POLITECHNIKA RZESZOWSKA
im. IGNACEGO ŁUKASIEWICZA

PROJEKT

Mobilny pojazd z nauczonym modelem AI wykrywającym piłkę

3EA-DI L2

Andrzej O.

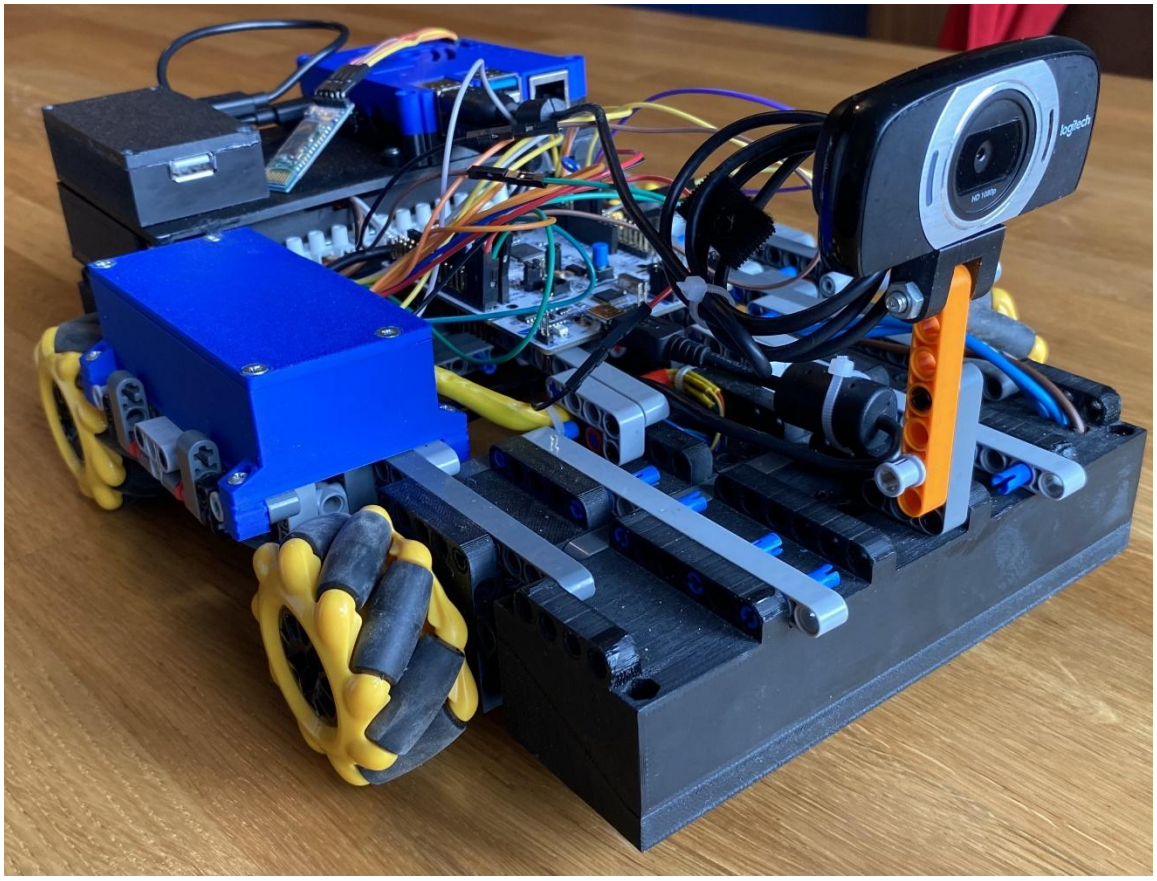
Jakub S.

Magdalena M.

Michał O.

Łukasz R.

1.Przedstawienie projektu



2.Inspiracja

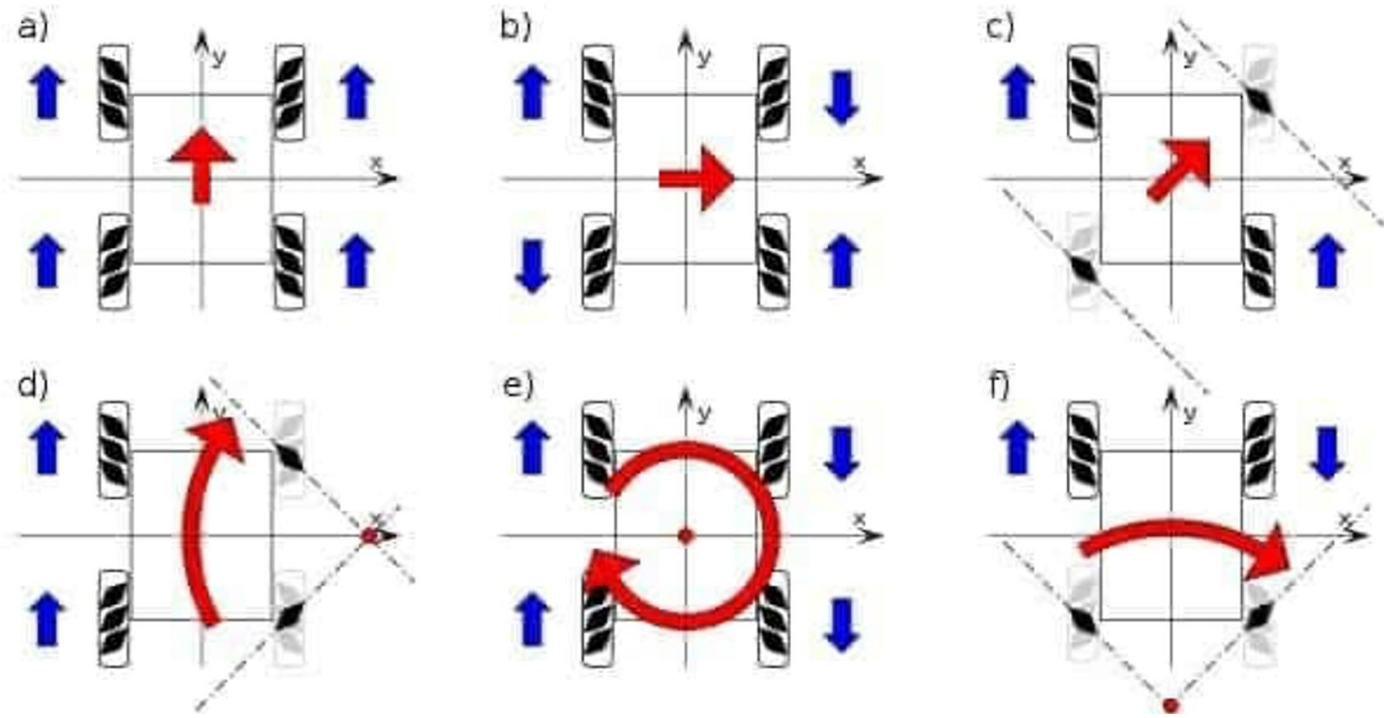
Inspiracją do stworzenia projektu było problem z jakim spotykają się bezzałogowe misje kosmiczne, gdzie na powierzchni księżyca lub marsa ląduje łazika w innym miejscu niż kapsuła/rakieta, w której łazik ma umieścić próbki gleby. Łazik w tedy musi przeskanować całą okolicę w celu znalezienia rakiety.

Nasze autko odwzorowuje ten właśnie problem z tym, że rakietę zastąpiliśmy piłeczką tenisową, autko skanuje teren w celu wykrycia celu następnie dojeżdża do niego i zatrzymuje się przed nim.

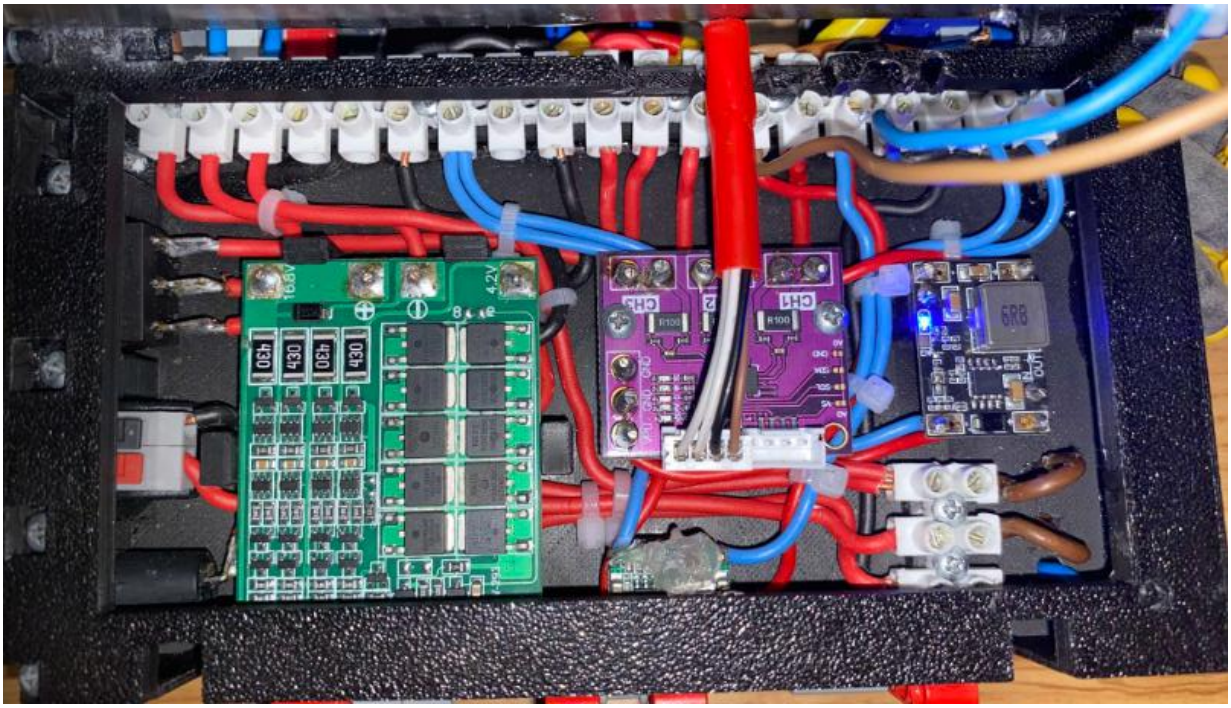
3.Budowa

Aby konstrukcja naszego projektu była modułowa postawiliśmy na wykorzystanie klocków Lego Technic wraz z drukiem 3D.

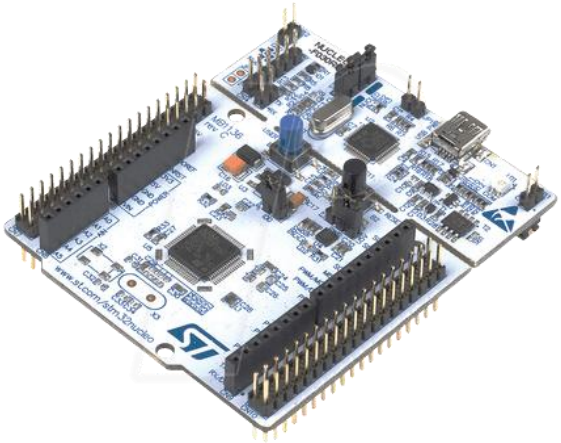
Do budowy układu napędu wykorzystaliśmy kompaktowe silniki BLDC ze zintegrowanym sterownikiem i enkoderem oraz koła omnikierunkowe, które zapewniają więcej sposobów lokomocji niż aniżeli standardowe koła co można zobaczyć na grafice umieszczonej poniżej.



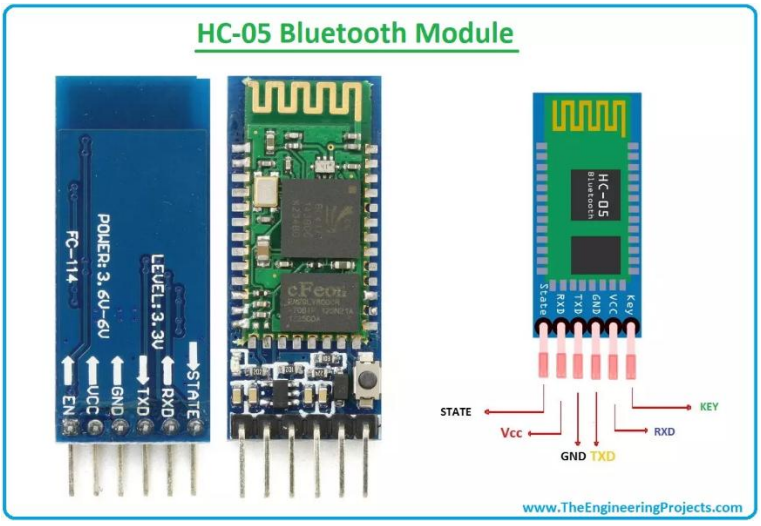
Do wykonania układu zasilania wykorzystaliśmy 8 ogniw 18650 zgrzanych w dwa pakiety 2S2P połączone szeregowo wraz z układem BMS, dwie przetwornice step-down 16,8V/12V, 16,8V/5V oraz trzy kanałowym modułem pomiaru prądu i napięcia INA3221 z interfejsem I2C.



