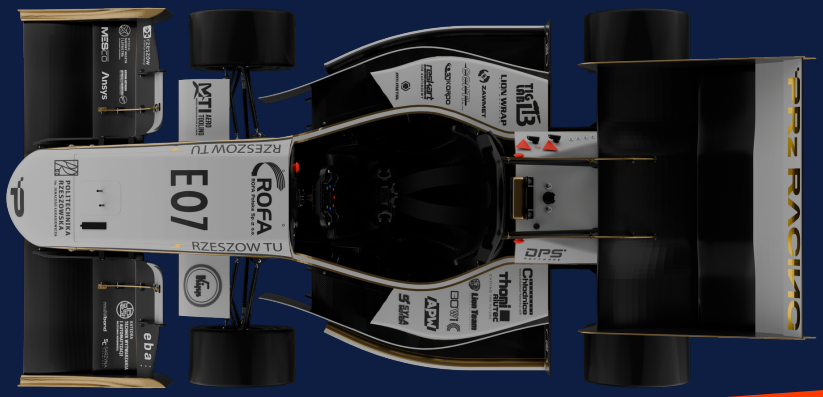
PRz RACING

System pomiaru geometrii zawieszenia pojazdów

Sekcja: Informatyka, Zawieszenie



Autor: **Bartosz Mytych**
Bolid: **PMT-07e, PMT-08e**
Sezon: **2024/25 – 2025/26**
Miejsce: **Rzeszów**

Spis treści

1 Wstęp

Niniejszy dokument opisuje system pomiaru geometrii zawieszenia, zaprojektowany dla bolidu PMT-07e. System ten stanowi kluczowe narzędzie w procesie przygotowania pojazdu do zawodów **Formuła Student**, umożliwiając precyzyjne i powtarzalne ustawienie parametrów jezdnych. System ten został opracowany, aby zastąpić tradycyjne, manualne metody pomiarowe rozwiązaniem cyfrowym, które pozwala na:

- monitorowanie kątów pochylenia (**Roll**) oraz zbieżności (**Yaw**) w czasie rzeczywistym,
- analizę kąta wyprzedzenia sworznia zwrotnicy (**Pitch**),
- jednoczesny pomiar nacisków na poszczególne koła (**corner weights**),
- jednoczesny pomiar nacisków na przekątne i osie pojazdu (**cross weights**),
- bezprzewodową transmisję danych do stacji bazowej (**PC GUI**),
- bezprzewodową transmisję danych do aplikacji mobilnej (**Android/iOS**).

1.1 Motywacja i cele

Głównym celem projektu było skrócenie czasu potrzebnego na ustawienie zawieszenia oraz eliminacja błędów odczytu. Precyzyjne ustawienie kół ma bezpośredni wpływ na temperaturę opon, stabilność bolidu w zakrętach oraz efektywność przenoszenia momentu obrotowego. Integracja czujników siły (tensometrów) z czujnikami orientacji (IMU) pozwala na analizę wpływu geometrii na rozkład mas w statyce.

2 Podstawy teoretyczne pomiarów

System WAS bazuje na fuzji danych z dwóch typów sensorów: magnetomechanicznych (IMU) oraz tensometrycznych. Kluczowym elementem jest interpretacja orientacji czujnika w przestrzeni 3D.

2.1 Pomiary kątowe (Roll, Pitch, Yaw)

Do wyznaczenia parametrów geometrii wykorzystywane są kąty Eulera dostarczane przez czujnik BNO055:

- **Roll:** Odpowiada za pomiar kąta nachylenia koła względem pionu patrząc od przodu bolidu.
- **Pitch:** Wykorzystywane do wyznaczenia kąta, co wpływa na stabilność kierunkową i siłę powracania kierownicy.
- **Yaw:** Stanowi bazę do pomiaru kąta zbieżności. Pozwala określić, czy koła są skierowane ku sobie, czy na zewnątrz osi symetrii pojazdu.

2.2 Pomiar nacisku (Wagi)

Wykorzystując belki tensometryczne z przetwornikiem HX711, system oblicza nacisk F na każde z kół. Wartość ta jest kluczowa dla obliczenia parametru *Cross Weight*:

$$CW[\%] = \frac{m_{FL} + m_{RR}}{m_{total}} \cdot 100 \tag{1}$$

gdzie m_{FL} i m_{RR} to masy nacisku odpowiednio dla koła przedniego-lewego i tylnego-prawego, analogicznie zadziała to dla koła przedniego-prawego i tylnego-lewego m_{FR} i m_{RL}.

3 Architektura systemu

System składa się z czterech warstw sprzętowo-programowych, co zapewnia dużą mobilność i odporność na zakłócenia w warunkach torowych.

3.1 ESP-32 Slave

Każde koło posiada własny mikrokontroler ESP32 połączone z czujnikiem `BNO055` oraz `HX711`. Dane przesyłane są w postaci struktury `pkt_t` poprzez protokół ESP-NOW.

