AKADEMIA NAUK STOSOWANYCH w ELBLĄGU

Rok akademicki 2024/25

Systemy Wbudowane i Mikroprocesory

Sprawozdanie – Projekt semestralny

Szymon Tułodziecki 21312
Kacper Olenkiewicz 20470

1 Cel i założenia projektu

Celem projektu jest **zaprojektowanie**, **wykonanie** oraz **oprogramowanie modelu autonomicznego pojazdu** z wykorzystaniem mikrokontrolera **STM32**. Projekt ma charakter **praktyczny** i obejmuje zarówno część **sprzętową**, jak i **programową**, ze szczególnym uwzględnieniem **implementacji systemów wbudowanych** oraz **sterowania w czasie rzeczywistym**.

Założeniem projektu jest stworzenie pojazdu wyposażonego w **elektryczny napęd sterowany sygnałami PWM**, **system sensoryczny do detekcji przeszkód** oraz **moduł komunikacyjny UART** (np. z wykorzystaniem modułu Bluetooth). Zasilanie pojazdu ma być **autonomiczne**, bez korzystania z interfejsu USB, a cały system powinien pracować **niezależnie**, bez ingerencji użytkownika podczas działania.

Projekt zakłada również **systematyczny rozwój oprogramowania** z wykorzystaniem **repozytorium GitHub** oraz opracowanie kompletnej **dokumentacji**, obejmującej **schematy elektryczne**, **opisy komponentów** oraz **końcowy raport** zawierający szczegóły konstrukcyjne i **wyniki testów**.

2 Repozytoria GitHub

Główne repozytorium projektu:
`https://github.com/szymon-tulodziecki/stm32g071-rc-robot`

Repozytorium z plikami do wykonania podwozia:
`https://github.com/szymon-tulodziecki/CircularChassis2WD`

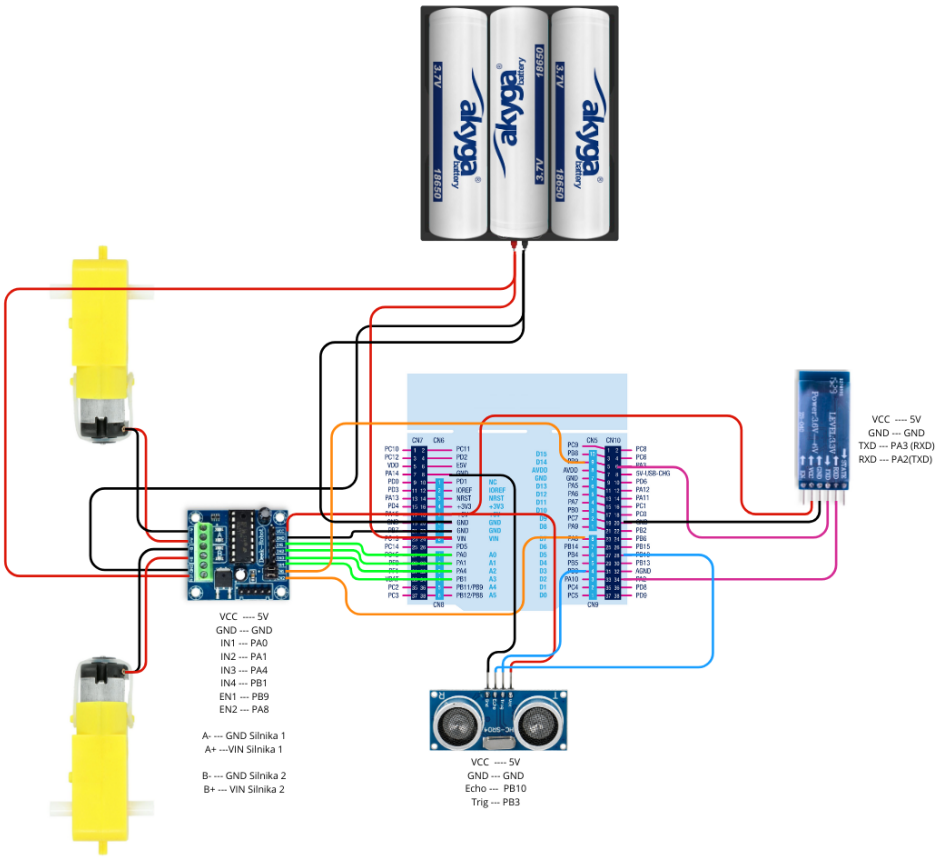
Repozytorium aplikacji sterującej:
`https://github.com/szymon-tulodziecki/rcar-control-app`

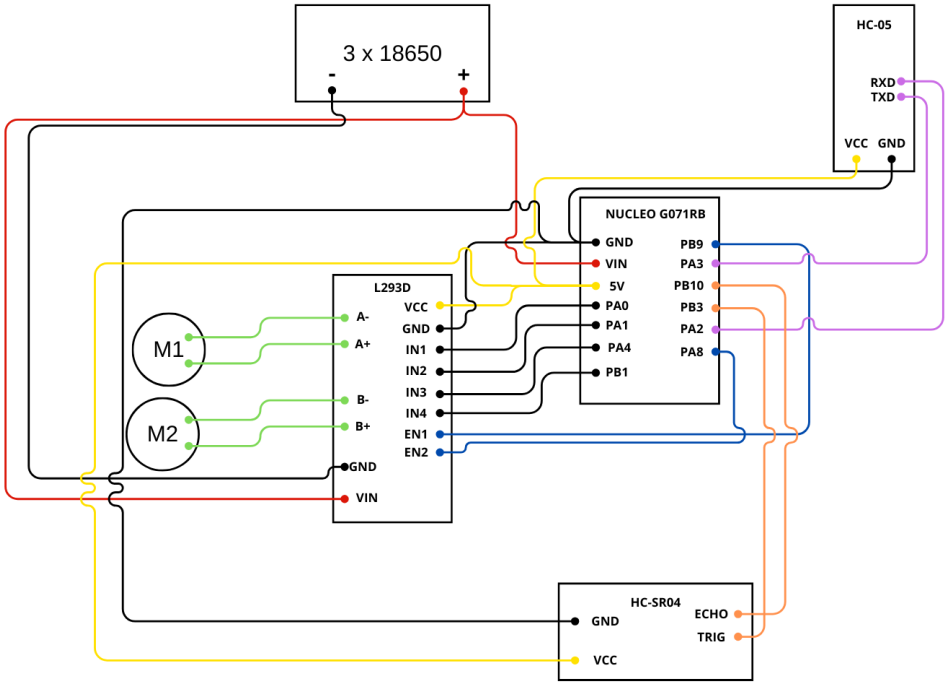
3 Podzespoły:

