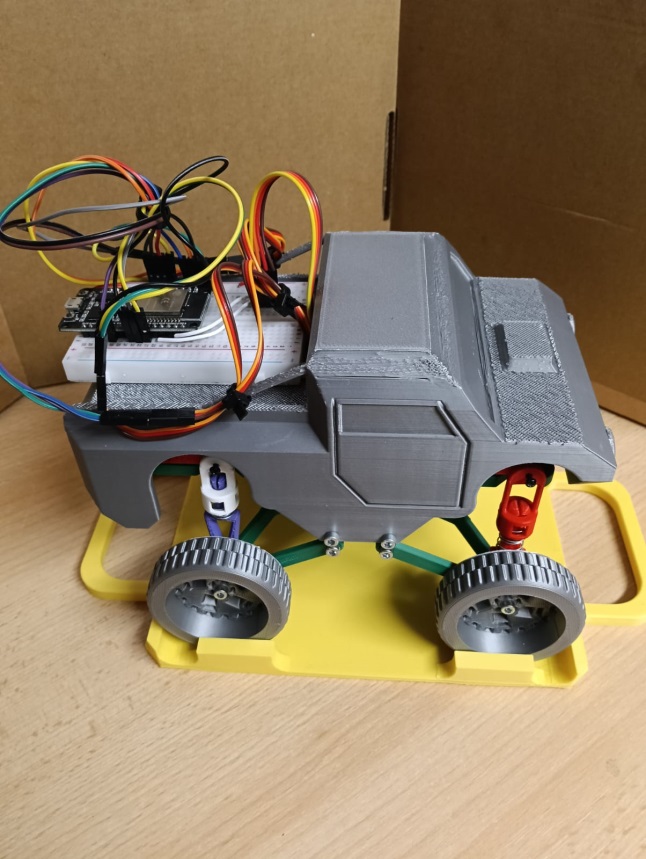
**3.5 Montarea finală**

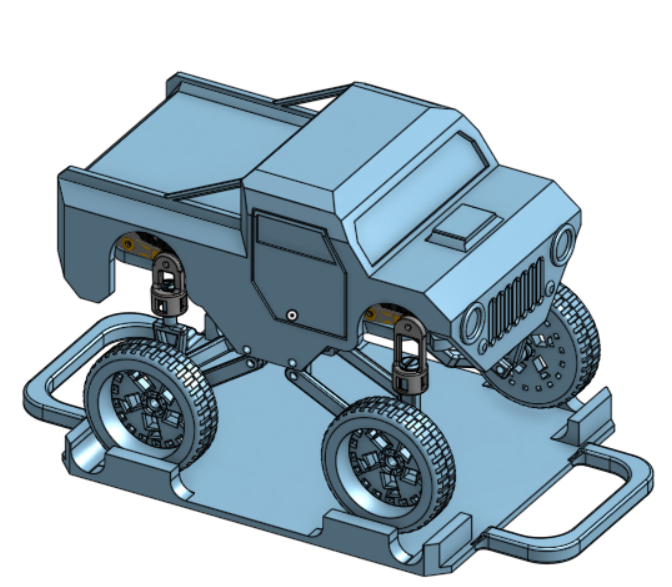
Am asamblat toate componentele și am realizat suportul final pentru MPU6050, poziționându-l în centrul mașinii. Acesta a fost securizat cu un șurub, iar suportul său a fost fixat cu alte două șuruburi de șasiu.

Restul componentelor electronice au fost amplasate pe carcasa mașinii pentru un acces mai facil în caz de defecte și pentru conectarea cablului de alimentare sau vizualizarea datelor.

****În cele din urmă, am ajuns la această versiune a mașinii, considerată varianta finală.



În cele din urmă, la partea practică, merită menționat faptul că, pentru prezentarea finală, am realizat și printat 3D o platformă pe care să o pot mișca manual, pentru a demonstra mai bine funcționalitatea sistemului.



De asemenea, aici sunt incluse toate componentele pe care le-am printat și reprintat, fie din motive de testare, fie pentru a corecta mici imperfecțiuni apărute pe parcursul acestui proiect.

**4. Rezultate**

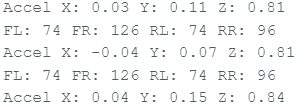
Ca produs final, mașina reușește să se adapteze la înclinațiile realiste ale solului, menținând totodată viziunea șoferului asupra drumului.