Listing 1. Struktura danych zawierająca kąty Roll, Pitch i Yaw

```
typedef struct __attribute__((packed)) {
  char  id[3];          // "FL", "FR", "RL", "RR"
  float roll;           // Camber
  float pitch;          // Caster
  float yaw;            // Toe
  float ax, ay, az;     // Akcelerometr
  float load;           // Masa (HX711)
} pkt_t;
```

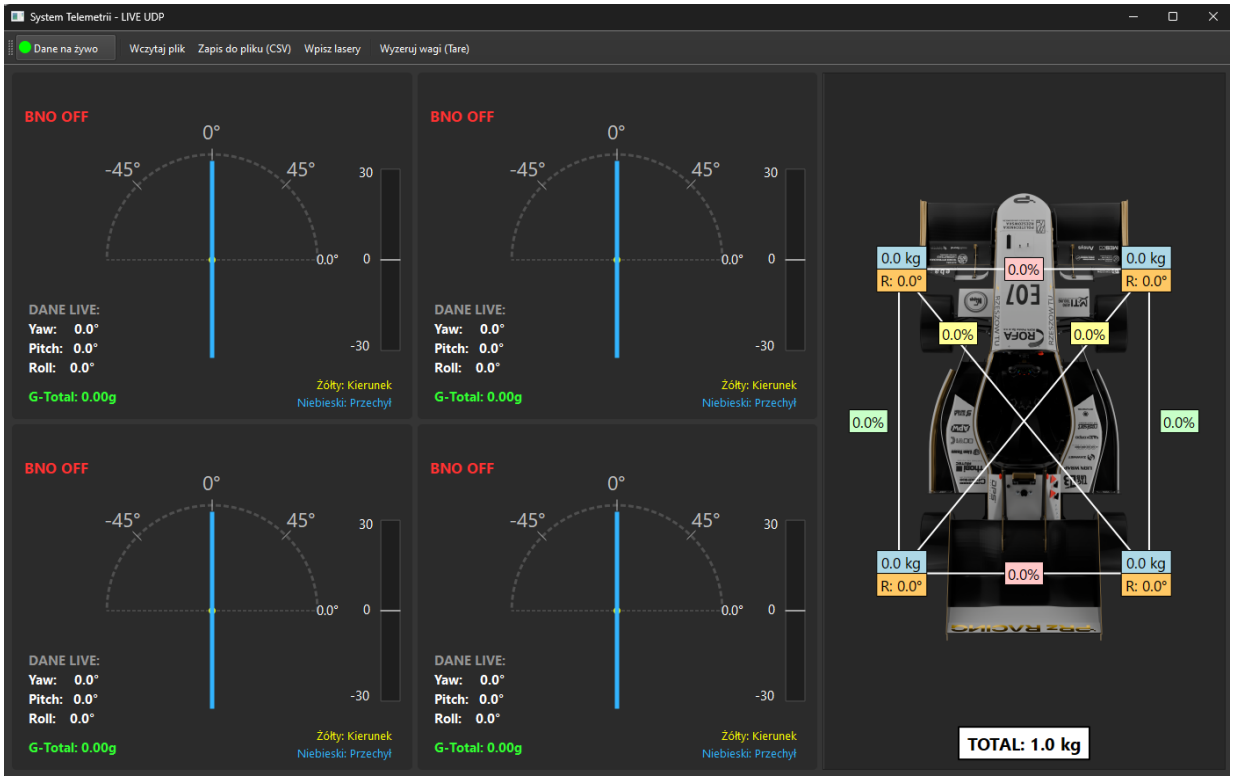
3.2 ESP-32 Master

Moduł centralny (*Master*) pełni rolę punktu dostępowego Wi-Fi. Odbiera dane ESP-NOW od wszystkich czterech sensorów, konwertuje je na format CSV (tekstowy) i rozsyła jako pakiet UDP Broadcast wewnątrz sieci `WHEELS_NET`.

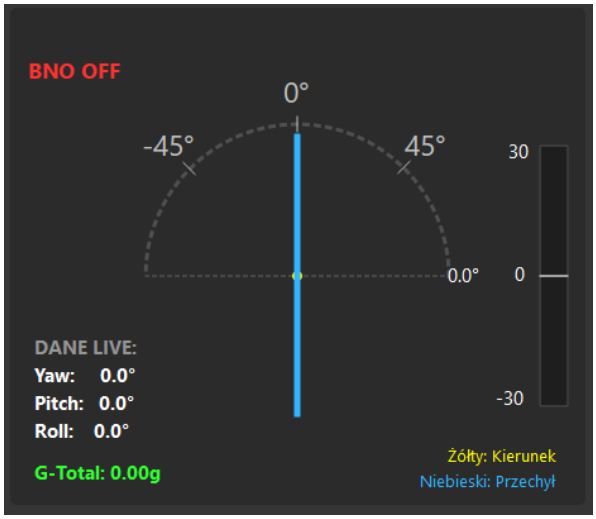
3.3 Aplikacja PC (Stacja bazowa)

Oprogramowanie wizualizacyjne (`wheel_gui_new.py`) zostało zaimplementowane w języku Python przy użyciu biblioteki **PyQt6**. Program pełni rolę głównego interfejsu inżynierskiego.