Do budowy robota wykorzystano zestaw podstawowych podzespołów elektronicznych i mechanicznych. Wszystkie elementy zostały dobrane tak, aby zapewnić prostą i funkcjonalną konstrukcję, którą można łatwo złożyć i uruchomić. Część komponentów pochodzi z gotowych modułów, a część – jak podwozie – została wykonana własnoręcznie.

- Podwozie własnego projektu
- Nucleo-G071RB – zestaw startowy z mikrokontrolerem STM32G071RB
- Moduł Bluetooth HC-05 – moduł komunikacji bezprzewodowej
- L293D – dwukanałowy sterownik silników
- HC-SR04 – ultradźwiękowy czujnik odległości
- Ogniwa 18650 (3 sztuki) – akumulatory zasilające
- Koszyk na ogniwa 18650 (szeregowy) – uchwyt na baterie
- Przewody M-F 17 cm – przewody połączeniowe
- Zestaw śrub i tulei M3 – elementy montażowe
- Płytka stykowa (170 pól) – prototypowanie połączeń

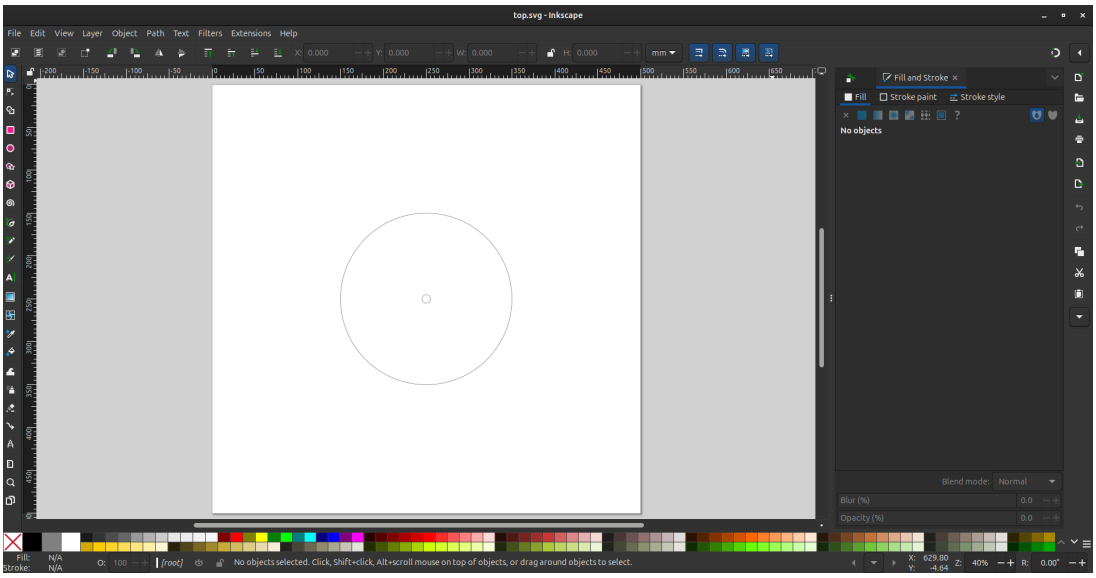
4 Schematy połączeń:



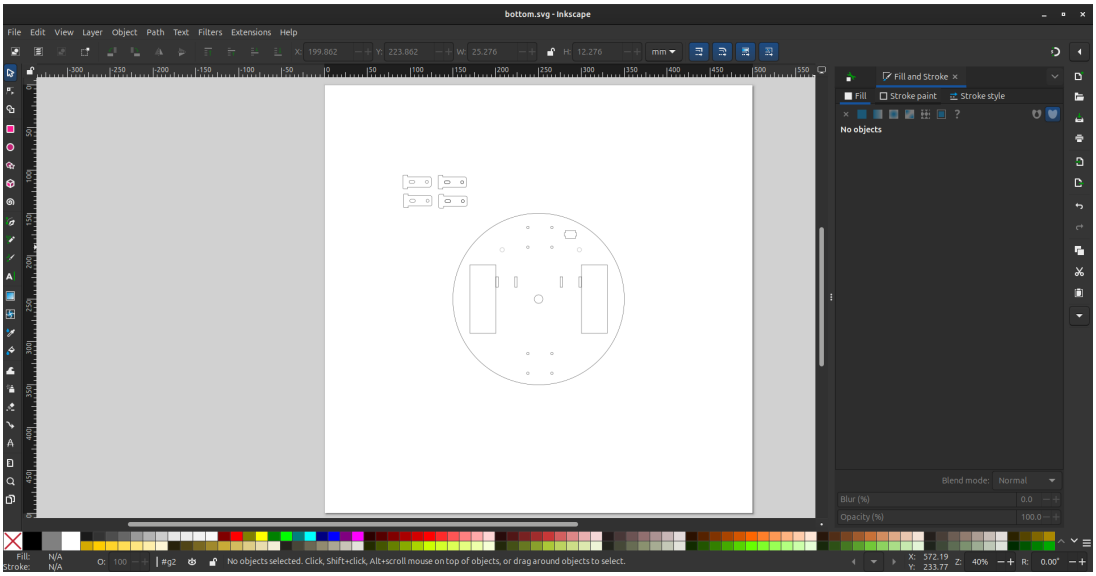


5 Projekt podwozia

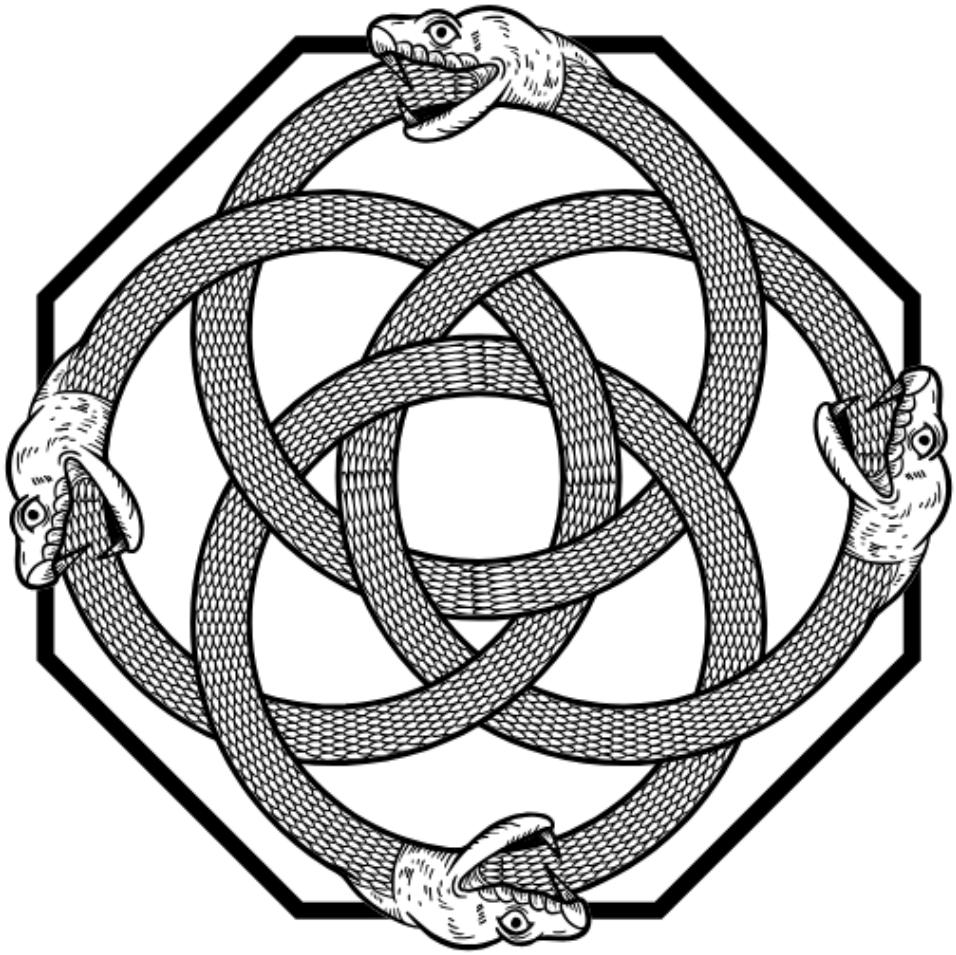
Podwozie robota zostało w pełni zaprojektowane i wykonane samodzielnie. Konstrukcja została dostosowana do wymagań elektroniki oraz planowanego sposobu poruszania się pojazdu. Na potrzeby projektu zaprojektowano zarówno dolną część podwozia, jak i górną osłonę (pokrywę), a także dodano elementy estetyczne w postaci graweru.