Astfel, în cazul înclinațiilor pe față, roțile din față se ridică pentru a menține o poziție relativ dreaptă a întregului șasiu, iar același principiu se aplică și pentru partea din spate.

De asemenea, în cazul înclinațiilor laterale, pentru a preveni dezechilibrarea și căderea într-o parte sau alta, roțile se ridică și coboară în perechi, două câte două, asigurând o poziție eficientă pentru menținerea echilibrului.

Viteza de răspuns a suspensiilor este calibrată pentru acest prototip: nici prea rapidă, ceea ce ar destabiliza sistemul, nici prea lentă, ceea ce ar face inutilă modificarea poziției. Totuși, pentru un model mai mare, ar fi necesară ajustarea acestei viteze.

Pentru o mai bună înțelegere a sistemului, am afișat în monitorul serial din Arduino IDE valorile accelerațiilor pentru axele X, Y și Z, precum și pozițiile servomotoarelor.



**4. Concluzii**

În final, obiectivele pe care mi le-am propus pentru acest proiect au fost realizate, proiectul reprezentând un succes personal. Prin acest proces, am reușit să îmi exersez cunoștințele deja acumulate și să explorez aspecte noi, pe care nu le mai abordasem până acum.

Pentru viitorul acestui proiect, se poate implementa un motor pentru propulsie și încă un servomotor pentru direcție. Totuși, nu am inclus aceste funcționalități în actuala versiune, deoarece proiectul s-a concentrat pe suspensii. Implementarea suspensiilor a fost cea mai dificilă parte, întrucât propulsia unei mașini de jucărie și direcționarea acesteia sunt aspecte realizate de nenumărate ori în alte proiecte și nu ar fi adus un plus de valoare sau complexitate. Practic, aș fi „reinventat roata”.

Algoritmul ar putea beneficia de îmbunătățiri în viitor, iar adăugarea de capabilități wireless ar putea extinde și mai mult funcționalitățile proiectului.

### Cost Total al Materialelor pentru Proiect

| **Componentă** | **Preț unitar (RON)** | **Cantitate** | **Total (RON)** |
| --- | --- | --- | --- |
| **ESP32** | 38,00 | 1 | 38,00 |
| **MPU6050** | 15,00 | 1 | 15,00 |
| **Servomotor** | 20,00 | 4 | 80,00 |
| **Cabluri, plastic, arcuri, șuruburi** | 20,00 | 1 | 20,00 |

### ****Cost total: 153,00 RON****

Ceea ce face acest proiect unul usor de reprodus la un cost minim, putand fi produs in masa pentru scopuri de salvare sau recunoastere in cazuri extreme, in locuri greu accesibile cum ar fi sub daramaturi sau in pesteri.

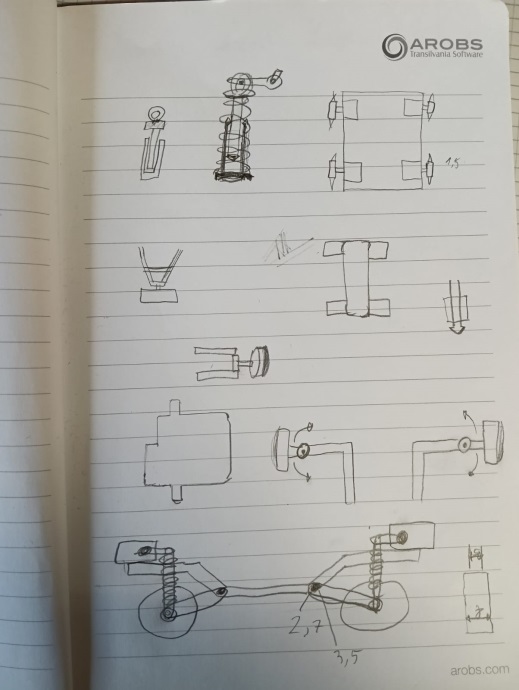
**Bibliografie**

1. **Random Nerd Tutorials - ESP32 Pinout Reference**  
Link: <https://randomnerdtutorials.com/esp32-pinout-reference-gpios/>  
Descriere: Ghid detaliat privind pinout-ul microcontrolerului ESP32, cu explicații despre funcționalitatea fiecărui GPIO și exemple de conexiuni pentru diverse aplicații.