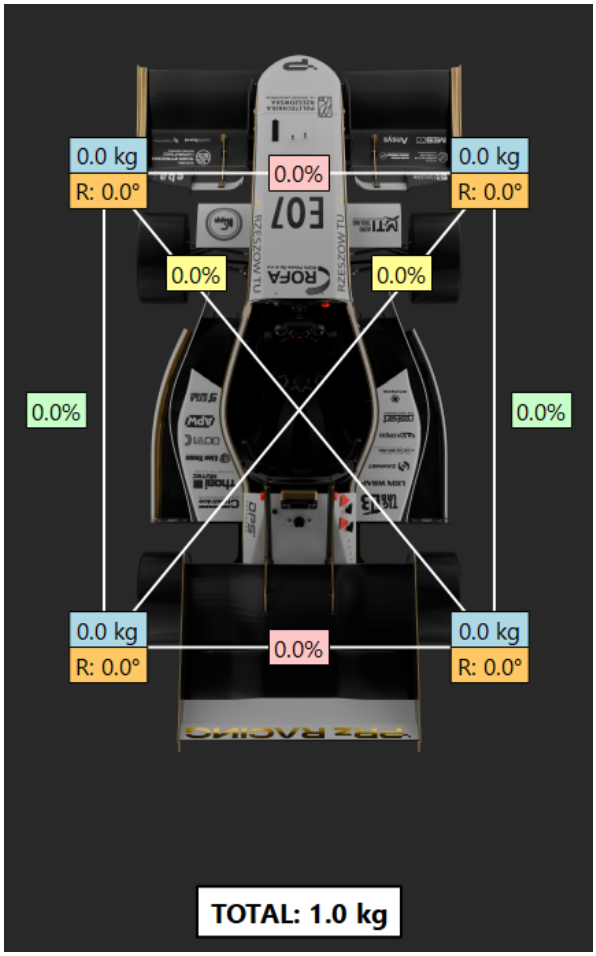
- **Komunikacja:** Aplikacja otwiera gniazdo UDP na porcie `4210` i nasłuchuje pakietów rozgłoszeniowych (*broadcast*) wysyłanych przez moduł Master.
- **Przetwarzanie danych:** Dane tekstowe w formacie CSV są parsowane w czasie rzeczywistym i przekazywane do managera danych, który przechowuje bufory ostatnich pomiarów.
- **Wizualizacja:** GUI renderuje dynamiczne wykresy słupkowe nacisków na koła oraz graficzną reprezentację pochylenia kół (**Pitch, Roll** i **Yaw**) z wykorzystaniem transformacji `QTransform`.



Rysunek 1. Główny widok aplikacji PC – cztery wskaźniki kątowe oraz widok nacisków na koła



Rysunek 2. Wskaźnik kątowy – Yaw, Pitch, Roll i przyspieszenie G

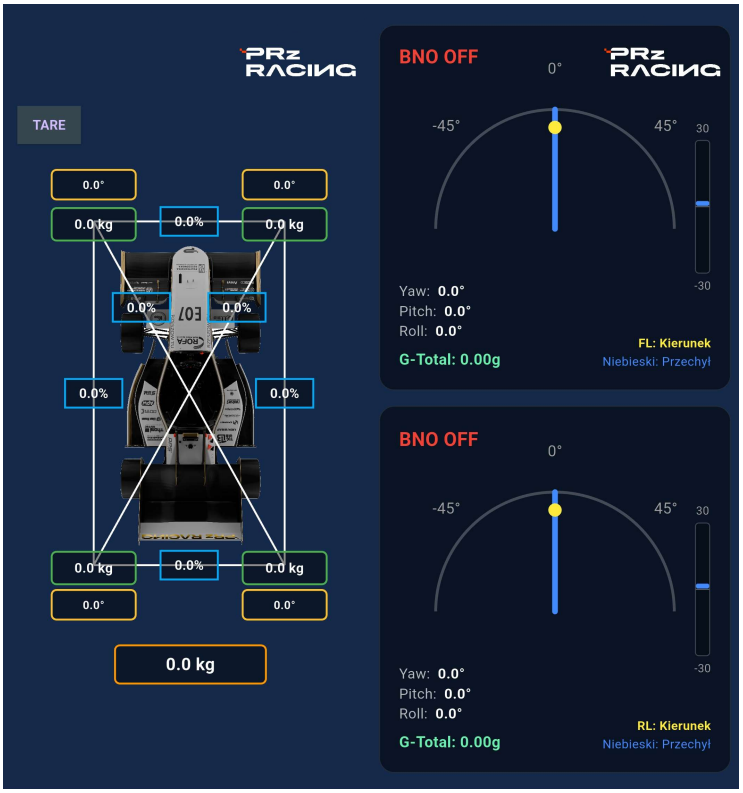


Rysunek 3. Naciski na koła – corner weights i cross weights

3.4 Aplikacja mobilna (Monitor pomocniczy)

W celu zwiększenia ergonomii pracy w warunkach warsztatowych, system został rozszerzony o lekki interfejs mobilny. Pozwala on mechanikowi znajdującemu się bezpośrednio przy zawieszeniu na podgląd wartości bez konieczności zerkania na ekran laptopa.

- **Dostępność:** Dzięki transmisji UDP Broadcast wewnątrz sieci `WHEELS_NET`, smartfon połączony z Wi-Fi może odbierać te same dane, co stacja bazowa PC.
- **Funkcjonalność:** Aplikacja skupia się na czytelnym wyświetlaniu surowych wartości **Pitch, Roll i Yaw** oraz statusu kalibracji wagi dla konkretnego koła, przy którym aktualnie odbywa się regulacja.
- **Mobilność:** Rozwiązanie to eliminuje potrzebę komunikacji głosowej między mechanikiem a operatorem komputera, co znacząco przyspiesza proces ustawiania zbieżności.