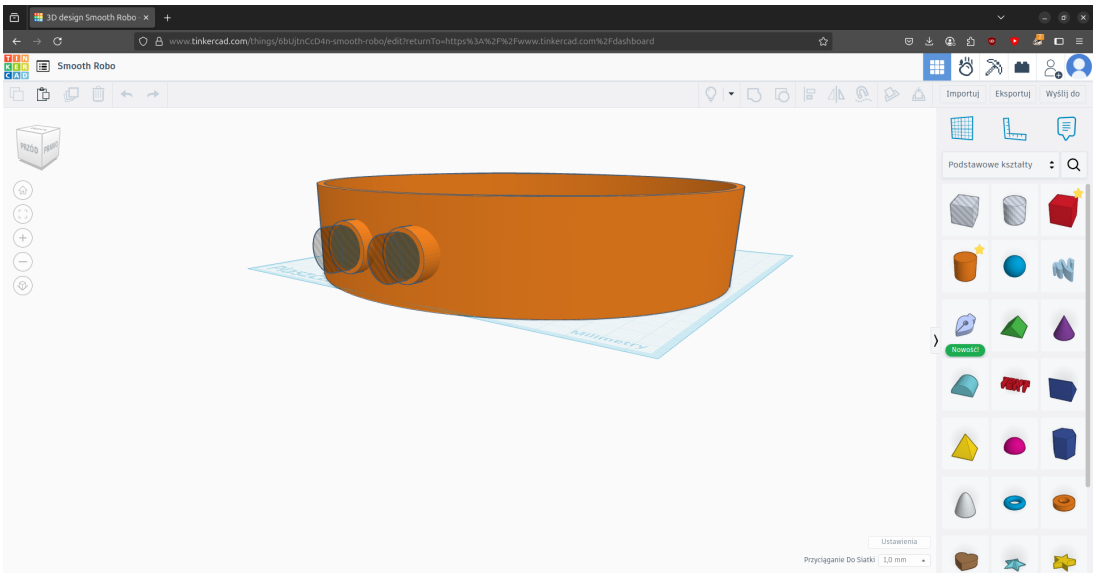
Zdjęcie 1: Model górnej pokrywy robota



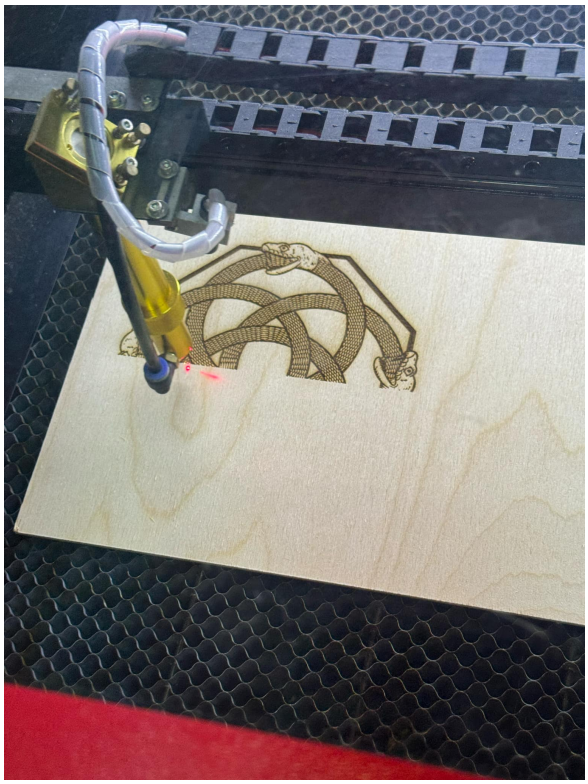
Zdjęcie 2: Model dolnej części podwozia robota wraz z uchwytami silników



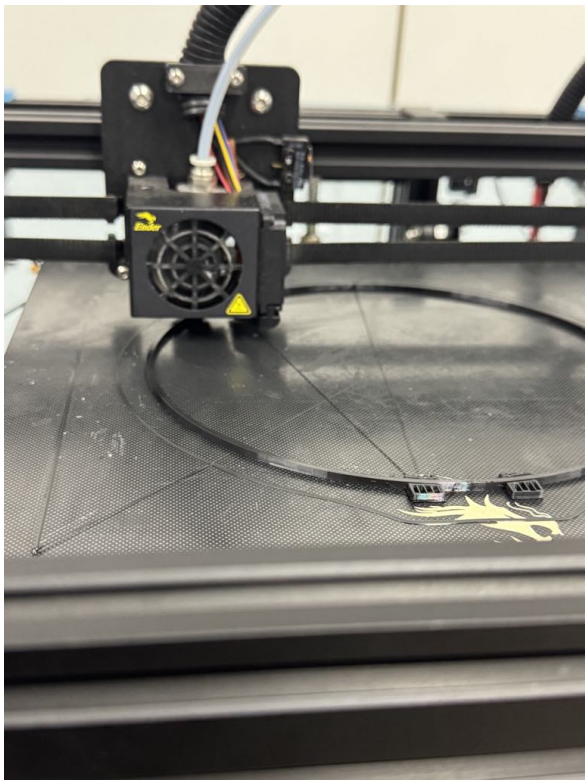
Zdjęcie 3: Grawer wykonany na powierzchni pokrywy robota