Układ sterowania został się zaprojektowany w oparciu o architekturę Master-Slave przy pomocy mikrokontrolera STM32 i mikrokomputera Raspberry Pi, komunikacja odbywa się przy pomocy interfejsu UART.

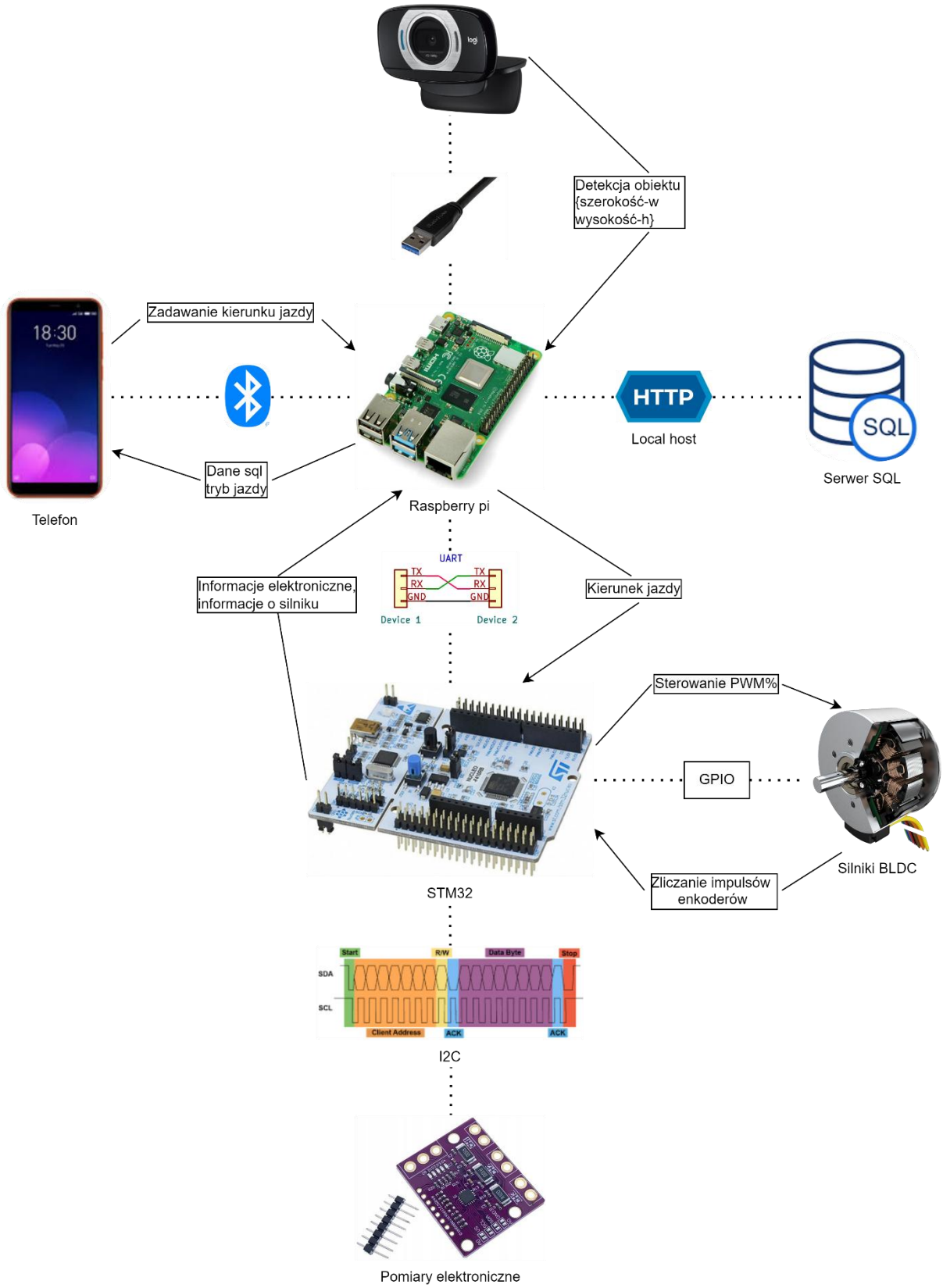


Ostatnim, lecz niezbędnym podzespołem była wizja wraz z komunikacją z aplikacją na telefonie. Do możliwości wykrywania piłeczki użyliśmy kamery Logitech C165, a do bezprzewodowego łączenia się z telefonem użyliśmy moduł Bluetooth HC-05.



4.Połączenie podzespołów

Raspberry Pi 4B poprzez interfejs UART komunikuje się z STM32 F407G, który poprzez piny GPIO łączy się z silnikami BLDC poruszającymi kołami omnikierunkowymi. Po magistrali I2C STM32 łączy się z czujnikiem pomiaru prądu i napięcia INA3221. Dane z Raspberry Pi są po protokole HTTP przekazywane do bazy SQL. Kamera Logitech C615 jest dołączona za pomocą złącza UBS. Telefon w celu sterowania komunikuje się za pomocą bluetooth.



5.Sterowanie

Założeniem układu sterowania jest możliwość sterownia autka w dwóch trybach:

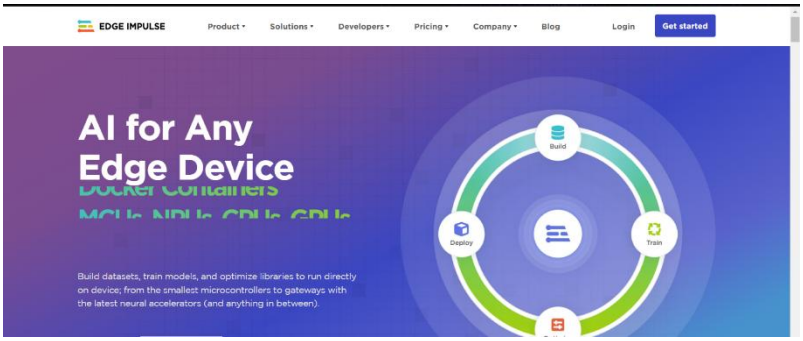
- **Tryb ręczny**: Sterowanie odbywa się za pomocą telefonu, który łączy się z pojazdem przez Bluetooth.
- **Tryb autonomiczny**: Pojazd używa kamery do skanowania otoczenia w celu wykrycia piłki. Po jej zlokalizowaniu, pojazd zbliża się do obiektu i zatrzymuje się przed nim. Ten tryb opiera się na modelu sztucznej inteligencji.

Układ sterowania ma również na celu odczyt informacji ze wszystkich modułów używanych w projekcie i enkoderów umieszczonych w silnikach w celu odpowiedniego ich sterowania.

6.Tworznie modelu AI

Nasz model ma za zadnie wykrywanie obiektu oraz zwracanie informacji o jego położeniu, wielkości oraz wynik precyzji wykrycia [%].

Do stworzenia modelu wykorzystaliśmy program: **Edge Impulse** https://edgeimpulse.com/

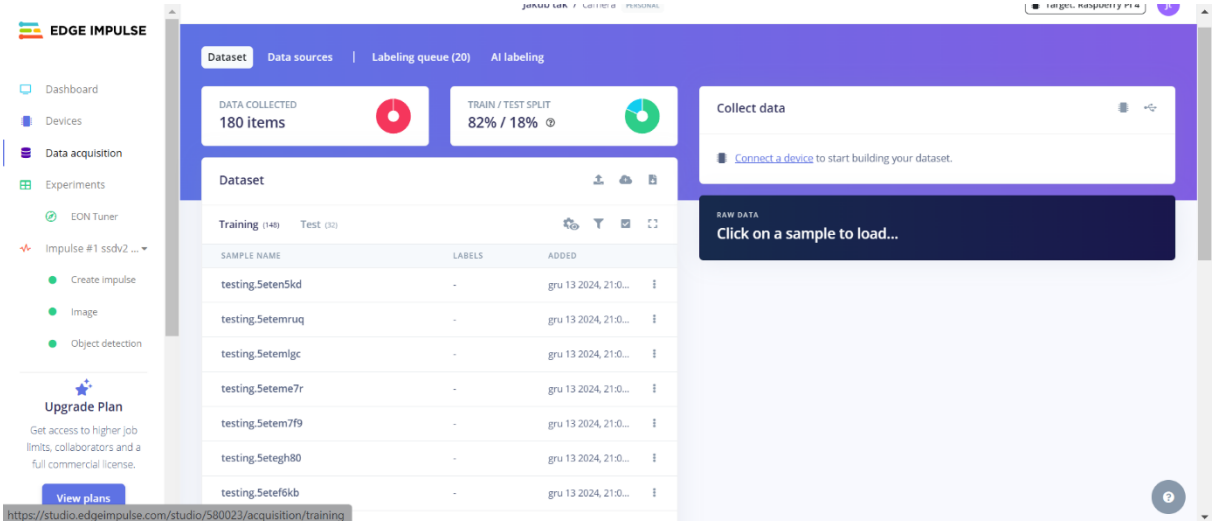


Krok 1.

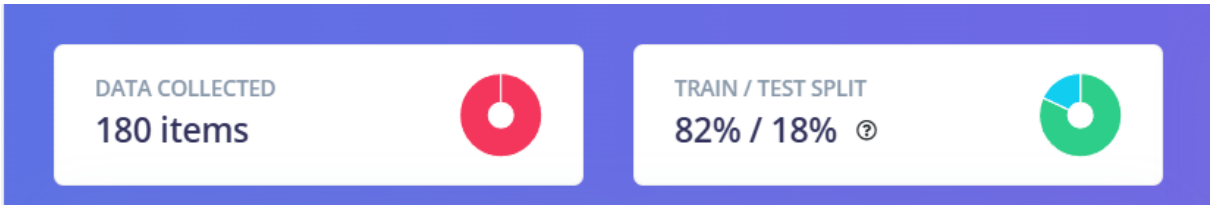
- Wykonanie jak największej ilości zdjęć obiektowi, który chcemy wykryć.

Krok 2.

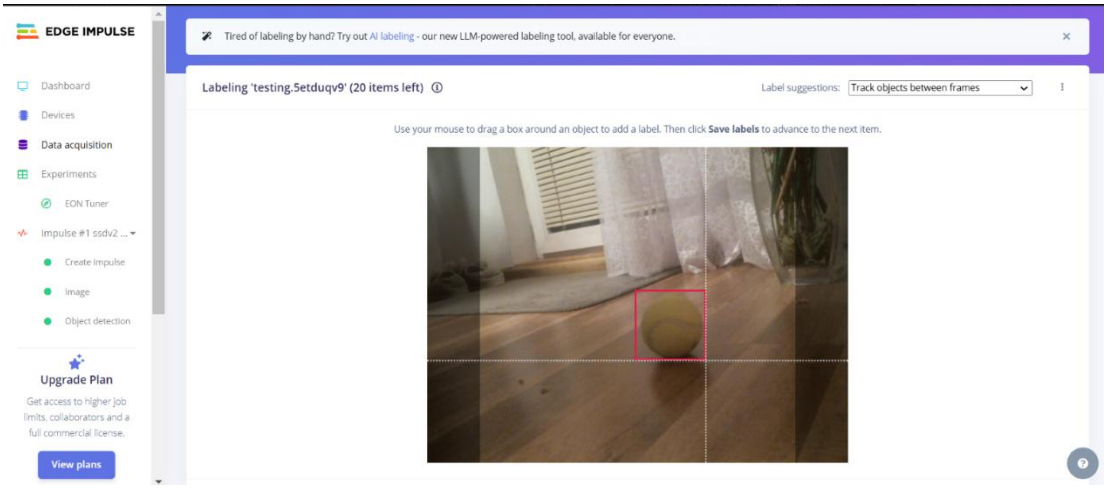
- Załadowanie zdjęć na stronę projektu Edge Impulse

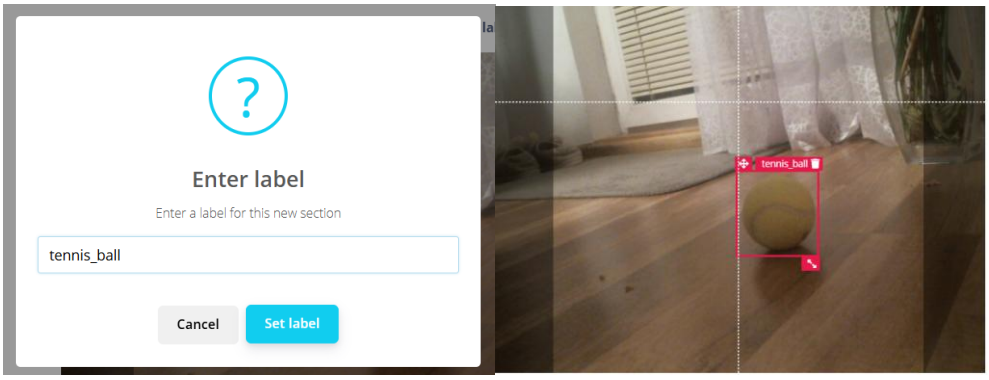


Ustawiamy proporcje zdjęć do trenowania i do testu:



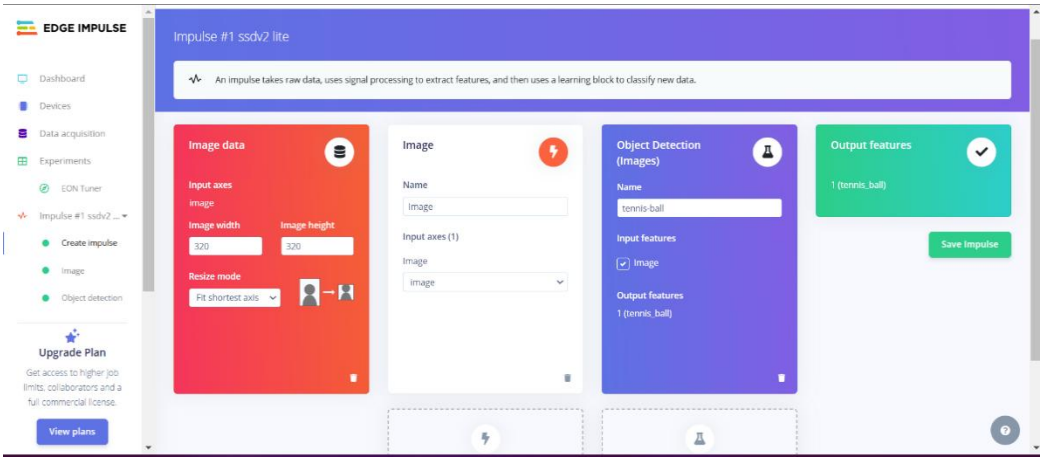
Następnie wykonujemy etykietowanie i podpis obiektu na zdjęciu:

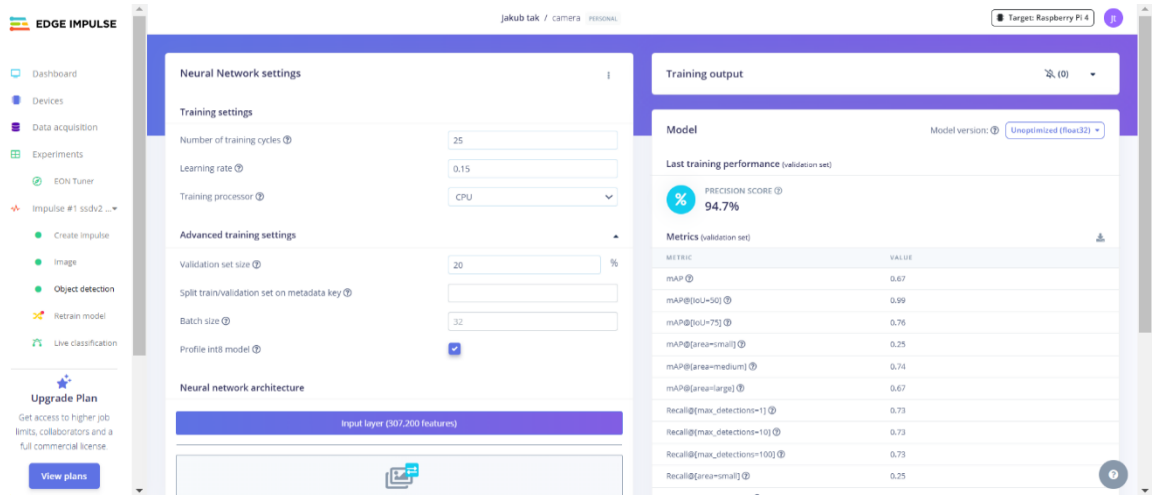




Dzięki programowi Edge Impulse nie musimy każdego zdjęcia ręcznie etykietować, wystarczy kilka-kilkanaście zdjęć zaetykietować ręcznie po czym program sam wykonuje za nas resztę zdjęć. Przeglądamy wszystkie zdjęcia czy zostały prawidłowo zaetykietowane i zapisujemy.

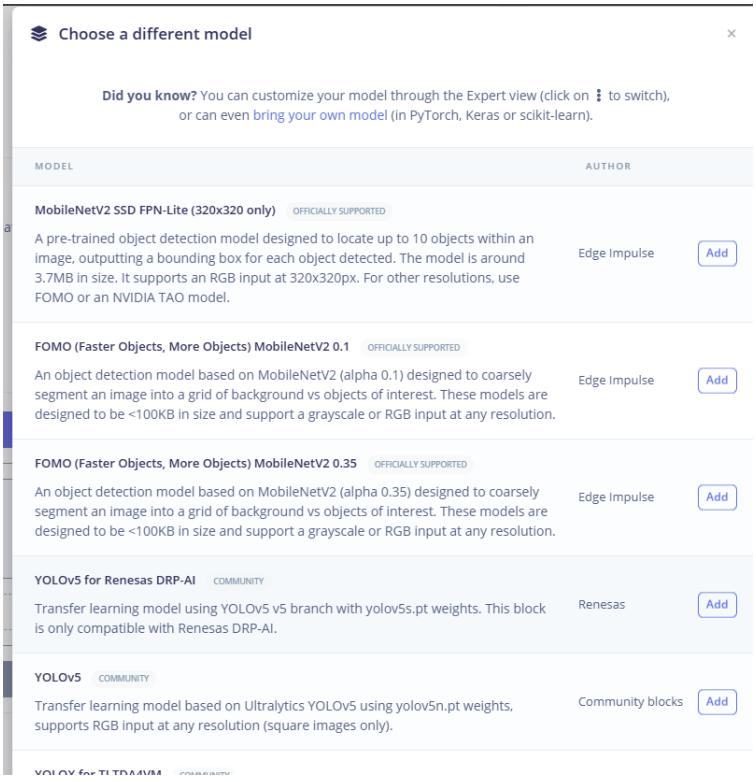
Następnie ustawiamy parametry trenowania modelu:



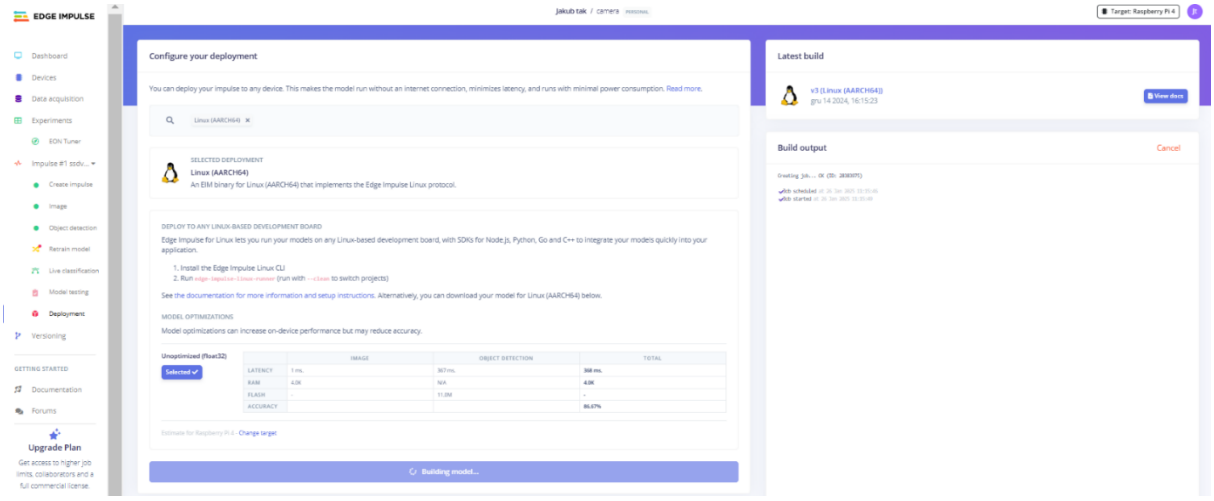


Wybieramy odpowiedni model:

Dla Raspberry Pi 4, która nie posiada dużej mocy obliczeniowej odpowiednim modelem jest MobileNetV2 SSD FPN-Lite.



Po utworzeniu modelu możemy pobrać gotowy plik modelu:



7.Konfiguracja mikrokomputera Raspberry Pi wraz z opisem kodu

●Na Raspberry pi4 instalujemy potrzebne rzeczy na pamiętając, że cały projekt tworzymy w wirtualnym środowisku:

```
python3 -m venv myenv
source myenv/bin/activate
```

```
sudo apt update
sudo apt install python3-opencv
pip install numpy
```

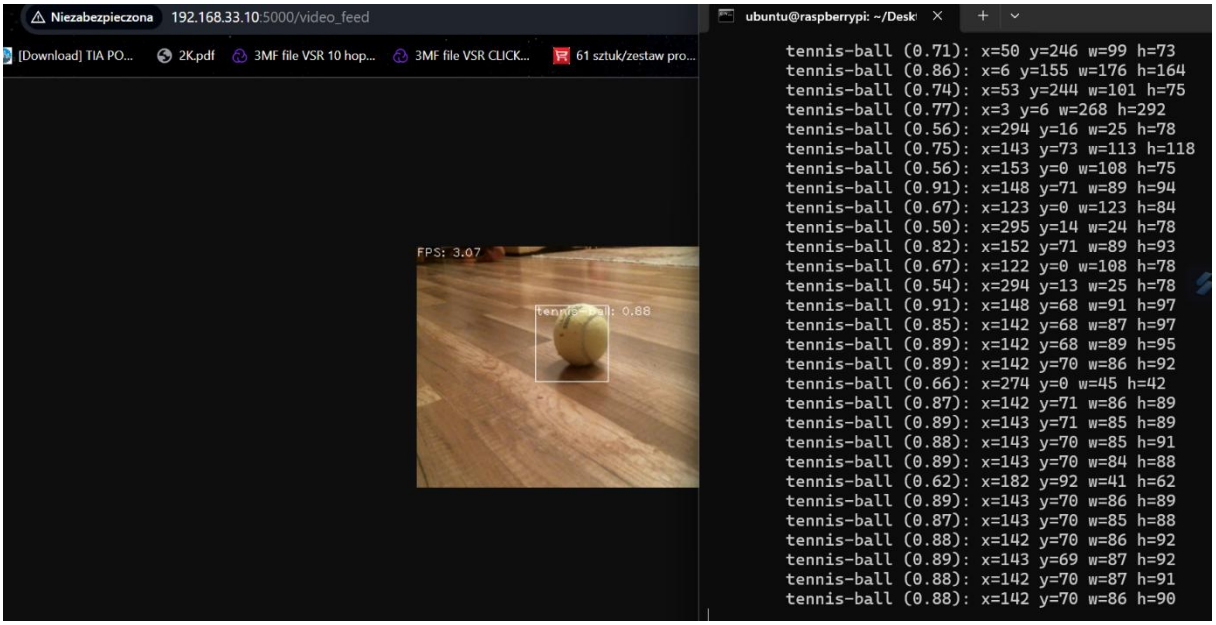
```
pip install flask
```

```
pip install edge-impulse-linux
```

```
pip install numpy
pip install imutils
```

Opis pliku camera.py

Plik camera.py jest kluczowym modułem odpowiedzialnym za obsługę kamery, przetwarzanie obrazu za pomocą modelu Edge Impulse oraz strumieniowanie wideo w czasie rzeczywistym poprzez serwer Flask. Poniżej znajduje się szczegółowy opis wszystkich funkcji i mechanizmów w tym pliku.



●Główne funkcje i moduły

Plik korzysta z poniższych modułów:

- **cv2 (OpenCV)**: Do obsługi kamery i przetwarzania obrazu.
- **Flask**: Do tworzenia serwera HTTP, umożliwiającego strumieniowanie wideo.
- **ImageImpulseRunner**: model Edge Impulse do przetwarzania obrazu i detekcji obiektów.

●Globalne zmienne

Plik używa kilku zmiennych globalnych:

- camera: Obiekt kamery (inicjalizowany za pomocą OpenCV).
- runner: Obiekt modelu Edge Impulse (inicjalizowany za pomocą ImageImpulseRunner).
- bbox_data: Słownik przechowujący dane o wykrytym obiekcie (np. etykieta, współrzędne, rozmiar, poziom pewności).