Rysunek 4. Aplikacja mobilna – widok nacisków na koła i wskaźniki kątowe (Android/iOS)

Zastosowanie standardu UDP pozwala na podłączenie nieograniczonej liczby urządzeń odbiorczych (PC, tablety, telefony) bez obciążania procesora modułu Master.

4 Konfiguracja i uruchomienie

4.1 Parametry sieciowe

Master tworzy hasłowaną sieć Wi-Fi o następujących parametrach:

- **SSID:** `WHEELS_NET`
- **UDP Port:** `4210`

4.2 Kalibracja czujników IMU

Aby zapewnić poprawność odczytów **Roll, Pitch i Yaw**, czujniki muszą przejść proces kalibracji po zamontowaniu na uchwytach kół.

1. Ustawienie bolidu na wypoziomowanych wagach.
2. Resetowanie orientacji w GUI (funkcja zerowania kątów).
3. Weryfikacja odczytów z poziomicą cyfrową.

Przed uruchomieniem aplikacji upewnij się, że komputer PC jest połączony z siecią Wi-Fi `WHEELS_NET`, a statusy sensorów w GUI wskazują aktywność
Ważne! Informacja BNO OFF informuje o braku aktywności czujnika.

5 Historia rozwoju i rozwiązywanie problemów

System ten powstał jako projekt rekrutacyjny do sekcji informatyki koła naukowego. Proces implementacji pozwolił na zidentyfikowanie kluczowych problemów w komunikacji bezprzewodowej niskiego poziomu.

5.1 Problemy z implementacją protokołu ESP-NOW

Początkowym i największym wyzwaniem był poprawny przesył informacji między modułami **Slave** (sensory) a modułem **Master**. Główną barierą był brak znajomości bibliotek obsługujących najnowsze rewizje układów ESP32 w środowisku Arduino/PlatformIO.

- **Problem:** Brak stabilnych bibliotek wymuszał operowanie na surowym API z nagłówka `esp_now.h`. Powodowało to błędy przy inicjalizacji interfejsu Wi-Fi w trybie hybrydowym (AP + STA).
- **Rozwiązanie:** Zaimplementowano bezpośrednią obsługę struktur `esp_now_recv_info_t` oraz ręczne zarządzanie kanałami Wi-Fi. Kluczowe okazało się ustawienie stałego kanału (`WIFI_CH = 1`) dla wszystkich urządzeń, co wyeliminowało gubienie pakietów podczas skanowania sieci.

5.2 Katalog znanych problemów (Debug)

Problem	Przyczyna	Rozwiązanie	Status
`len mismatch`	Niezgodność struktur `pkt_t` między Slave a Masterem.	Użycie atrybutu `__attribute__((packed))`.	Rozwiązano
Lag w GUI	Zbyt duża częstotliwość odświeżania wykresów.	Zastosowanie `QTimer` 50 ms w Pythonie.	Rozwiązano

W przypadku braku odczytów w GUI, pierwszym krokiem powinno być sprawdzenie monitora szeregowego na Masterze. Jeśli tam dane trafiają (logi „ESP-NOW OK”), problem leży w konfiguracji sieciowej komputera lub zablokowanym porcie UDP 4210.

6 Bezpieczeństwo i niezawodność systemu

System wheel aligmentu, mimo że przeznaczony głównie do prac stacjonarnych, został zaprojektowany z uwzględnieniem rygorystycznych zasad niezawodności, typowych dla systemów telemetrii wyścigowej.

6.1 Odporność na awarie czujników

W oprogramowaniu sensorów (`main.cpp`) zaimplementowano mechanizm wykrywania obecności czujników na magistrali I2C.

- **Watchdog czujnika BNO055:** System cyklicznie sprawdza rejestr statusu czujnika. W przypadku wykrycia błędu (np. przerwanie przewodu, zakłócenia EM), sensor nie zawiesza pętli głównej, lecz zeruje wartości kątów **Roll, Pitch i Yaw** i podejmuje próbę reinicjalizacji co 3 sekundy.
- **Weryfikacja długości pakietu:** Moduł Master przed przekazaniem danych do GUI sprawdza sumaryczną długość odebranego pakietu (`len == sizeof(pkt_t)`). Zapobiega to przetwarzaniu danych powstałych w wyniku kolizji radiowych.

6.2 Niezawodność transmisji bezprzewodowej

Zastosowanie protokołu ESP-NOW w warstwie krytycznej (Sensor-Master) eliminuje narzut związany z utrzymywaniem sesji TCP, co jest kluczowe w środowisku o dużym zagłuszeniu.

> Dzięki stałej konfiguracji kanału Wi-Fi (`WIFI_CH = 1`), system pomija fazę negocjacji połączenia, co pozwala na natychmiastowy powrót do odczytów po restarcie zasilania.

6.3 Integrity Check w GUI

Aplikacja komputerowa została wyposażona w mechanizmy zapobiegające błędnej interpretacji danych:

1. **Status połączenia:** Ikona statusu w GUI zmienia kolor na czerwony, jeśli przez więcej niż 500 ms nie odebrano nowego pakietu UDP od Mastera.
2. **Zabezpieczenie zapisu:** Zapis do pliku CSV odbywa się w oddzielnym wątku, co gwarantuje, że awaria interfejsu graficznego nie spowoduje utraty danych.
3. **Filtrowanie danych:** GUI odrzuca pakiety, których identyfikator ID nie pasuje do zdefiniowanej listy (FL, FR, RL, RR).