2. **Wokwi Documentation - MPU6050**  
Link: <https://docs.wokwi.com/parts/wokwi-mpu6050>  
Descriere: Documentație completă pentru senzorul MPU6050 utilizat în simulări și proiecte electronice, incluzând descrierea pinilor, utilizare și exemple practice.

**Anexe**

Schite initiale ale mecanismelor.



Intregul cod al proiectului.

#include <Wire.h>          // I2C library for MPU-6050

#include <ESP32Servo.h>    // Servo library for ESP32

// Uncomment to enable debugging

#define DEBUG

// Pin assignment

#define SERVO\_FL\_PIN 4  // Front left servo pin

#define SERVO\_FR\_PIN 5  // Front right servo pin

#define SERVO\_RL\_PIN 18  // Rear left servo pin

#define SERVO\_RR\_PIN 19  // Rear right servo pin

#define MPU\_SDA 21       // I2C SDA pin

#define MPU\_SCL 22       // I2C SCL pin

// Servo objects

Servo servoFL;

Servo servoFR;

Servo servoRL;

Servo servoRR;

// MPU6050-related variables

float along = 0, across = 0, vertical = 0;

// Function prototypes

void setupMpu6050();

void readMpu6050Data();

void setup() {

#ifdef DEBUG

  Serial.begin(115200);

  delay(3000);  // Wait for serial monitor connection

  Serial.println("Starting ESP32 Active Suspension System...");

#endif

  // Attach servos

  servoFL.attach(SERVO\_FL\_PIN);

  servoFR.attach(SERVO\_FR\_PIN);

  servoRL.attach(SERVO\_RL\_PIN);

  servoRR.attach(SERVO\_RR\_PIN);

  // Initialize I2C for MPU-6050

  Wire.begin(MPU\_SDA, MPU\_SCL);

  // MPU6050 setup

  setupMpu6050();

}

void loop() {

  // Calculate suspension

  calculateSuspension();

  // Write servo positions

  writeServos();

}

// Setup MPU-6050

void setupMpu6050() {

  Wire.beginTransmission(0x68); // MPU-6050 I2C address

  Wire.write(0x6B);             // Power management register

  Wire.write(0);                // Wake up MPU-6050

  Wire.endTransmission(true);

#ifdef DEBUG

  Serial.println("MPU-6050 initialized");

#endif

}

// Read MPU-6050 data

void readMpu6050Data() {

  Wire.beginTransmission(0x68);

  Wire.write(0x3B); // Starting register for accelerometer data

  Wire.endTransmission(false);

  Wire.requestFrom(0x68, 6, true); // Request accelerometer data (6 bytes)

  int16\_t accelX = (Wire.read() << 8 | Wire.read());

  int16\_t accelY = (Wire.read() << 8 | Wire.read());

  int16\_t accelZ = (Wire.read() << 8 | Wire.read());

  along = accelX / 16384.0;   // Convert to g's

  across = accelY / 16384.0;  // Convert to g's

  vertical = accelZ / 16384.0; // Convert to g's

#ifdef DEBUG

  Serial.print("Accel X: "); Serial.print(along);

  Serial.print(" Y: "); Serial.print(across);

  Serial.print(" Z: "); Serial.println(vertical);

#endif

}

// Calculate suspension

void calculateSuspension() {

  readMpu6050Data();

  // Perform calculations or filtering if needed

}

// Smoothing parameters

const float smoothingFactor = 0.1;   // Low-pass filter weight (adjust for smoother/slower response)

const int transitionSpeed = 2;      // Speed of transition between current and target servo positions

// Store previous servo positions for interpolation

int currentServoFL = 90, currentServoFR = 90, currentServoRL = 90, currentServoRR = 90;

// Low-pass filtered MPU values

float filteredAlong = 0, filteredAcross = 0, filteredVertical = 0;

const float deadZoneThreshold = 0.2; // Ignore small MPU readings (increase to reduce jitter)

const int servoChangeThreshold = 2; // Ignore small servo position changes

// Move servo gradually toward target

int moveTowards(int current, int target, int step) {

  if (current < target) {

    return min(current + step, target);