● Kluczowe funkcje

initialize_system()

- **Cel**: Inicjalizuje kamerę i model Edge Impulse.

```
# Inicjalizacja modelu Edge Impulse
runner = ImageImpulseRunner(model_file)
runner.init()
print("Model Edge Impulse został pomyślnie zainicjalizowany.")

# Inicjalizacja kamery
camera = cv2.VideoCapture(0)
camera.set(cv2.CAP_PROP_FRAME_WIDTH, cam_width)
camera.set(cv2.CAP_PROP_FRAME_HEIGHT, cam_height)

if not camera.isOpened():
    raise Exception("Nie udało się otworzyć kamery.")
print("Kamera została pomyślnie zainicjalizowana.")
```

- **Argumenty**:
  - model_file: Ścieżka do pliku modelu Edge Impulse (.eim).
  - cam_width, cam_height: Wymiary obrazu z kamery.
- **Działanie**:
  - Ładuje model Edge Impulse.
  - Inicjalizuje kamerę i ustawia jej wymiary.
  - Tworzy pusty słownik bbox_data na dane o wykrytym obiektcie.

get_bbox_data()

```
# Funkcja do pobierania danych z kamery
def get_bbox():
    #Pobiera dane z get_bbox_data i zwraca w uporządkowanym formacie.
    data = get_bbox_data()
    # Zabezpieczenie przed brakującymi kluczami
    return {
        "label":        data.get("label", None),
        "confidence":   data.get("confidence", 0.0),
        "x":            data.get("x", 0),
        "y":            data.get("y", 0),
        "width":        data.get("width", 0),
        "height":       data.get("height", 0)
    }
```

- **Cel**: Zwraca aktualny stan zmiennej bbox_data.
- **Zastosowanie**: Funkcja używana do pobierania danych o wykrytych obiektach.

generate_frames()

- **Cel**: Przetwarza obrazy z kamery w pętli i generuje je jako strumień wideo.
- **Działanie**:

- Odczytuje klatki z kamery i co trzecią klatkę przetwarza za pomocą modelu Edge Impulse. Ma to na celu odciążenie procesora raspberry pi4.

```
frame_counter += 1
```

```
if frame_counter % 3 == 0:  # Infer every 3rd frame
```

- Wykrywa obiekty na obrazie (np. "tennis-ball").

```
try:
    res = runner.classify(features)
    bboxes = res['result'].get('bounding_boxes', [])
```

- Aktualizuje bbox_data na podstawie wykrytego obiektu.
- Rysuje prostokąt wokół wykrytego obiektu i wyświetla wynik na obrazie, wykorzystując do tego działania bibliotekę openCV.

```
# Draw bounding box for the highest-confidence object
b_x0, b_y0 = best_bbox['x'], best_bbox['y']
b_x1, b_y1 = best_bbox['x'] + best_bbox['width'], best_bbox['y'] + best_bbox['height']
print('\t%s (%.2f): x=%d y=%d w=%d h=%d' % (best_bbox['label'], best_bbox['value'], best_bbox['x'],
                                            best_bbox['y'], best_bbox['width'], best_bbox['height']))
cv2.rectangle(img, (b_x0, b_y0), (b_x1, b_y1), (255, 255, 255), 1)
cv2.putText(img, f"{best_bbox['label']}: {round(best_bbox['value'], 2)}",
            (b_x0, b_y0 + 12), cv2.FONT_HERSHEY_PLAIN, 1, (255, 255, 255))
```

- Ogranicza liczbę klatek na sekundę (FPS).

- **Wynik**:
  - Zwraca ramki w formacie MJPEG, które można strumieniować za pomocą serwera Flask.

start_flask()

```
#Uruchamia serwer Flask w osobnym wątku.
def start_flask():
    try:
        app.run(host="0.0.0.0", port=5000, debug=False)
    except Exception as e:
        print(f"Błąd w serwerze Flask: {e}")
```

- **Cel**: Uruchamia serwer Flask, aby strumieniować wideo w przeglądarce.
- **Działanie**:
  - Uruchamia aplikację Flask na domyślnym porcie 5000.
  - Strumień wideo jest dostępny pod adresem http://localhost:5000.

close_resources()

```
# Funkcja do zamykania zasobów
def close_resources():
    #Zamyka kamerę i model Edge Impulse.
    global camera, runner
    if camera is not None:
        camera.release()
        print("Kamera została zamknięta.")
    if runner is not None:
        runner.stop()
        print("Model Edge Impulse został zamknięty.")
    cv2.destroyAllWindows()
    print("Zasoby zostały pomyślnie zamknięte.")
```

- **Cel**: Zamyka wszystkie zasoby używane przez kamerę i model Edge Impulse.

Architektura systemu

Przepływ danych

1. Kamera zbiera obrazy w czasie rzeczywistym.
2. Co trzecią klatkę model Edge Impulse przetwarza i klasyfikuje obiekty.
3. Informacje o wykrytych obiektach są zapisywane w bbox_data, następnie wybierany jest obiekt o najwyższym współczynniku ‘confidence’, dzięki czemu samochodzik podąża za jednym obiektem oraz eliminuje ewentualne błędy w wykryciu.

```
best_bbox = max(bboxes, key=lambda bbox: bbox['value'])
```

4. Przetworzone obrazy (z zaznaczonym obiektem) są strumieniowane za pomocą Flask.

Potrzebne pliki do instalacji:

```
npm install -g edge-impulse-linux
```

```
sudo apt install ffmpeg -y
```

Opis pliku main_threading.py

Plik main_threading.py to główny moduł zarządzający działaniem całego systemu. Jego głównym zadaniem jest uruchamianie i synchronizacja wielu wątków obsługujących różne komponenty, takie jak kamera, sterowanie, komunikacja UART, czy baza danych. Plik implementuje także logikę sterowania pojazdem w trybie manualnym i automatycznym.

1. Główne funkcje i moduły

Plik korzysta z poniższych modułów:

- **threading**: Do zarządzania wątkami.
- **time**: Do wprowadzenia opóźnień i pomiaru czasu.
- Importowane moduły:
  - **Camera**: Zawiera funkcje związane z kamerą, Flaskiem i przetwarzaniem wideo.
  - **sql_&_stm**: Obsługuje komunikację UART i połączenie z bazą danych MySQL.
  - **hc05_21_01**: Zarządza komunikacją z modułem Bluetooth (HC-05).
- **lock**: Obiekt synchronizacji (mutex) do zapewnienia bezpieczeństwa wątków.
- **running**: Flaga sterująca działaniem wątków.

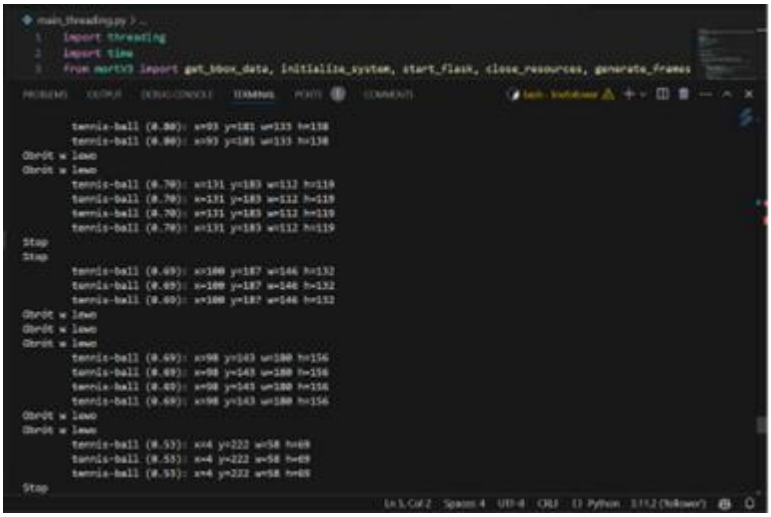
3. Kluczowe funkcje

camera_processing()

- **Cel**: Przetwarza strumień wideo w pętli, generując klatki i wykrywając obiekt.

control_processing()

- **Cel**: Obsługuje logikę sterowania pojazdem w trybach manualnym i automatycznym.
- **Działanie**:
    - **Manual Mode**:
        - Pobiera dane sterowania z Bluetooth (get_phone_box()).
        - Na podstawie kierunku (direction) i prędkości (velocity) generuje komendy (np. "A" – szybki przód, "C" – obrót w lewo).
    - **Auto Mode**:



        - Używa danych z kamery (bbox_data) do podejmowania decyzji.
        - Na przykład: Jeśli wykryto piłkę tenisową, porusza się w jej kierunku.
    - Komendy są aktualizowane w zmiennej data_to_stm i wysyłane do STM32.

main()

Cel: Główna funkcja uruchamiająca cały system.

- Tworzy poszczególne wątki:

```
camera_thread   = None
flask_thread    = None
control_thread  = None
uart_thread     = None
sql_thread      = None
hc_thread       = None
receive_thread  = None
```

- Wykonuje funkcje inicjalizujące podzespoły przed uruchomieniem wątków
- Uruchamia wątki

```
# Tworzenie wątków
camera_thread   = threading.Thread(target=camera_processing)
flask_thread    = threading.Thread(target=start_flask)
control_thread  = threading.Thread(target=control_processing)
uart_thread     = threading.Thread(target=send_message)
sql_thread      = threading.Thread(target=get_connection_to_SQL)
hc_thread       = threading.Thread(target=uart_hc_processing)
receive_thread  = threading.Thread(target=receive_message)
```

- Wprowadza pętlę główną programu, która trwa, dopóki użytkownik nie przerwie działania (Ctrl+C).

```
# Główna pętla programu
while True:
    time.sleep(1)
```

- Obsługuje zamknięcie całego systemu oraz wszystkich wątków

```
finally:
    print("Zamykanie zasobów...")
    running = False

    # Zatrzymywanie wątków (jeśli istnieją)
    if camera_thread is not None:
        camera_thread.join()
    if flask_thread is not None:
        flask_thread.join()
    if control_thread is not None:
        control_thread.join()
    if uart_thread is not None:
        uart_thread.join()
    if sql_thread is not None:
        sql_thread.join()
    if hc_thread is not None:
        hc_thread.join()
    if receive_thread is not None:
        receive_thread.join()
```

- Zamyka zasoby (kamera, UART, MySQL).

```
close_resources()          #Zamknięcie kamery, flaska
close_port()               #UART close
close_connection_to_SQL() #Zamknięcie połączenia z bazą danych
close_uart_hc()            #Zamknięcie portu UART HC`
print("Zakończono działanie programu.")
```

Problemy:

Kiedy wątki współdzielą dane, może dojść do tzw. **"race condition"**, czyli sytuacji, w której dwa lub więcej wątków próbują jednocześnie modyfikować te same dane.

Aby temu zapobiec, używamy **Lock** – mechanizmu blokady, który pozwala na sekwencyjny dostęp do współdzielonych zasobów. Przykładem może być zmienna ‘data_to_stm’, która jest używana w dwóch różnych funkcjach, w oddzielnych wątkach.

```
    uart_thread      = threading.Thread(target=send_message)
    sql_thread       = threading.Thread(target=get_connection_to_SQL)
```

```
# Funkcja wysyłająca dane do STM32
def send_message(data_to_stm):
    global running
    while running:
        try:
            if ser.is_open:
                with lock:
                    command = data_to_stm.get('command', '0')  # Domyślnie '0' (Stop)
                    ser.write(f"{command}".encode())
                    print(f"Wiadomość do STM32 wysłana!: {command}")
```

```
# Funkcja zapisująca dane do MySQL
def get_connection_to_SQL(data_to_stm):
    global connection, received_data_from_stm, command#, data_to_stm
    while running:
        try:
            if connection and connection.is_connected():
                with lock:
                    if received_data_from_stm:
                        cursor = connection.cursor()
                        data = received_data_from_stm
                        command = data_to_stm.get('command', '0')  # Pobierz komendę z data_to_stm
                        print(f"Zapisuję dane do MySQL: {command}")
                        #print(f"Zapisuję dane do MySQL: {data}")
```

8.Konfiguracja mikrokontrolera STM32 wraz z opisem kodu

Przekładnia znajdująca się na wale silnika ma przełożenie 1:45, enkoder znajduje się na wale silnika daje nam 6 impulsów na każdy obrót wału silnika, nie daje jednak informacji o kierunku obrotów. Proste obliczenia matematyczne dostarczają informację, że na jeden obrót wału przekładni przypada 270 impulsów enkodera. Regulator będzie operował na danych surowych, czyli na ilości impulsów enkodera.

Przewody czujnika.

Podłączenie

PRZEWÓD	OPIS
czarny	Napięcie zasilania -.
czerwony	Napięcie zasilania +.
żółty	Wyjście częstotliwościowe enkodera.
biały (szary)	Regulacja obrotów - sygnał PWM o amplitudzie 5 V
niebieski	Zmiana kierunku obrotów: +5 V / GND.