> Należy pamiętać, że system bazuje na transmisji UDP Broadcast. W przypadku pracy wielu zespołów należy zmienić identyfikator `SSID` oraz port UDP w celu uniknięcia interferencji.

7 Pliki projektowe i kod źródłowy

Niniejszy rozdział zawiera kompletny wykaz plików projektowych wraz z kluczowymi listingami kodu źródłowego. Projekt składa się z czterech niezależnych komponentów programowych, opracowanych w różnych środowiskach i językach.

7.1 Struktura plików projektu

Plik / Katalog	Język	Opis
`Suspension_tool/src/main.cpp`	C++ (Arduino)	Firmware modułu Slave – sensor koła
`Suspension_tool/master/master.ino`	C++ (Arduino)	Firmware modułu Master – brama sieciowa
`Suspension_tool/wheel_gui_new.py`	Python 3	Aplikacja PC – stacja bazowa (PyQt6)
`Suspension_tool/odbiornik.py`	Python 3	Narzędzie diagnostyczne – odbiornik UDP
`app/telemetry_app/`	Dart (Flutter)	Aplikacja mobilna – monitor pomocniczy
`Suspension_tool/dist/wheel_gui_new.exe`	Wykonywalny	Skompilowana wersja aplikacji PC (.exe)
`Suspension_tool/platformio.ini`	INI	Konfiguracja środowiska PlatformIO

Wersja wykonywalna aplikacji PC (`wheel_gui_new.exe`) dostępna jest w katalogu `Suspension_tool/dist/`. Plik ten nie wymaga instalacji środowiska Python – uruchamia się bezpośrednio w systemie Windows.

7.2 Firmware modułu Slave (`main.cpp`)

Poniższy listing przedstawia kompletny kod firmware uruchamianego na każdym z czterech modułów ESP32, zamontowanych przy kołach bolidu. Każdy sensor posiada unikalny identyfikator (`FL`, `FR`, `RL`, `RR`) oraz adres MAC modułu Master zakodowany statycznie.

Listing 2. Firmware modułu Slave – `main.cpp`

```
include <WiFi.h>
#include <esp_wifi.h>
#include <esp_now.h>

#include <Wire.h>
#include <SPI.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>
#include "HX711.h"
```

```
// Piny HX711
constexpr int LOADCELL_DOUT_PIN = 16;
constexpr int LOADCELL_SCK_PIN  = 4;

HX711 scale;
bool bnoPresent = false; // Flaga obecnosci czujnika
uint32_t lastBnoRetry = 0;
constexpr uint32_t BNO_RETRY_INTERVAL = 3000; // 3 sekundy

// Kalibracja wagi
float loadOffset = 0;
float loadScale  = 14.2;

// Struktura pakietu
typedef struct __attribute__((packed)) {
  char  id[3];
  float roll, pitch, yaw;
  float ax, ay, az;
  float load;
} pkt_t;

pkt_t pkt;

// ID sensora i MAC Mastera
constexpr char SENSOR_ID[] = "RR";
uint8_t masterMac[] = {0xCC, 0x7B, 0x5C, 0x27, 0xF1, 0x18};

// BNO055
Adafruit_BNO055 bno(55, 0x28);

void onSent(const uint8_t *mac, esp_now_send_status_t status) {
  // Serial.println(status == ESP_NOW_SEND_SUCCESS ? "ESP-NOW OK"
     : "ESP-NOW FAIL");
}

void setup() {
  Serial.begin(115200);
  delay(300);

  strncpy(pkt.id, SENSOR_ID, 2);
  pkt.id[2] = '\0';

  Serial.println("\n=== SENSOR START ===");

  // --- WiFi & ESP-NOW ---
  WiFi.mode(WIFI_STA);
  WiFi.disconnect();
  esp_wifi_set_channel(1, WIFI_SECOND_CHAN_NONE);
```

```
  if (esp_now_init() != ESP_OK) {
    Serial.println("ESP-NOW init FAIL");
  }
  esp_now_register_send_cb(onSent);

  esp_now_peer_info_t peer{};
  memcpy(peer.peer_addr, masterMac, 6);
  peer.channel = 1;
  peer.encrypt = false;
  esp_now_add_peer(&peer);

  // --- BNO055 (Proba inicjalizacji) ---
  if (!bno.begin()) {
    Serial.println("UWAGA: BNO055 nie zostal znaleziony! Bede
       wysylal 0.0.");
    bnoPresent = false;
  } else {
    Serial.println("BNO055 OK!");
    bnoPresent = true;
    bno.setExtCrystalUse(true);
    bno.setMode(OPERATION_MODE_NDOF);
  }

  // --- HX711 ---
  scale.begin(LOADCELL_DOUT_PIN, LOADCELL_SCK_PIN);
  if (scale.is_ready()) {
    scale.tare();
    loadOffset = scale.get_offset();
    Serial.println("Waga gotowa.");
  } else {
    Serial.println("Blad wagi!");
  }
}

void loop() {
  uint32_t now = millis();
// --- 1. PROBA ODZYSKANIA CZUJNIKA ---
  if (!bnoPresent && (now - lastBnoRetry >= BNO_RETRY_INTERVAL)) {
    lastBnoRetry = now;
    Serial.println("Proba odnalezienia BNO055...");

    Wire.end();
    delay(50);
    Wire.begin();
    Wire.setClock(100000); // Ustawiamy stabilne 100kHz (zamiast
       domyslnych 400kHz)

    if (bno.begin()) {
      delay(200); // DAJEMY CZAS NA ROZRUCH UKLADU
```

```
      bno.setExtCrystalUse(true);
      bno.setMode(OPERATION_MODE_NDOF);
      delay(100);

      // Sprawdzamy czy faktycznie zyje przed ustawieniem flagi
      uint8_t sys, self, err;
      bno.getSystemStatus(&sys, &self, &err);
      if (err == 0) {
        bnoPresent = true;
        Serial.println("BNO055 PODLACZONY I GOTOWY!");
      } else {
        Serial.printf("BNO znaleziony, ale zglasza blad: %d\n",
           err);
      }
    }
  }

  // --- 2. PETLA DANYCH 10Hz ---
  static uint32_t t0 = 0;
  if (now - t0 >= 100) {
    t0 = now;

    if (bnoPresent) {
      // Pobieramy status, zeby sprawdzic czy czujnik zyje
      uint8_t sys, self, err;
      bno.getSystemStatus(&sys, &self, &err);

      // Jesli system_status == 0 lub blad, uznajemy ze odpieto
         czujnik
      if (err != 0 || sys == 0) {
        Serial.println("BLAD: Stracono polaczenie z BNO!");
        bnoPresent = false;
        // Zerujemy dane w pakiecie
        pkt.roll = pkt.pitch = pkt.yaw = 0.0f;
        pkt.ax = pkt.ay = pkt.az = 0.0f;
      } else {
        // Jesli wszystko OK - czytamy dane
        imu::Vector<3> e = bno.getVector(Adafruit_BNO055::
           VECTOR_EULER);
        imu::Vector<3> a = bno.getVector(Adafruit_BNO055::
           VECTOR_LINEARACCEL);
        pkt.roll = e.x(); pkt.pitch = e.y(); pkt.yaw = e.z();
        pkt.ax = a.x();   pkt.ay = a.y();    pkt.az = a.z();
      }
    } else {
      // Czujnika nie ma - wysylamy zera
      pkt.roll = pkt.pitch = pkt.yaw = 0.0f;
      pkt.ax = pkt.ay = pkt.az = 0.0f;
    }
```

```
    // Odczyt wagi (zawsze)
    if (scale.is_ready()) {
      pkt.load = (scale.read_average(2) - loadOffset) / loadScale;
    }

    // Wysylka Serial i ESP-NOW
    Serial.printf("%s, %.2f %.2f %.2f   %.2f %.2f %.2f   %.2f\n",
                  pkt.id, pkt.roll, pkt.pitch, pkt.yaw,
                  pkt.ax, pkt.ay, pkt.az, pkt.load);

    // Wysylanie ESP-NOW do MASTER
    esp_now_send(masterMac, (uint8_t*)&pkt, sizeof(pkt));
  }
}
```