Zdjęcie 4: Projekt 3D bocznych ścianek podwozia z uchytem na czujnik HC-SR04



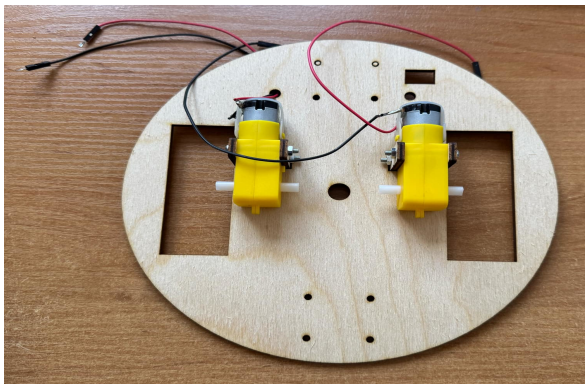
Zdjęcie 5: Wycinanie podwozia

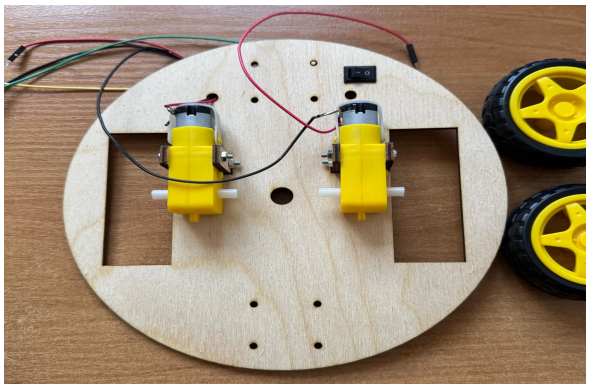


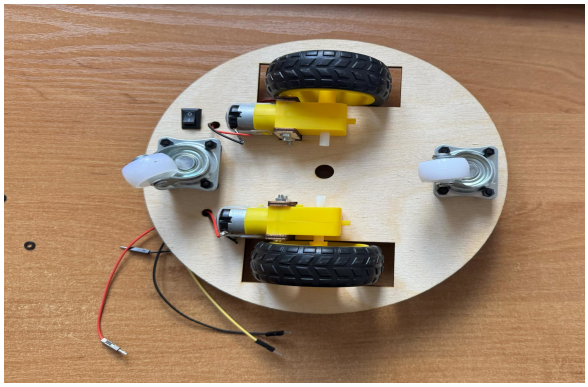
Zdjęcie 6: Drukowanie podwozia

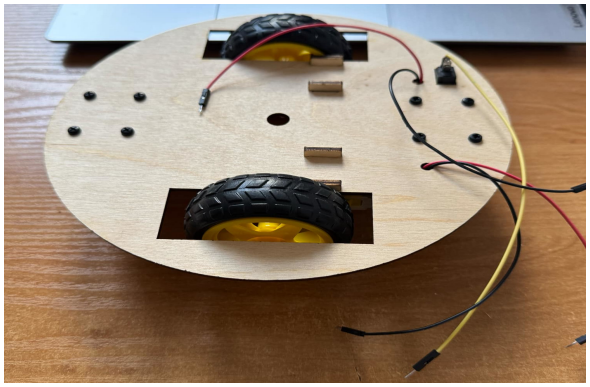
6 Montaż komponentów

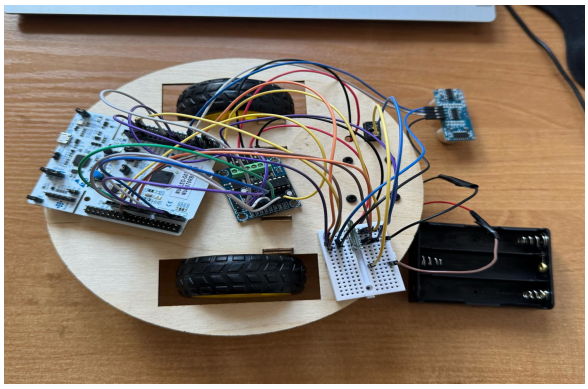
Wszystkie komponenty wymienione w sekcji 2 zostały połączone zgodnie ze schematami przedstawionymi w sekcji 3. Elementy zostały zamocowane na zaprojektowanym podwoziu w sposób zapewniający stabilność i łatwy dostęp do przewodów oraz złączy. Montaż przeprowadzono ręcznie, z wykorzystaniem podstawowych narzędzi i elementów mocujących.