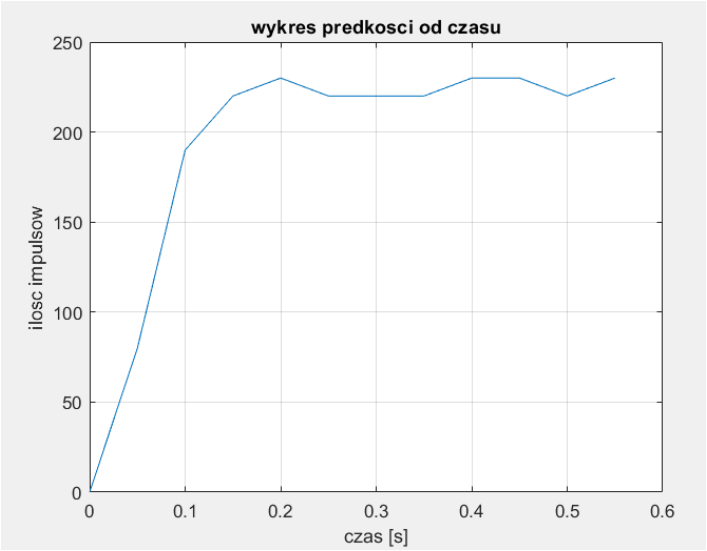
Konwerter stanów logicznych jest potrzebny, aby osiągnąć pełny zakres obrotów silnika, ponieważ mikrokontroler ma sygnał PWM o amplitudzie 3.3V a sterownik silnika BLDC odczytuje informacje o prędkości jako średnią wartość sygnału.

Identyfikacja modelu silnika:

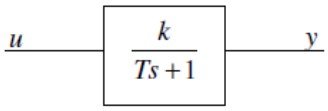
- Skrypt w Matlabie ze zmierzoną prędkością w jednostce ilości impulsów enkodera na sekundę, pomiary prędkości były dokonywane co 50ms. Wypełnienie sygnału PWM wynosi 100%.

```
odczyt=[0;80;190;220;230;220;220;220;230;230;220;230];

t=0.05*[0;1;2;3;4;5;6;7;8;9;10;11];

plot(t,odczyt);
grid
hold on;
xlabel('czas [s]');
ylabel('ilosc impulsow');
title('wykres predkosci od czasu');

```

- Wykres prędkości od czasu



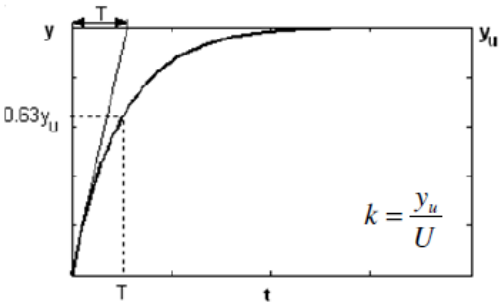
Wykres prędkości aproksymowanej inercją pierwszego rzędu

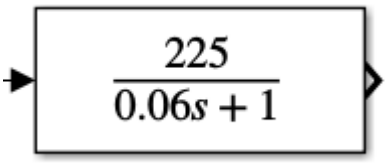


$$y(t) = kU(1 - e^{-\frac{t}{T}})$$

$$y(t \to \infty) = y_u = kU$$

$$y(t = T) = kU(1 - e^{-1}) \cong 0.63kU = 0.63y_u$$

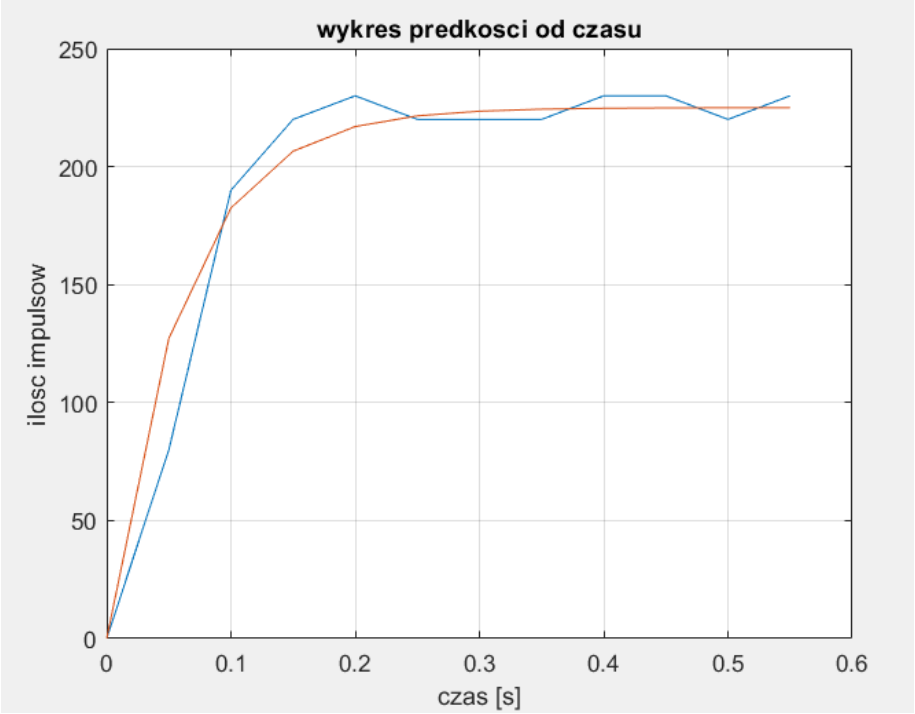




●Kod w Matlabie

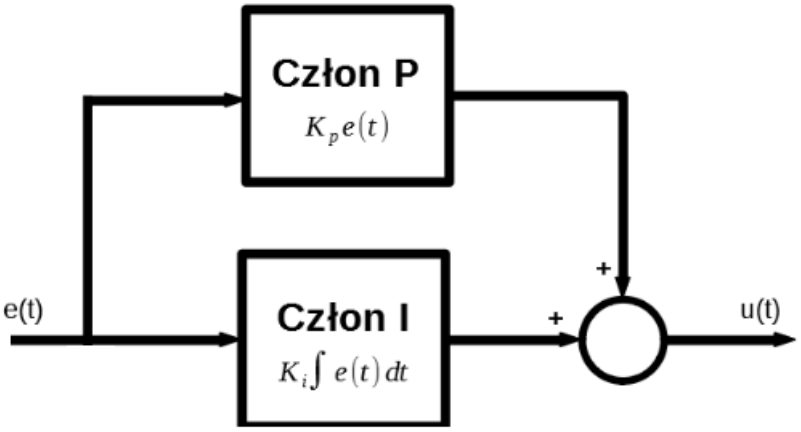
```
13  t1=0.05*(0:1:11);
14  sys=tf(225,[0.06 1]);
15  [y,t2] = step(sys,t1);
16  plot(t2,y);
17  grid on;
```

●Wykres prędkości od czasu

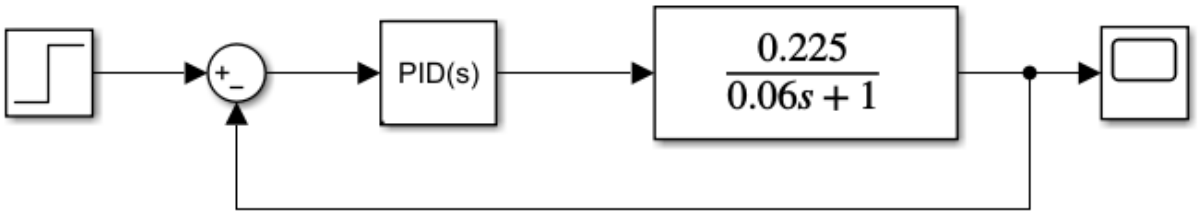


Dobranie współczynników regulatora prędkości w Simulinku wraz z symulacją

● Schemat blokowy regulatora PI



● Schemat blokowy obiektu wraz z regulatorem



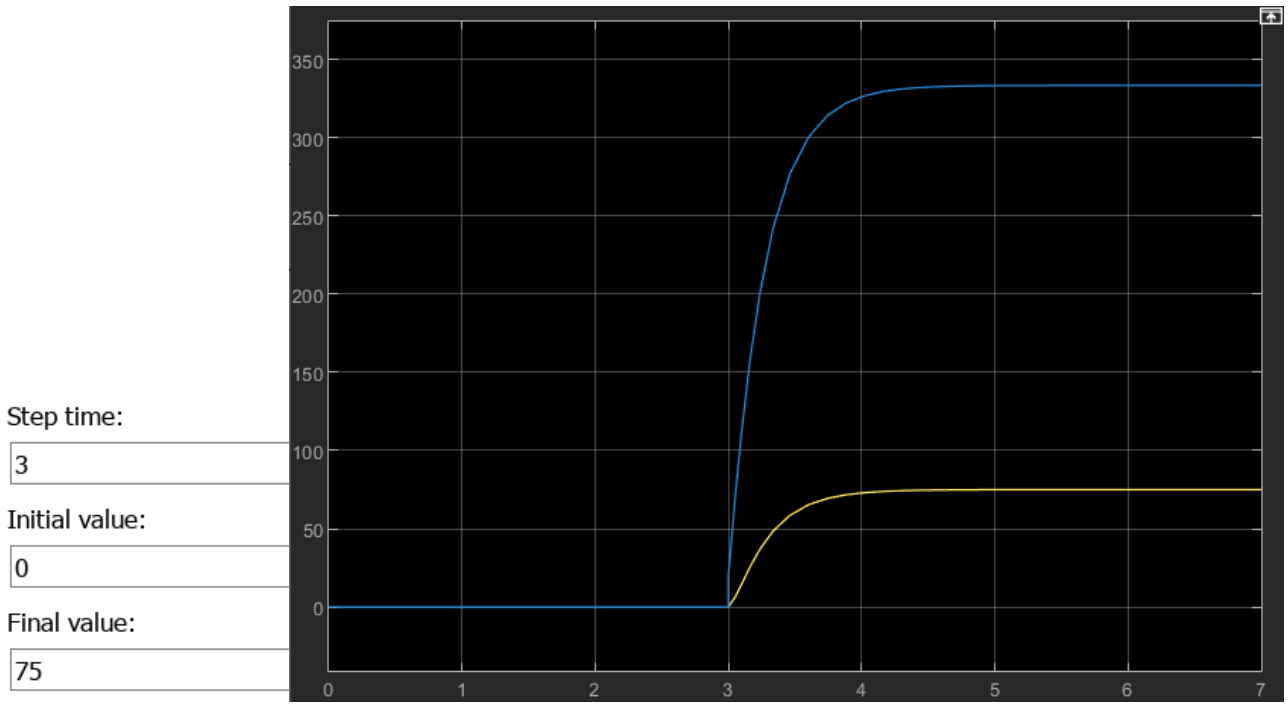
Współczynnik K wynosi 0.225, ponieważ sygnał PWM będzie dzielony na 1000 części aby uzyskać lepszą dokładność w sterowaniu prędkością

● Dobrane współczynniki regulatora PI

Proportional (P): 0.267

Integral (I): 14.1

● Otrzymany przebieg symulacji wraz z sygnałem sterującym



Wnioski: Dobrane współczynniki zapewniają dobrą odpowiedź obiektu, aby uzyskać prędkość wynoszącą 75 sygnał PWM musi mieć wartość ok 32%.