7.3 Firmware modułu Master (master.ino)

Moduł Master tworzy sieć Wi-Fi (WHEELS_NET), przyjmuje pakiety ESP-NOW od wszystkich czterech sensorów i konwertuje je do formatu CSV, który następnie rozsyła jako UDP Broadcast do wszystkich urządzeń w sieci.

Listing 3. Firmware modułu Master – master.ino

```
/*****************************************************************
 *  MASTER -- SoftAP + ESP-NOW -> UDP (broadcast, 7 pol)
 *****************************************************************/

#include <WiFi.h>
#include <esp_wifi.h>
#include <esp_now.h>
#include <WiFiUdp.h>

/*---- parametry sieci ----*/
constexpr char   AP_SSID[] = "WHEELS_NET";
constexpr char   AP_PASS[] = "wheels123";
constexpr uint8_t WIFI_CH   = 1;            // wspolny kanal

constexpr uint16_t PC_PORT   = 4210;        // port PC
constexpr uint16_t LOCAL_PORT = 4211;       // lokalny
IPAddress  PC_BCAST(192, 168, 4, 255);      // broadcast

/*---- struktura pakietu (7 floatow) ----*/
typedef struct __attribute__((packed)) {
  char  id[3];
  float roll, pitch, yaw;
  float ax, ay, az;
  float load;
} pkt_t;
```

```
/*---- globalne ----*/
WiFiUDP udp;

/*---- callback ESP-NOW ----*/
void onEspNowRecv(const esp_now_recv_info_t *info,
                  const uint8_t *data, int len)
{
  Serial.printf("ESP-NOW %02X:%02X ... len=%d\n",
                info->src_addr[0], info->src_addr[1], len);

  if (len != sizeof(pkt_t)) {
    Serial.println("[!] len mismatch");
    return;
  }

  const pkt_t *p = reinterpret_cast<const pkt_t*>(data);
  char buf[96];
  int n = snprintf(buf, sizeof(buf),
                   "%s,%.2f,%.2f,%.2f,%.2f,%.2f,%.2f,%.2f\n",
                   p->id, p->roll, p->pitch, p->yaw,
                   p->ax, p->ay, p->az, p->load);

  udp.beginPacket(PC_BCAST, PC_PORT);
  udp.write((uint8_t*)buf, n);
  bool ok = udp.endPacket();
  Serial.printf(" -> UDP %s\n", ok ? "OK" : "FAIL");
}

/*---- SETUP ----*/
void setup()
{
  Serial.begin(115200);
  Serial.println("=== WHEELS MASTER v2 (RPY + ACC + LOAD) ===");

  /* 1. SoftAP */
  WiFi.mode(WIFI_AP_STA);
  WiFi.softAP(AP_SSID, AP_PASS, WIFI_CH);
  WiFi.softAPConfig(IPAddress(192,168,4,1),
                    IPAddress(192,168,4,1),
                    IPAddress(255,255,255,0));
  Serial.printf("SoftAP %s  CH=%u\n", AP_SSID, WIFI_CH);

  /* 2. kanał dla radia */
  esp_wifi_set_channel(WIFI_CH, WIFI_SECOND_CHAN_NONE);

  /* 3. UDP */
  udp.begin(LOCAL_PORT);

  /* 4. ESP-NOW */
```

```
  if (esp_now_init() != ESP_OK) {
    Serial.println("[X] ESP-NOW init fail");
    while (true) delay(1000);
  }
  esp_now_register_recv_cb(onEspNowRecv);
  Serial.println("ESP-NOW ready, czekam ...");
}

/*---- LOOP ----*/
void loop() { delay(1000); }  // calkowite przetwarzanie w
   callbacku
```