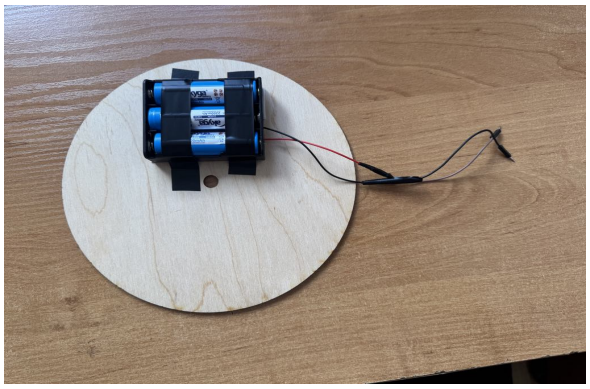


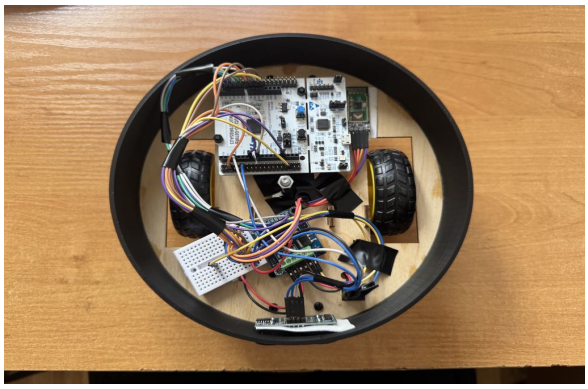


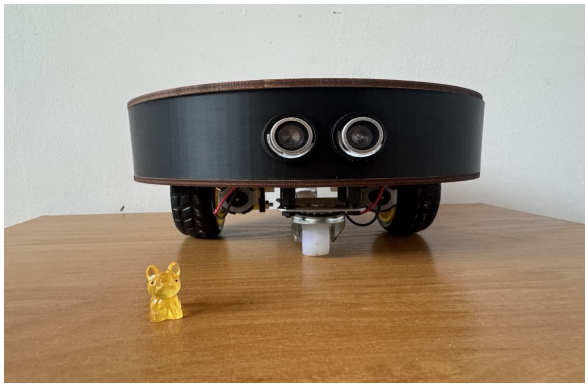


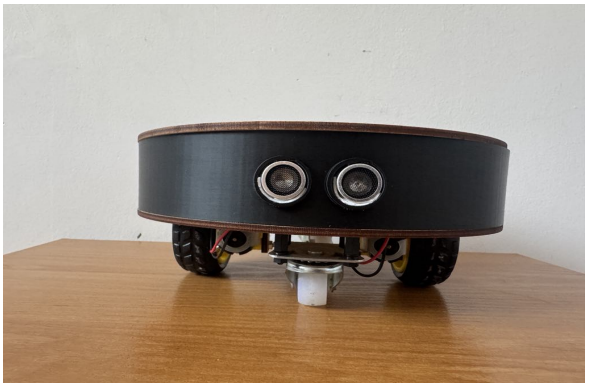






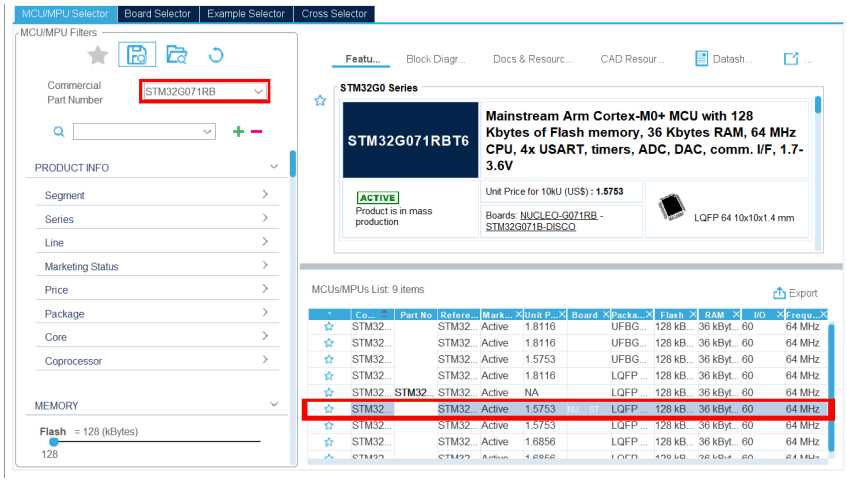




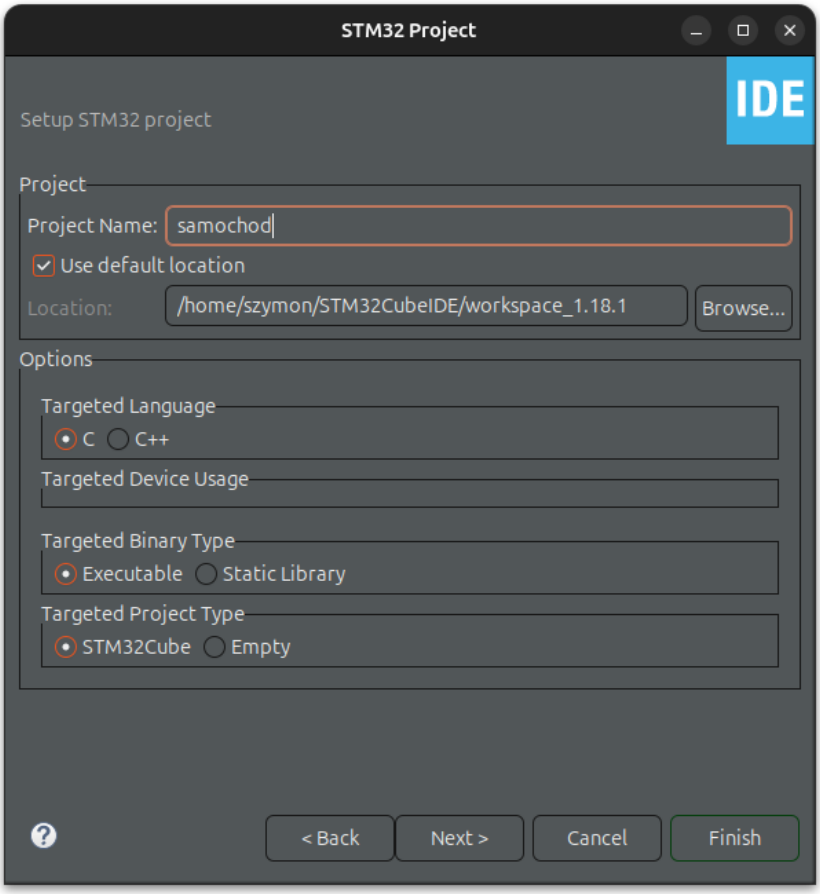


7 Konfiguracja projektu STM32CubeIDE