Implementacja kodu w języku C na mikrokontroler STM32

- Pierwszym krokiem jest obliczanie prędkości silników

Stworzono 4 zmienne strukturalne(obiekty) – cztery koła, w których będą przechowywane wszystkie informacje od kół wraz z numerami portów, pinów, timerów i kanałów.

```
typedef struct //toworze strukture-WHEEL_Type sa tutaj wszytkie parametry kola
{
    MOTOR_STATE_Type      MOTOR_STATE;      //to stan silnika
    MOVE_STATE_Type       MOVE_STATE;       //to stan poruszania sie autka
    ENCODER_Type          encoder_data;     //to struktura w której są wszystkie zmienne potrzebne do obsługi enkodera
    TIMER_Type            timer_data;       //to struktura w której mam wszystkie kanały i timery
    PINOUT_Type           pinout_data;      //to struktura w której mam wszystkie piny i porty potrzebne do koła
    PID_Type              pid_data;         //to struktura w której mam wszystkie dane od reg. pid
    RAMP_Type             ramp_data;        //to struktura w której mam wszystkie dane do rampy prędkosci

    int                   set_velocity;     //to predkość jaką ustawiam
    int                   motor_siganl;     //to sygnal jaki jest ustawiony na silniku
}WHEEL_Type;
```

A poniżej znajdują się wszystkie pozostałe struktury użyte w strukturze powyżej

```
typedef enum //tworzę nowy typ zmiennej enumerycznej-MOTOR_STATE_Type - czyli stan w jakim znajduje sie dane koło
{
   IDLE_1=0,              //to stan jałowy
   DRIVE,                 //to stan jazdy
   BRAKE                  //to stan hamowania
}MOTOR_STATE_Type;

typedef enum //tworzę nowy typ zmiennej-MOVE_STATE_Type czyli sposob w jaki auto sie porusza
{
   IDLE_2= 0,             //to stan nie ustalony
   FORWARD,               //to stan jazdy do przodu/tyłu
   RIGHT_SIDE,            //to stan jazdy w stronę prawą/lewą
   //DIAGONAL_P,            //to stan jazdy po ukosie w prawą stronę do przodu/tyłu
   //DIAGONAL_L,            //to stan jazdy po ukosie w lewą strone do przodu/tyłu
   AROUND,                //to stan jazdy wokół własnej osi
   //AROUND_F,              //to stan jazdy gdzie obracamy sie wokoł osi z przodu
   //AROUND_B,              //to stan jazdy gdzie obracamy sie wokoł osi z tyłu
   //AROUND_R,              //to stan jazdy gdzie obracamy sie wokoł osi z prawej strony
   //AROUND_L               //to stan jazdy gdzie obracamy sie wokoł osi z lewj strony
}MOVE_STATE_Type;

typedef struct //toworze strukture-ENCODER_Type sa tutaj wszytkie parametry dla enkodera
 {
    int Value;             //tworzę zmienną która pamięta ostanią wartość
    uint32_t TotalTurns;       //tworze zmienną która przechowuje wszytkie obroty które zrobił silnik
    float CurrentVelocity;     //tworze zmienną ktora przechowuje aktualną prędkosc  znak to kierunek obrotów
    float V1;                  //to wartości predkosci o 1,2,3 próbki wcześniej będą służyć do średniej aby wyelminować skoki prędkosci
    // tutaj bedą jeszcze inne parametry które można obliczyć z enkdera
 }ENCODER_Type;

 typedef struct //toworze strukture-TIMER_Type sa tutaj wszytkie numery timerów i kanały
   {
    TIM_HandleTypeDef     *_htim_PWM;          //to wskaźnik na timer od pwma
    uint32_t              _channel_PWM;        //to wskaźnik na kanał od pwma
   }TIMER_Type;

  typedef struct //toworze strukture-PINOUT_Type sa tutaj wszytkie numery pinów i portów dla jednego koła
  {
      GPIO_TypeDef*       DirectionPort;     //to wskaźnik na port od kierunku jazdy
      uint16_t            DirectionPin;      //to pin od kierunku jazdy

      GPIO_TypeDef*       EncoderPort;     //to wskaźnik na port od enkodera
      uint16_t            EncoderPin;      //to pin od enkodera
  }PINOUT_Type;

  typedef struct //tworzę strukture PID_Type są tutaj wszystkie dane potrzebne do reg. PID
  {
      float               PidSignal;     //to sygnał obliczony przez nas regulator PID
      float               Deviation;     //to uchyb regulacji - wartość zadana - wartość bieżąca
      float               Integral;      //to całka z uchybu
      float               Derivative;    //to pochodna z uchybu
  }PID_Type;

  typedef struct //tworze strukture RAMP_Type są tutaj wszystkie dane potrzebne do obliczania rampy
  {
     int          StartSpeed;         //to prędkość początkowa
     int          FinalSpeed;         //to predkosć do której chcę dojść
     int          LastFinalSpeed;     //to ostatnia predkosc do której chcieliśmy dojść
     float        Delta;              //to przyrost predkosci z kolejnym krokiem
     float        RisingTime;         //to czas narastania w sekundach
  }RAMP_Type;
```

Teraz stworzono 4 obiekty jako zmienna globalna – plik main.c

```
65 WHEEL_Type Front_Right;  //tworzę 4 obiekty - 4 koła, jako obiekty globalne
66 WHEEL_Type Front_Left;
67 WHEEL_Type Back_Right;
68 WHEEL_Type Back_Left;
```

Oraz je zainicjowano wraz z timerami

```
136   InitFunction(&Front_Left, &Front_Right, &Back_Left, &Back_Right);   //to wywołanie funkcji inicjalizujacej 4 koła-są w niej też starty timerów

102 void InitFunction(WHEEL_Type* FL,WHEEL_Type* FR,WHEEL_Type* BL,WHEEL_Type* BR)//funkcja incalizujaca 4 obiekty
103 {
104     FL->timer_data._htim_PWM=&htim3;
105     FL->timer_data._channel_PWM=TIM_CHANNEL_2;
106
107     FR->timer_data._htim_PWM=&htim3;
108     FR->timer_data._channel_PWM=TIM_CHANNEL_1;
109
110     BL->timer_data._htim_PWM=&htim3;
111     BL->timer_data._channel_PWM=TIM_CHANNEL_3;
112
113     BR->timer_data._htim_PWM=&htim3;
114     BR->timer_data._channel_PWM=TIM_CHANNEL_4;
115
116     //to piny od enkdera
117     FL->pinout_data.EncoderPin=EN_1_Pin;
118     FL->pinout_data.EncoderPort=EN_1_GPIO_Port;
119
120     FR->pinout_data.EncoderPin=EN_2_Pin;
121     FR->pinout_data.EncoderPort=EN_2_GPIO_Port;
122
123     BL->pinout_data.EncoderPin=EN_3_Pin;
124     BL->pinout_data.EncoderPort=EN_3_GPIO_Port;
125
126     BR->pinout_data.EncoderPin=EN_4_Pin;
127     BR->pinout_data.EncoderPort=EN_4_GPIO_Port;
128
129     //to piny i porty od kierunku
130     FL->pinout_data.DirectionPin=KIER_1_Pin;
131     FL->pinout_data.DirectionPort=KIER_1_GPIO_Port;
132
133     FR->pinout_data.DirectionPin=KIER_2_Pin;
134     FR->pinout_data.DirectionPort=KIER_2_GPIO_Port;
135
136     BL->pinout_data.DirectionPin=KIER_3_Pin;
137     BL->pinout_data.DirectionPort=KIER_3_GPIO_Port;
138
139     BR->pinout_data.DirectionPin=KIER_4_Pin;
140     BR->pinout_data.DirectionPort=KIER_4_GPIO_Port;
141
142     //teraz ustawiam domyślne tryby silnika i sposoby poruszania
143     FL->MOTOR_STATE=DRIVE;                          //domyślny tryb silnika to jazda-->raczej tutaj nic nie zmienimy
144     FL->MOVE_STATE=FORWARD;                         //domyślny tryb to jazda do przodu
145
146     FR->MOTOR_STATE=DRIVE;
147     FR->MOVE_STATE=FORWARD;
148
149     BL->MOTOR_STATE=DRIVE;
150     BL->MOVE_STATE=FORWARD;
151
152     BR->MOTOR_STATE=DRIVE;
153     BR->MOVE_STATE=FORWARD;
154
155     FL->ramp_data.RisingTime=0.6;
156     FR->ramp_data.RisingTime=0.6;
157     BL->ramp_data.RisingTime=0.6;
158     BR->ramp_data.RisingTime=0.6;
159
160     //teraz startuje timery
161     HAL_TIM_PWM_Start(FL->timer_data._htim_PWM, FL->timer_data._channel_PWM);
162     HAL_TIM_PWM_Start(FR->timer_data._htim_PWM, FR->timer_data._channel_PWM);
163     HAL_TIM_PWM_Start(BL->timer_data._htim_PWM, BL->timer_data._channel_PWM);
164     HAL_TIM_PWM_Start(BR->timer_data._htim_PWM, BR->timer_data._channel_PWM);
165 }
```

Aby obliczyć prędkoSć zliczamy impulsy, które przyszły z enkodera w trybie przerwaniowym

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    if(GPIO_Pin==Front_Left.pinout_data.EncoderPin)//sprawdzam z którego pinu przyszło przerwanie i inkrementuję
    {                                              //odpowiednią zmienne od odpowiedniego koła
        PositionIncrement(&Front_Left);
    }
    if(GPIO_Pin==Front_Right.pinout_data.EncoderPin)
    {
        PositionIncrement(&Front_Right);
    }
    if(GPIO_Pin==Back_Left.pinout_data.EncoderPin)
    {
        PositionIncrement(&Back_Left);
    }
    if(GPIO_Pin==Back_Right.pinout_data.EncoderPin)
    {
        PositionIncrement(&Back_Right);
    }
}
```

```
void PositionIncrement(WHEEL_Type* key)
{
    key->encoder_data.Value++; //inkrementuję wartość impulsów
}
```

Teraz obliczamy finalną prędkość, w Callback-u od timera co 200ms

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)//to callback od przerwania co określony czas
{
    if(htim->Instance==TIM11)//sprawdzam czy przerwanie pochodzi z timera który mnie interesuje - przerwanie co 0.2s
    {
        //HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin); //to część kodu testujacego
        EncoderVelocity(&Front_Left);
        EncoderVelocity(&Front_Right);
        EncoderVelocity(&Back_Left);
        EncoderVelocity(&Back_Right);
```

```
void EncoderVelocity(WHEEL_Type* key)              //tworzę funkcje która oblicza aktualną prędkość
{
    if(key->motor_siganl<0)    //sprawdzam czy silnik obraca się w lewą stronę czy prawą, od tego zależy znak predkosci
    {
        key->encoder_data.Value=key->encoder_data.Value*-1;
    }
    key->encoder_data.TotalTurns+=key->encoder_data.Value;        //uaktualniam liczbę wszytkich obrotów które zrobił silnik
    float alpha;
    if(key->encoder_data.Value!=0)
    {
        alpha = 0.01*(abs(key->encoder_data.Value));     //to zrobienie dynamicznego filtra - poprzez dynamiczną zmianę współczynnika wygładzającego
        key->encoder_data.CurrentVelocity=key->encoder_data.Value*alpha+(1-alpha)*key->encoder_data.V1;//Obliczam prędkość z filtra ekspotencjalnego
        key->encoder_data.V1=key->encoder_data.CurrentVelocity;                    //cofam próbki o jeden
    }
    else
    {
        key->encoder_data.V1=0;
        key->encoder_data.CurrentVelocity=0;
    }
    key->encoder_data.Value=0;                                  //zeruję wartość imulsów
}
```

Do obliczenia prędkość wykorzystano dynamiczny filtr ekspotencjalny, aby zniwelować błąd próbkowania impulsów – szczególnie podczas niskich prędkości obrotowych.

- Drugim krokiem jest obliczenie sygnału dla silnika

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)//to callback od przerwania co określony czas
{
    if(htim->Instance==TIM11)//sprawdzam czy przerwanie pochodzi z timera który mnie interesuje - przerwanie co 0.2s
    {
        //HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin); //to część kodu testujacego
        EncoderVelocity(&Front_Left);
        EncoderVelocity(&Front_Right);
        EncoderVelocity(&Back_Left);
        EncoderVelocity(&Back_Right);

        SpeedRamp(&Front_Left);
        SpeedRamp(&Front_Right);
        SpeedRamp(&Back_Left);
        SpeedRamp(&Back_Right);

        PID(&Front_Left);
        PID(&Front_Right);
        PID(&Back_Left);
        PID(&Back_Right);

    }
```

```
void SpeedRamp(WHEEL_Type *key)
{
    if(key->ramp_data.LastFinalSpeed!=key->ramp_data.FinalSpeed)
    {
        key->ramp_data.LastFinalSpeed=key->ramp_data.FinalSpeed;     //przypisuje starej predkosci nową prędkość
        key->ramp_data.StartSpeed=key->encoder_data.CurrentVelocity; //predkosc startująca to aktualna predkość
    }
    if(key->ramp_data.LastFinalSpeed==key->ramp_data.FinalSpeed)
    {
        key->ramp_data.Delta=(key->ramp_data.FinalSpeed-key->ramp_data.StartSpeed)/(key->ramp_data.RisingTime*5);
        //Obliczam o ile mam zwiększać predkosc z kolejnym krokiem + plus kontrola przed złym znakiem
        if(key->ramp_data.Delta<0)//dla rampy mniejszej od zera czyli zmniejszanie predkosci są inne warunki w if poniżej
        {
            if((key->set_velocity<(key->ramp_data.FinalSpeed-key->ramp_data.Delta))&&(key->set_velocity>(key->ramp_data.FinalSpeed+key->ramp_data.Delta)))
            {
                key->set_velocity=key->ramp_data.FinalSpeed;
            }
            else
            {
                key->set_velocity+=key->ramp_data.Delta;
            }
        }
        else if(key->ramp_data.Delta>0)
        {
            if((key->set_velocity>(key->ramp_data.FinalSpeed-key->ramp_data.Delta))&&(key->set_velocity<(key->ramp_data.FinalSpeed+key->ramp_data.Delta)))
            {
                key->set_velocity=key->ramp_data.FinalSpeed;
            }
            else
            {
                key->set_velocity+=key->ramp_data.Delta;
            }
        }
    }
}
```

```
void PID(WHEEL_Type* key)
{
    float P=0.267;     //definiuje współczynnik P,I,D
    float I=14.1;
    float D=0.0;
    key->pid_data.Deviation=key->set_velocity-key->encoder_data.CurrentVelocity;    //obliczam uchyb regulacji
    if(key->pid_data.PidSignal>-1000&&key->pid_data.PidSignal<1000)     //to zabezpieczenie przed windup
    {
        key->pid_data.Integral+=key->pid_data.Deviation*0.2;//obliczam całkę z uchybu 0.2 ponieważ bede tą funkcję wykonywał co 0.2s
    }
    else
    {
        if(key->encoder_data.CurrentVelocity==0)
        {
            key->pid_data.Integral=0;
        }
        if(key->encoder_data.CurrentVelocity!=0)
        {
            key->pid_data.Integral=key->pid_data.Integral;
        }
    }

    key->pid_data.PidSignal = key->pid_data.Deviation*P + key->pid_data.Integral*I + key->pid_data.Derivative*D;//to standardowy regulator PID
    key->motor_siganl=key->pid_data.PidSignal;
}
```

Funkcja SpeedRamp odpowiada stopniowe zwiększanie/zmniejszanie prędkości, która ma być zdana na regulator PI.

● Czwartym krokiem jest obliczenie prędkość jaką ma mieć koło w zależności od sposobu lokomocji w jakim autko ma się poruszać.