7.4 Narzędzie diagnostyczne (odbiornik.py)

Skrypt odbiornik.py służy do weryfikacji poprawności transmisji UDP bez uruchamiania pełnego interfejsu graficznego. Nasłuchuje na porcie 4210 i wyświetla odebrane dane w konsoli w czasie rzeczywistym. Jest szczególnie przydatny podczas uruchamiania i kalibracji systemu.

Listing 4. Narzędzie diagnostyczne – odbiornik.py

```
!/usr/bin/env python3
import socket
import threading
import os
import time

UDP_IP = "0.0.0.0"
UDP_PORT = 4210


# przechowujemy: (roll, pitch, yaw, ax, ay, az, tensometr)
sensor_ids = ("FL", "FR", "RL", "RR")
sensor_data = {sid: None for sid in sensor_ids}

lock = threading.Lock()

# ----- watek odbioru UDP -----
def udp_listener():
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sock.bind((UDP_IP, UDP_PORT))
    print(f"Nasluchiwanie na UDP {UDP_PORT} ...")

    while True:
        data, addr = sock.recvfrom(256)

        try:
            line = data.decode(errors="ignore").strip()
            print("[RAW] ->", repr(line), "from", addr)
```

```
            fields = [f.strip() for f in line.split(",")]

            # Kazdy czujnik ma: ID + 7 wartosci (R,P,Y,Ax,Ay,Az,T)
            if fields[0] in sensor_ids and len(fields) == 8:
                sid = fields[0]
                try:
                    nums = tuple(map(float, fields[1:]))

                    with lock:
                        sensor_data[sid] = nums

                except:
                    print(f"[!] Blad konwersji w ramce {sid}:",
                       fields)

            else:
                print("[!] Nieoczekiwany format:", line)

        except Exception as e:
            print("Blad parsowania:", e)

# ----- petla wyswietlania -----
def display_loop():
    while True:
        time.sleep(0.2)
        with lock:
            print("\033[2J\033[H")
            os.system('cls' if os.name == 'nt' else 'clear')
            print("===  DANE Z CZUJNIKOW (RPY, Acc, Tensometr)
               ===\n")

            for sid in sensor_ids:
                if sensor_data[sid]:
                    r,p,y, ax,ay,az, t = sensor_data[sid]
                    print(f"{sid}: "
                          f"R={r:7.2f}deg  P={p:7.2f}deg  Y={y:7.2
                             f}deg   |  "
                          f"Ax={ax:+6.2f}  Ay={ay:+6.2f}  Az={az
                             :+6.2f}   |  "
                          f"Load={t/1.961:+7.2f}")
                else:
                    print(f"{sid}:  -- brak danych --")

# ----- main -----
if __name__ == "__main__":
    threading.Thread(target=udp_listener, daemon=True).start()
    display_loop()
```

7.5 Aplikacja PC – kluczowe fragmenty (`wheel_gui_new.py`)

Aplikacja PC jest największym komponentem programowym projektu (ok. 900 linii kodu). Poniżej przedstawiono najważniejsze elementy architektury oprogramowania.

7.5.1 Konfiguracja i stałe systemowe

Listing 5. Konfiguracja aplikacji PC

```
UDP_PORT    = 4210        # port nasluchu UDP
SENSOR_IDS  = ("FL", "FR", "RL", "RR")
PANEL_HZ    = 30          # czestotliwosc odswiezania GUI [FPS]
G_WARN      = 1.5         # prog ostrzezenia przyspieszenia [g]

CSV_HEADER  = [
    "Timestamp", "Note",
    "Laser_FL", "Laser_FR", "Laser_RL", "Laser_RR",
    "FL_Roll", "FL_Pitch", "FL_Yaw", "FL_Ax", "FL_Ay", "FL_Az", "
       FL_Load",
    "FR_Roll", "FR_Pitch", "FR_Yaw", "FR_Ax", "FR_Ay", "FR_Az", "
       FR_Load",
    "RL_Roll", "RL_Pitch", "RL_Yaw", "RL_Ax", "RL_Ay", "RL_Az", "
       RL_Load",
    "RR_Roll", "RR_Pitch", "RR_Yaw", "RR_Ax", "RR_Ay", "RR_Az", "
       RR_Load"
]
```

7.5.2 Manager danych (klasa `DataManager`)

Klasa `DataManager` odpowiada za odbieranie pakietów UDP w osobnym wątku i udostępnianie danych do warstwy wizualizacji. Zastosowanie blokady `threading.Lock` gwarantuje bezpieczny dostęp współbieżny.