  } else if (current > target) {

    return max(current - step, target);

  } else {

    return current;

  }

}

// Neutral points for each servo

const int neutralFL = 140;  // Front left servo neutral position

const int neutralFR = 60;  // Front right servo neutral position

const int neutralRL = 140;  // Rear left servo neutral position

const int neutralRR = 30;  // Rear right servo neutral position

void writeServos() {

  const float sensitivityFactor = 2.0; // Amplifies small MPU movements

  const float maxInputRange = 1.5;     // Maximum expected value from MPU

  const int maxServoRange = 60;        // Maximum servo deviation

  // Apply low-pass filter to MPU values

  filteredAlong = (smoothingFactor \* along) + ((1 - smoothingFactor) \* filteredAlong);

  filteredAcross = (smoothingFactor \* across) + ((1 - smoothingFactor) \* filteredAcross);

  filteredVertical = (smoothingFactor \* vertical) + ((1 - smoothingFactor) \* filteredVertical);

  // Apply dead zone: Ignore small changes

  float effectiveAlong = (abs(filteredAlong) > deadZoneThreshold) ? filteredAlong : 0;

  float effectiveAcross = (abs(filteredAcross) > deadZoneThreshold) ? filteredAcross : 0;

  float effectiveVertical = (abs(filteredVertical) > deadZoneThreshold) ? filteredVertical : 0;

  // Normalize and amplify MPU readings

  float normalizedAlong = constrain(effectiveAlong, -maxInputRange, maxInputRange) / maxInputRange;

  float normalizedAcross = constrain(effectiveAcross, -maxInputRange, maxInputRange) / maxInputRange;

  float normalizedVertical = constrain(effectiveVertical, -maxInputRange, maxInputRange) / maxInputRange;

  // Reverse the behavior for left/right tilt (across)

  normalizedAlong = -normalizedAlong;

  // Calculate target positions for servos using individual neutral points

  int targetFL = constrain(neutralFL + normalizedAlong \* maxServoRange \* sensitivityFactor -

                           normalizedAcross \* maxServoRange \* sensitivityFactor -

                           normalizedVertical \* maxServoRange \* sensitivityFactor, 0, 180);

  int targetFR = constrain(neutralFR - normalizedAlong \* maxServoRange \* sensitivityFactor -

                           normalizedAcross \* maxServoRange \* sensitivityFactor +

                           normalizedVertical \* maxServoRange \* sensitivityFactor, 0, 180);

  int targetRL = constrain(neutralRL - normalizedAlong \* maxServoRange \* sensitivityFactor -

                           normalizedAcross \* maxServoRange \* sensitivityFactor -

                           normalizedVertical \* maxServoRange \* sensitivityFactor, 0, 180);

  int targetRR = constrain(neutralRR + normalizedAlong \* maxServoRange \* sensitivityFactor -

                           normalizedAcross \* maxServoRange \* sensitivityFactor +

                           normalizedVertical \* maxServoRange \* sensitivityFactor, 0, 180);

  // Gradual transition to target positions only if change is significant

  if (abs(currentServoFL - targetFL) > servoChangeThreshold) {

    currentServoFL = moveTowards(currentServoFL, targetFL, transitionSpeed);

  }

  if (abs(currentServoFR - targetFR) > servoChangeThreshold) {

    currentServoFR = moveTowards(currentServoFR, targetFR, transitionSpeed);

  }

  if (abs(currentServoRL - targetRL) > servoChangeThreshold) {

    currentServoRL = moveTowards(currentServoRL, targetRL, transitionSpeed);

  }

  if (abs(currentServoRR - targetRR) > servoChangeThreshold) {

    currentServoRR = moveTowards(currentServoRR, targetRR, transitionSpeed);

  }

  // Write smoothed positions to servos

  servoFL.write(currentServoFL);

  servoFR.write(currentServoFR);

  servoRL.write(currentServoRL);

  servoRR.write(currentServoRR);

#ifdef DEBUG

  // Debug output

  Serial.print("FL: "); Serial.print(currentServoFL);

  Serial.print(" FR: "); Serial.print(currentServoFR);

  Serial.print(" RL: "); Serial.print(currentServoRL);

  Serial.print(" RR: "); Serial.println(currentServoRR);

#endif

}