```
  while (1)
  {

      //wywołuję funkcję która ustawia predkość i kierunek jazdy dla czterech kół
      MotorState(&Front_Left);
      MotorState(&Front_Right);
      MotorState(&Back_Left);
      MotorState(&Back_Right);

      VelocityCalculation(&Front_Left, &Front_Right, &Back_Left, &Back_Right,speed);


    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
```

```c
void MotorState(WHEEL_Type *key)
{
    switch(key->MOTOR_STATE)                    //a teraz sprawdzam w którym dokładnie trybie ruchu jestem
    {
        case IDLE_1: //w tym stanie predkosc wynosi 0
            __HAL_TIM_SET_COMPARE(key->timer_data._htim_PWM,key->timer_data._channel_PWM,0);    //ustawiam sygnał PWM 0%
            break; //zapobiega wykonywaniu kolejnych przypadków.
        case DRIVE: //w tym stanie nie hamujemy i mamy zezwolnie na prace i ustawaim zadaną prędkość
            if(key->motor_siganl<0)             //gdy predkosc jest mniejsza od zera to obracam silnk w lewo a gdy wieksza to w prawo
            {
                HAL_GPIO_WritePin(key->pinout_data.DirectionPort,key->pinout_data.DirectionPin, RESET);      //gdy set to obroty w lewo
                __HAL_TIM_SET_COMPARE(key->timer_data._htim_PWM,key->timer_data._channel_PWM,abs(key->motor_siganl));
            }                                       //ustawiam żądany sygnał na odpowiednim timerze i kanale ale sygnał bez minusa
            else if(key->motor_siganl>=0)
            {
                HAL_GPIO_WritePin(key->pinout_data.DirectionPort,key->pinout_data.DirectionPin, SET);      //gdy reset to obroty w prawo
                __HAL_TIM_SET_COMPARE(key->timer_data._htim_PWM,key->timer_data._channel_PWM,key->motor_siganl);
            }                                                                   //ustawiam żądany sygnał na odpowiednim timerze i kanale
            break;
        case BRAKE: //w tym stanie predkosc wynosi 0
            __HAL_TIM_SET_COMPARE(key->timer_data._htim_PWM,key->timer_data._channel_PWM,0);        //ustawiam sygnał PWM 0%
            break;
    }
}
```

```c
void VelocityCalculation(WHEEL_Type* FL,WHEEL_Type* FR,WHEEL_Type* BL,WHEEL_Type* BR,int speed)
{                                           //ta funkcja oblicza jaką predkosc ma mieć kazde kolo
    if((FR->MOTOR_STATE==DRIVE)&&(FL->MOTOR_STATE==DRIVE)&&(BR->MOTOR_STATE==DRIVE)&&(BL->MOTOR_STATE==DRIVE))
    {                                                       //sprawdzam czy kazde kolo jest w stanie drive
        if((FR->MOVE_STATE==FL->MOVE_STATE)&&(BR->MOVE_STATE==BL->MOVE_STATE)&&(FL->MOVE_STATE==BR->MOVE_STATE))
        {                                                   //sprawdzam czy kazde kolo jest w tym samym trybie ruchu
            switch(FR->MOVE_STATE)                      //a teraz sprawdzam w którym dokładnie trybie ruchu jestem
            {
            case IDLE_2:
                FL->ramp_data.FinalSpeed=0;      //w trybie nieustalonym predkosc wynosi 0
                FR->ramp_data.FinalSpeed=0;
                BL->ramp_data.FinalSpeed=0;
                BR->ramp_data.FinalSpeed=0;

                FL->pid_data.Integral=0;
                FR->pid_data.Integral=0;
                BL->pid_data.Integral=0;
                BR->pid_data.Integral=0;

                FL->pid_data.PidSignal=0;
                FR->pid_data.PidSignal=0;
                BL->pid_data.PidSignal=0;
                BR->pid_data.PidSignal=0;
                break;
            case FORWARD:
                FL->ramp_data.FinalSpeed=-speed;//w trybie do przodu - wszystkie koła krecą się w tą samą stronę
                FR->ramp_data.FinalSpeed=speed;
                BL->ramp_data.FinalSpeed=-speed;
                BR->ramp_data.FinalSpeed=speed;
                break;
            case RIGHT_SIDE:
                FL->ramp_data.FinalSpeed=-speed;
                FR->ramp_data.FinalSpeed=-speed;
                BL->ramp_data.FinalSpeed=speed;
                BR->ramp_data.FinalSpeed=speed;
                break;
           case AROUND:
               FL->ramp_data.FinalSpeed=-speed;
               FR->ramp_data.FinalSpeed=-speed;
               BL->ramp_data.FinalSpeed=-speed;
               BR->ramp_data.FinalSpeed=-speed;
               break;
            }

        }
    }
}
```

- Kolejnym krokiem jest odebranie informacji, która przyszła po UART-cie

```c
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    if(huart->Instance == USART2)
        {
          HAL_UART_Receive_IT(&huart2, &znak,1);
              speed=ReadInfoFromUart(znak, &Front_Left, &Front_Right, &Back_Left, &Back_Right);
        }

}
```

Odczyt informacji z czujnika INA3221 po interfejsie I2C

- Pierwszym krokiem jest stworzenie struktury, w której przechowuje informację

```
70  INA3221_Type Sensor_INA3221; //tworze obiekt mojego czujnika do pomiaru pradu i napiecia

87  typedef struct      //tworze strukture w której mam informacje z danego kanału
88  {
89      float           Voltage;
90      float           Current;
91  }CHANNEL_Type;
92
93  typedef struct     //tworze strukture w której mam wszystkie dane z czujnika INA3221 - trzy kanałowego
94  {
95      CHANNEL_Type    Channel1;  //to kanał od obw. ster  - 5V
96      CHANNEL_Type    Channel2;  //to kanał od obw. mocy  - 12V
97      CHANNEL_Type    Channel3;  //to kanał od baterii    - 16V
98      float           Power;     //to moc jaką pobieramy z baterii - jednostki to WATY
99      float           TotalEnergy; //to całkowita zużyta energia - jednostki to Wh
00  }INA3221_Type;
```

- Kolejnym krokiem jest odczyt danych

```
if(htim->Instance==TIM10)//sprawdzam czy przerwanie pochodzi z timera który mnie interesuje - przerwanie co 0.5s
{
    //HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin); //to część kodu testujacego
    //odczytuję informację z czujnika i przypisuję do odpowiedniego miejsca w strukturze
    Sensor_INA3221.Channel1.Current=INA3221_GetCurrentRaw(&ina3221, INA3221_CHANNEL_1);
    Sensor_INA3221.Channel1.Voltage=INA3221_GetBusVoltage(&ina3221, INA3221_CHANNEL_1);
    Sensor_INA3221.Channel2.Current=INA3221_GetCurrentRaw(&ina3221, INA3221_CHANNEL_2);
    Sensor_INA3221.Channel2.Voltage=INA3221_GetBusVoltage(&ina3221, INA3221_CHANNEL_2);
    Sensor_INA3221.Channel3.Current=INA3221_GetCurrentRaw(&ina3221, INA3221_CHANNEL_3);
    Sensor_INA3221.Channel3.Voltage=INA3221_GetBusVoltage(&ina3221, INA3221_CHANNEL_3);

    Sensor_INA3221.Power=Sensor_INA3221.Channel3.Voltage*Sensor_INA3221.Channel3.Current;//obliczam teraz moc w watach
    Sensor_INA3221.TotalEnergy+=Sensor_INA3221.Power/7200;                    //teraz obliczam enegie zużytą w Wh
```

Informacje są odczytywane w Callbacku co 0.5s i zapisywane do odpowiednich zmiennych w strukturze

Wysłanie wszystkich zebranych informacji do jednostki Master po interfejsie UART

```
297     if(htim->Instance==TIM10)//sprawdzam czy przerwanie pochodzi z timera który mnie interesuje - przerwanie co 0.5s
298     {
299         //HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin); //to część kodu testujacego
300         //odczytuję informację z czujnika i przypisuję do odpowiedniego miejsca w strukturze
301         Sensor_INA3221.Channel1.Current=INA3221_GetCurrentRaw(&ina3221, INA3221_CHANNEL_1);
302         Sensor_INA3221.Channel1.Voltage=INA3221_GetBusVoltage(&ina3221, INA3221_CHANNEL_1);
303         Sensor_INA3221.Channel2.Current=INA3221_GetCurrentRaw(&ina3221, INA3221_CHANNEL_2);
304         Sensor_INA3221.Channel2.Voltage=INA3221_GetBusVoltage(&ina3221, INA3221_CHANNEL_2);
305         Sensor_INA3221.Channel3.Current=INA3221_GetCurrentRaw(&ina3221, INA3221_CHANNEL_3);
306         Sensor_INA3221.Channel3.Voltage=INA3221_GetBusVoltage(&ina3221, INA3221_CHANNEL_3);
307
308         Sensor_INA3221.Power=Sensor_INA3221.Channel3.Voltage*Sensor_INA3221.Channel3.Current;//obliczam teraz moc w watach
309         Sensor_INA3221.TotalEnergy+=Sensor_INA3221.Power/7200;                    //teraz obliczam enegie zużytą w Wh
310         //teraz wysyłanie informacj
311         char Message[100];
312         //to obroty kół na minutę
313         float Velo=(abs(Front_Left.encoder_data.CurrentVelocity)+abs(Front_Right.encoder_data.CurrentVelocity)+
314                 abs(Back_Left.encoder_data.CurrentVelocity)+abs(Back_Right.encoder_data.CurrentVelocity))*15/54;
315         int Lenght = sprintf((char*)Message,"V=%.2f,C1=%.2f,N1=%.2f,C2=%.2f,N2=%.2f,C3=%.2f,N3=%.2f,P=%.2f,E=%.2f\n\r",Velo
316             ,Sensor_INA3221.Channel1.Current,Sensor_INA3221.Channel1.Voltage,Sensor_INA3221.Channel2.Current,
317             Sensor_INA3221.Channel2.Voltage,Sensor_INA3221.Channel3.Current,Sensor_INA3221.Channel3.Voltage,
318             Sensor_INA3221.Power,Sensor_INA3221.TotalEnergy);
319         HAL_UART_Transmit_IT(&huart2, Message, Lenght);
```

Informacje są wysyłane również w Callback-u co 0.5s, dodatkowo obliczana jest prędkość w obrotach na minutę

Opis kodu do łączenia po module bluetooth z aplikacją na telefonie

hc05.py > ...

```python
import serial
import threading
import time
```

Importujemy biblioteki wysyłania znaków, czasu oraz wątkowania.

```python

# Globalne zmienne
ser = None
running = True
lock = threading.Lock()

# Słownik przechowujący dane z UART
phone_box = {
    "velocity": 0,
    "direction": 0,
    "poz_x": 0,
    "poz_y": 0,
    "around": "neutral"
}
```

Nadajemy stany początkowe zmiennym globalnym oraz definiujemy ciąg danych z UART.

```python
def initialize_uart_hc():
    """
    Inicjalizacja połączenia UART.
    """
    global ser
    try:
        ser = serial.Serial('/dev/serial0', 9600, timeout=1) # /dev/serial0 , /dev/ttyS0
        ser.flush()
        print("UART HC został zainicjalizowany.")
        return True
    except serial.SerialException as e:
        print(f"Błąd inicjalizacji UART HC: {e}")
        return False

```

Inicjalizyjemy połączenie UART i definiujemy by nam wyświetliło czy inicjalizacja się powiodła.