Listing 6. Szkielet klasy DataManager

```
class DataManager:
    def __init__(self):
        self._lock   = threading.Lock()
        self._latest = {sid: None for sid in SENSOR_IDS}
        self._sock   = socket.socket(socket.AF_INET, socket.
           SOCK_DGRAM)
        self._sock.setsockopt(socket.SOL_SOCKET, socket.
           SO_REUSEADDR, 1)
        self._sock.bind(("0.0.0.0", UDP_PORT))
        threading.Thread(target=self._recv_loop, daemon=True).
           start()

    def _recv_loop(self):
        while True:
            data, _ = self._sock.recvfrom(256)
```

```
            fields = data.decode(errors="ignore").strip().split(",
                ")
            if fields[0] in SENSOR_IDS and len(fields) == 8:
                with self._lock:
                    self._latest[fields[0]] = tuple(map(float,
                        fields[1:]))

    def get(self, sid):
        with self._lock:
            return self._latest.get(sid)
```

7.5.3 Budowanie wersji wykonywalnej (.exe)

Wersja wykonywalna aplikacji PC została zbudowana przy użyciu narzędzia **PyInstaller**. Plik wynikowy `wheel_gui_new.exe` zlokalizowany jest w katalogu `Suspension_tool/dist/` i nie wymaga instalacji interpretera Python na komputerze docelowym.

Listing 7. Polecenie budowania wersji .exe

```
pyinstaller --onefile --windowed \
  --add-data "bolid.png;." \
  wheel_gui_new.py
```

7.6 Aplikacja mobilna (app/telemetry_app/)

Aplikacja mobilna została opracowana w technologii **Flutter** (język Dart), co zapewnia jej działanie zarówno na systemach **Android**, jak i **iOS**. Architektura oparta jest o wzorzec Provider.

Plik	Rola
`lib/main.dart`	Punkt wejścia aplikacji
`lib/app.dart`	Główny widget aplikacji
`lib/providers/telemetry_provider.dart`	Odbiornik UDP + zarządzanie stanem
`lib/models/wheel_data.dart`	Model danych koła
`lib/widgets/wheels_bno_page.dart`	Strona z kątami Roll/Pitch/Yaw
`lib/widgets/car_load_view.dart`	Widok nacisków na koła

Listing 8. Punkt wejścia aplikacji Flutter – `main.dart`

```
import 'package:flutter/material.dart';
import 'package:telemetry_app/app.dart';

void main() {
  runApp(const TelemetryApp());
}
```

7.7 Konfiguracja środowiska PlatformIO

Firmware dla modułów ESP32 jest kompilowany i wgrywany przy użyciu platformy **PlatformIO** (zintegrowanej z VS Code). Plik `platformio.ini` definiuje docelową płytkę oraz

wymagane biblioteki.

Listing 9. Konfiguracja PlatformIO – `platformio.ini`

```
; PlatformIO Project Configuration File
;
;   Build options: build flags, source filter
;   Upload options: custom upload port, speed and extra flags
;   Library options: dependencies, extra library storages
;   Advanced options: extra scripting
;
; Please visit documentation for the other options and examples
; https://docs.platformio.org/page/projectconf.html

[env:upesy_wroom]
platform = espressif32@6.4.0
board = upesy_wroom
framework = arduino
lib_deps =
    bogde/HX711@^0.7.5
    adafruit/Adafruit BNO055@^1.6.4
    adafruit/Adafruit Unified Sensor@^1.1.15
    adafruit/Adafruit BusIO@^1.17.4
    espressif/esp32-camera@^2.0.4
monitor_speed = 115200
```

8 Kierunki rozwoju i wskazówki dla następców

Jako projekt rekrutacyjny, system ten stanowi solidną bazę, która może zostać rozwinięta w stronę profesjonalnego systemu telemetrii warsztatowej.

8.1 Rozbudowa sprzętowa

1. **Integracja Laserów:** Dodanie czujników odległości do pomiaru ugięcia zawieszenia w czasie rzeczywistym.
2. **Obudowy 3D:** Zaprojektowanie lekkich, magnetycznych uchwytów na sensory, które pozwolą na błyskawiczny montaż na tarczach hamulcowych lub felgach.

8.2 Rozbudowa oprogramowania

- **Moduł Analityczny:** Implementacja w GUI algorytmu sugerującego zmiany w ustawieniach (np. „Zwiększ Camber o 0.5 stopnia”), bazując na docelowych parametrach wpisanych przez inżyniera.

> Dla osób chcących rozwijać ten projekt: Największą wartością systemu jest jego modularność. Protokół UDP pozwala na łatwe dopisanie nowego klienta (np. wtyczki do programów typu MATLAB czy Excel) bez ingerencji w kod mikrokontrolerów.

8.3 Wskazówki dla nowych członków

- Zawsze dbaj o to, by struktura `pkt_t` była identyczna w kodzie `main.cpp` i `master.ino`. Każda zmiana kolejności pól zepsuje odczyt danych.
- Przy problemach z zasięgiem Wi-Fi, rozważ dodanie anteny zewnętrznej do modułu Master.