7.1 Wybór mikrokontrolera STM32G071RB

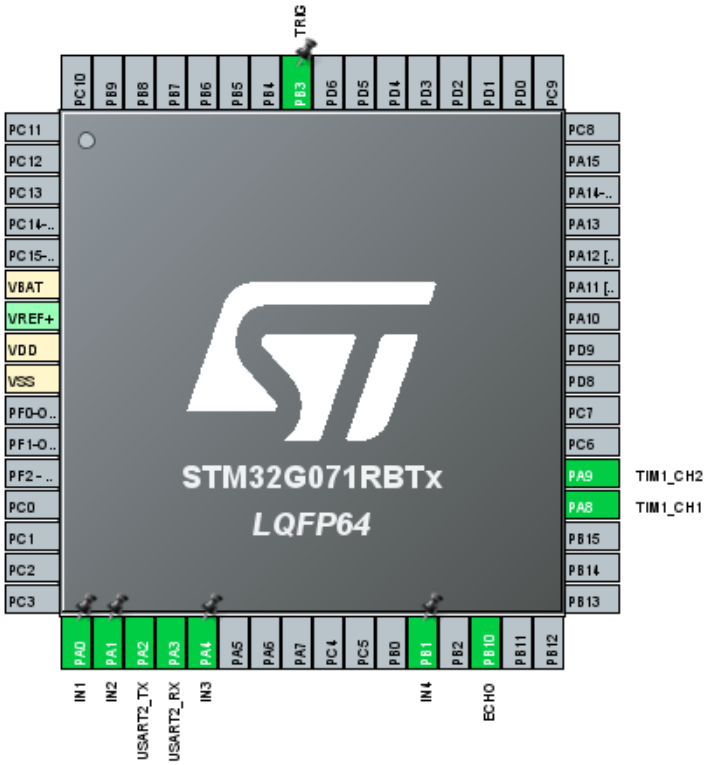


7.2 Tworzenie projektu

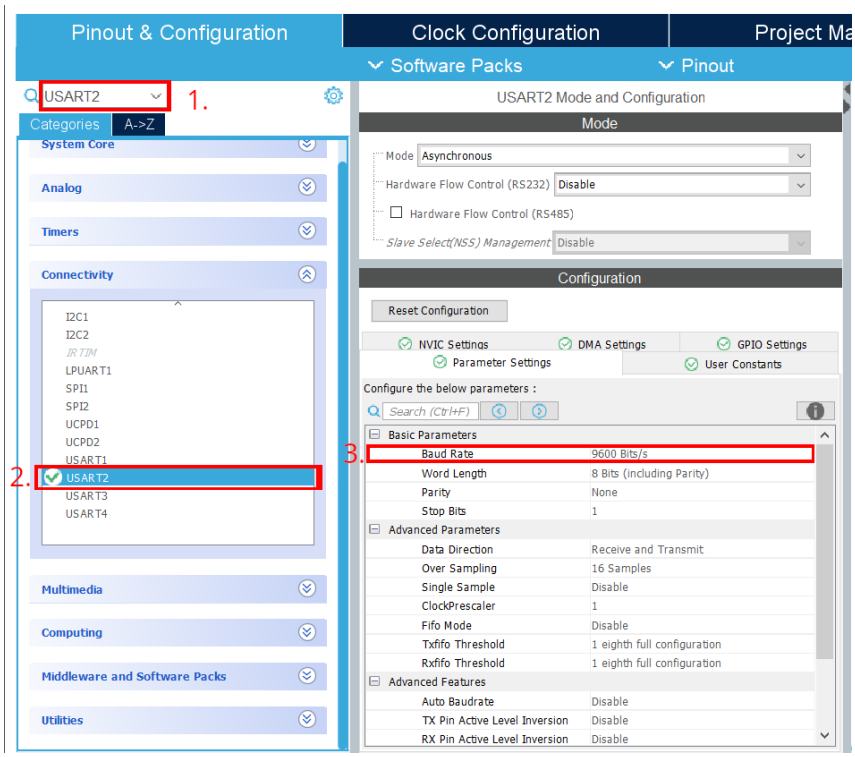


7.3 Schemat pinów STM32G071RB

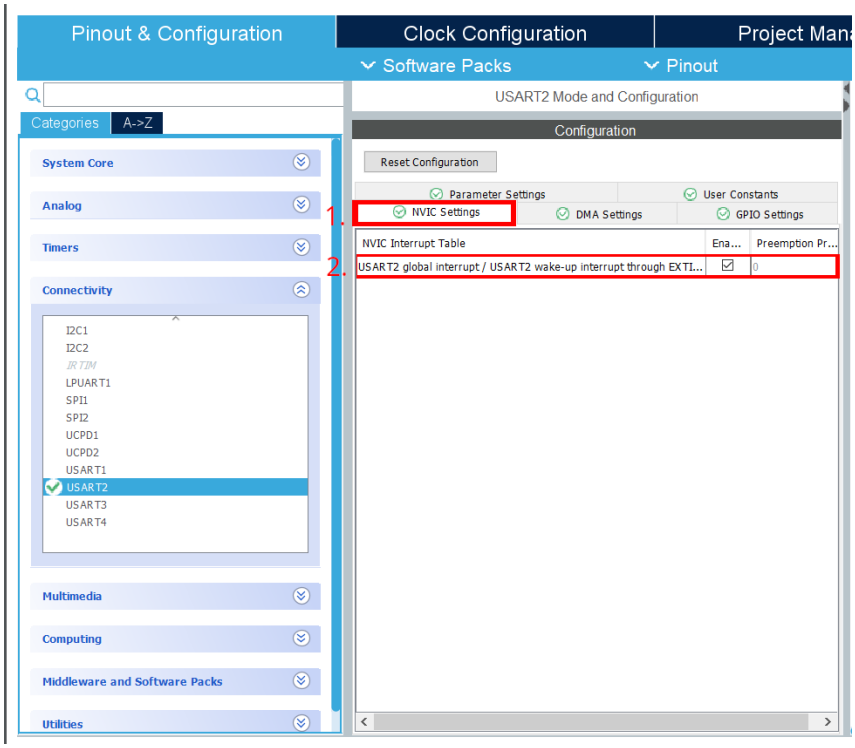
7.4 Konfiguracja pinów jako `GPIO_Output`