```python
def parse_data(data):
    """
    Parsowanie danych odebranych z UART.
    Oczekiwany format: 'Velocity= 0, Direction= 0, PozX= 140, PozY= 140, Around= 1'
    Wartości pola "around":
    - 0 -> neutral
    - 1 -> left
    - 2 -> right
    """
    global phone_box
    try:
        print(f"Raw data for parsing: {data}")  # Log danych wejściowych

        parts = data.split(',')
        normalized_keys = {
            "velocity": "velocity",
            "direction": "direction",
            "pozx": "poz_x",
            "pozy": "poz_y",
            "around": "around"
        }
```

```python
        # Mapa konwersji dla pola "around"
        around_map = {
            "0": "neutral",
            "1": "left",
            "2": "right"
        }

        temp_data = {}
        for part in parts:
            try:
                # Rozdzielenie klucza i wartości
                if '=' in part:
                    key, value = part.split('=')
                    key = key.strip().lower()
                    value = value.strip()

                    # Obsługa pola "around"
                    if key == "around":
                        # Zamiana wartości numerycznej na tekstową (np. 1 -> left)
                        value = around_map.get(value, "unknown")
                    else:
                        value = int(value)  # Konwersja wartości liczbowych

                    # Normalizacja kluczy
                    normalized_key = normalized_keys.get(key, key)
                    temp_data[normalized_key] = value
                else:
                    print(f"Nieprawidłowy fragment danych: {part}")

            except ValueError as ve:
                print(f"Nieprawidłowy fragment danych: {part} -> {ve}")

        with lock:
            phone_box.update(temp_data)

        print(f"Parsed phone_box: {phone_box}")
    except Exception as e:
        print(f"Błąd podczas parsowania danych: {e}")
```

W tej części następuje chronologiczny zapis danych wejściowych, analiza i konwersja danych na oczekiwany format oraz aktualizacja ich. Jeśli dane są nieprawidłowe zwraca nam informacje o tym.

```python
def uart_hc_processing():
    """
    Funkcja wątku do odbioru i przetwarzania danych z UART.
    """
    global running
    try:
        while running:
            if ser and ser.in_waiting > 0:
                received_data = ser.readline().decode('utf-8', errors='ignore').strip()
                #print(f"Odebrano z telefonu: {received_data}")

                if "velocity" in received_data.lower() and "direction" in received_data.lower():
                    parse_data(received_data)
                    print(f"Odebrano z telefonu 2: {received_data}")
            else:
                print("Brak danych na porcie UART.")  # Log gdy brak danych

            time.sleep(0.15)  # Ograniczenie obciążenia procesora
    except serial.SerialException as e:
        print(f"Błąd portu szeregowego: {e}")
    except OSError as e:
        print(f"Błąd wejścia/wyjścia: {e}")
    except KeyboardInterrupt:
        print("Program zatrzymany.")
    finally:
        close_uart_hc()
```

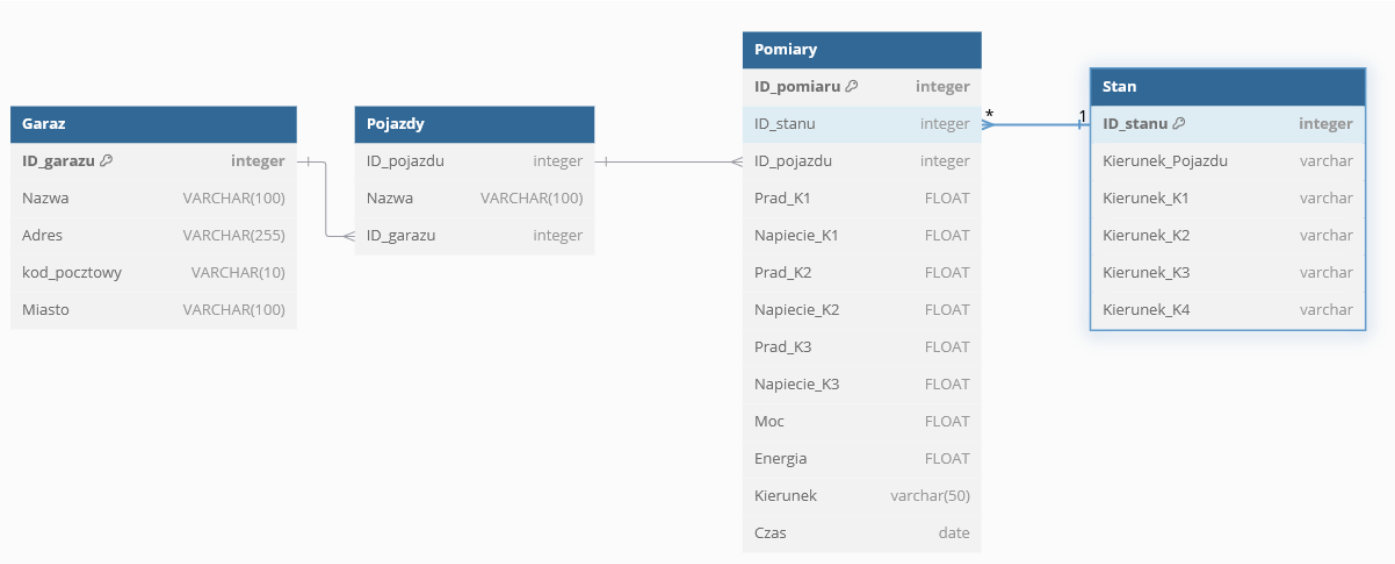
Tworzymy wątek do przetwarzania danych odebranych z UART. Następuje tu wypisanie odebranych danych oraz wprowadzenie ograniczeń procesora.

```python
def get_phone_box():
    """
    Zwraca aktualne dane z UART.
    """
    global phone_box
    print(f"phone_box w get_phone_box: {phone_box}")  # Diagnostyka
    with lock:
        return phone_box.copy()

def close_uart_hc():
    """
    Zamykanie portu UART.
    """
    global ser
    if ser and ser.is_open:
        ser.close()
        print("Port UART HC został zamknięty.")
```

Tu następuje zwrot aktualnych danych w celu diagnostyki oraz zamknięcie portu UART.

Baza Danych

Diagram ERD:



-Baza danych została stworzona w programie mysql pobranej na Rasberry Pi 4B. Po pobraniu utworzono konto root do zarządzania bazą danych.

-Po połączeniu mysql z phpmyadmin należało zalogować się danymi (z pliku api.php). Nastepnie stworzono nową bazę danych *Projekt_db.* W zakładce SQL został wpisany kod powyższy *kod*. Od tego momentu można było monitorować pliki zapisywane w bazie danych. Tworzenie wykresów z bazy danych uzyskano dzięki programowi **Jupiter -** *edytor Jupyter Notebook jest używany do interaktywnego, eksploracyjnego programowania analizy danych i wizualizacji danych.* Do realizacji anlizy i wizualizacji wykresów z otrzymanych danych należało zacząć kod łączący się z bazą danych. Następnie napisanie funkcji pobierającej wybrane dane. Na końcu kod przetwarzający otrzymane wyniki w wykres.

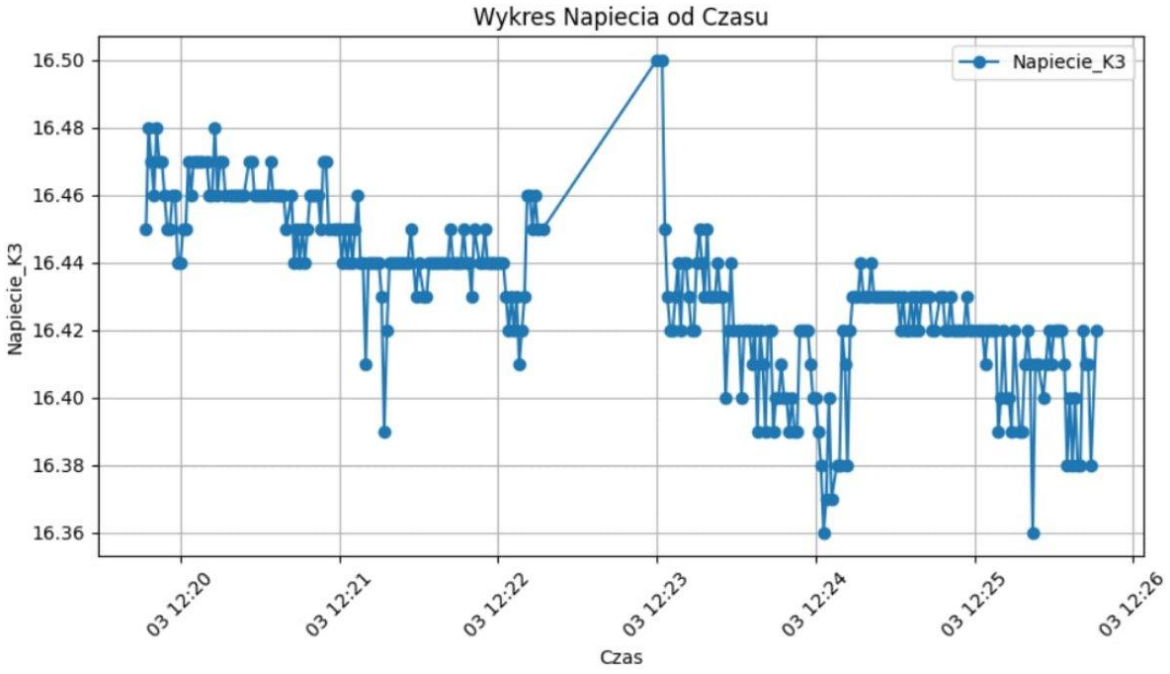
wykers_jupyter.ipynb > import mysql.connector

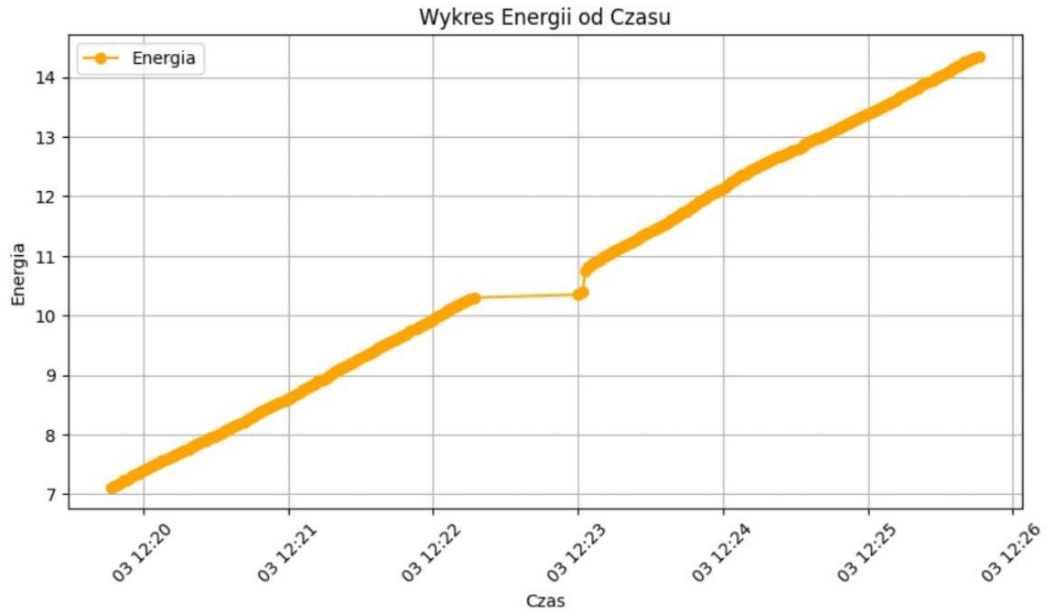
Generate | Code | Markdown | Run All | Restart | Clear All Outputs | View data | Jupyter Variables | Outline | ...

```
try:
    # Połączenie z bazą MySQL
    conn = mysql.connector.connect(
        host="localhost",
        user="root",
        password="123",
        database="Projekt_db"
    )

    cursor = conn.cursor()

    try:
        # Pobieranie danych
        cursor.execute("SELECT Czas, Napiecie_K2, FROM Pomiary ORDER BY Czas ASC")
        rows = cursor.fetchall()

        if not rows:
            raise ValueError("Zapytanie nie zwróciło żadnych wyników.")

        # Tworzenie DataFrame
        df = pd.DataFrame(rows, columns=['Czas', 'Napiecie_K2'])

        try:
            df['Data'] = pd.to_datetime(df['Czas'])
        except Exception as e:
            raise ValueError(f"Błąd konwersji kolumny 'Czas' na format datetime: {e}")

        # Rysowanie wykresu
```

[1]

Możemy wybrać intersujące nas dane, a następnie stworzyć wykres. Dzięki takiemu rozwiązaniu można dokonać obserwacji stanów zachodzących podczas działania pojazdu.
Sprawdzając np. napięcia można zaobserwować m.in napięcie baterii, a tym samym stan pojemności.




Wykres Energii od Czasu
Energia
14
13
12
11
10
9
8
7
Energia
03 12:20
03 12:21
03 12:22
03 12:23
03 12:24
03 12:25
03 12:26
Czas