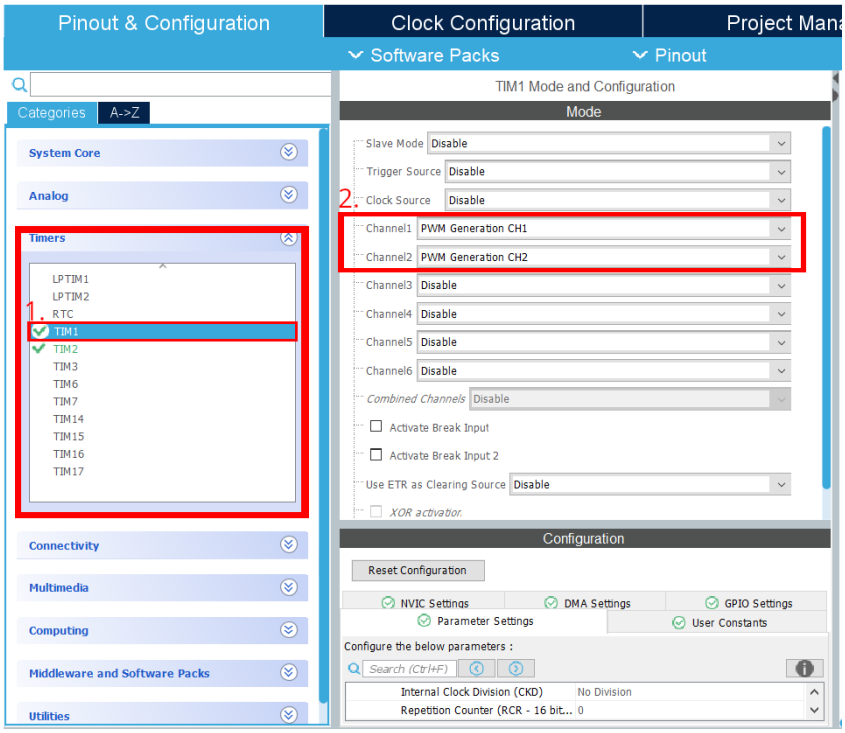
7.5 Konfiguracja USART2



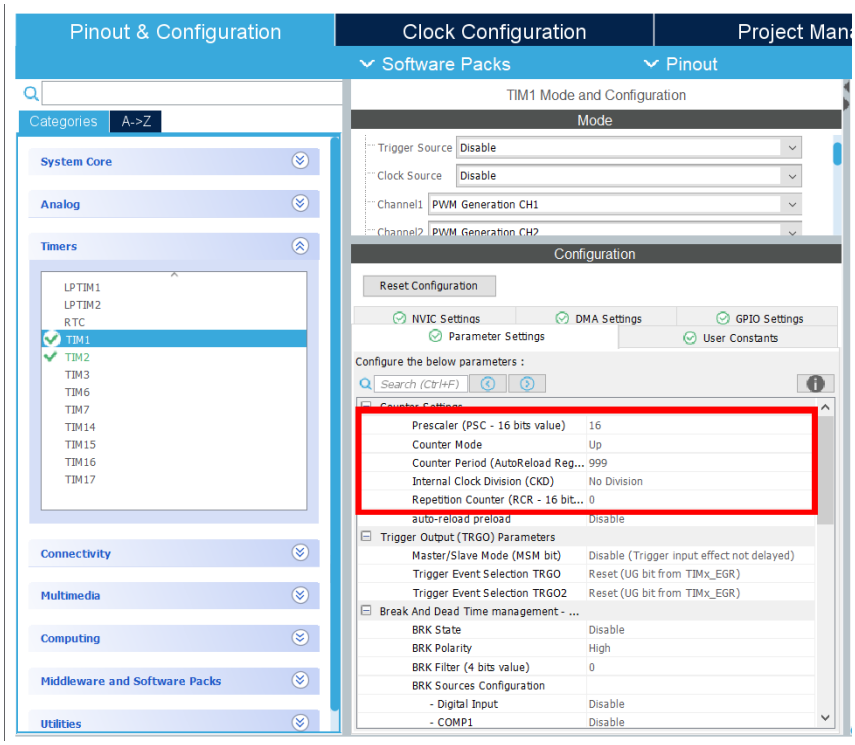
7.6 Ustawienia NVIC



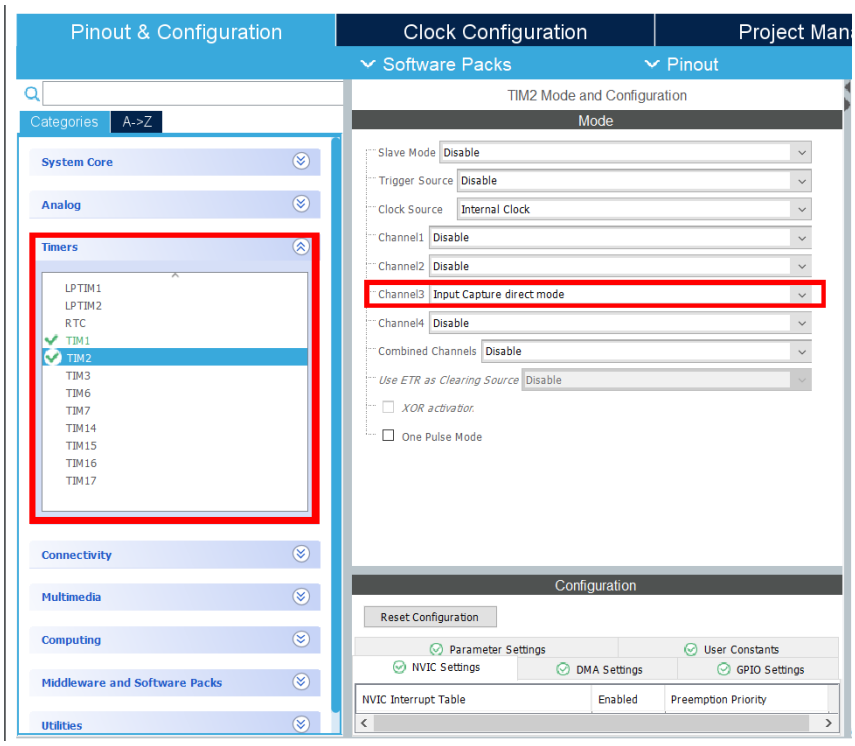
7.7 Ustawienia PWM dla Timer 1



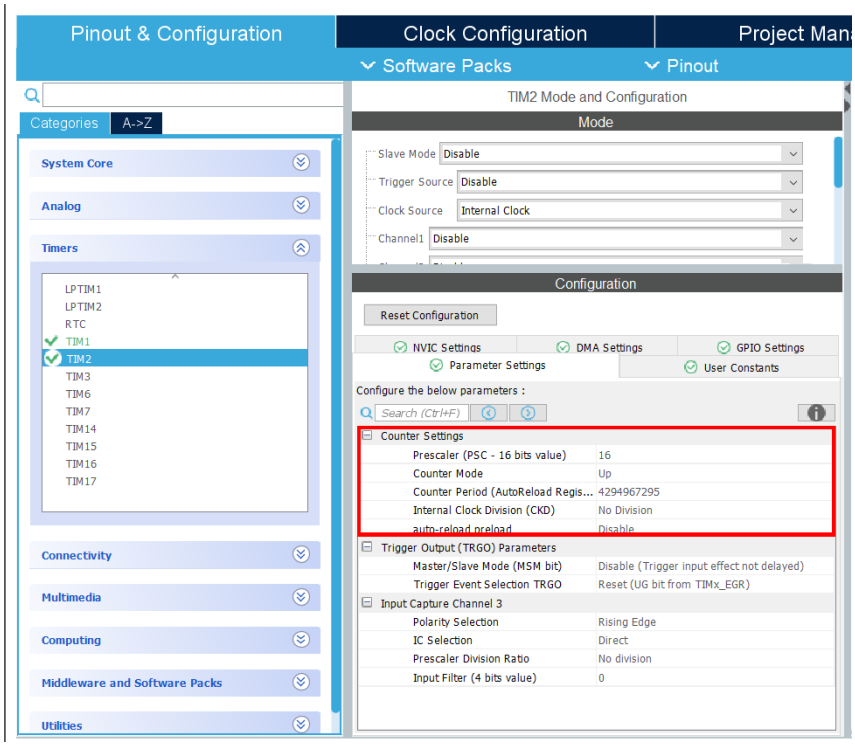
7.8 Prescaler i Counter Period dla Timer 1



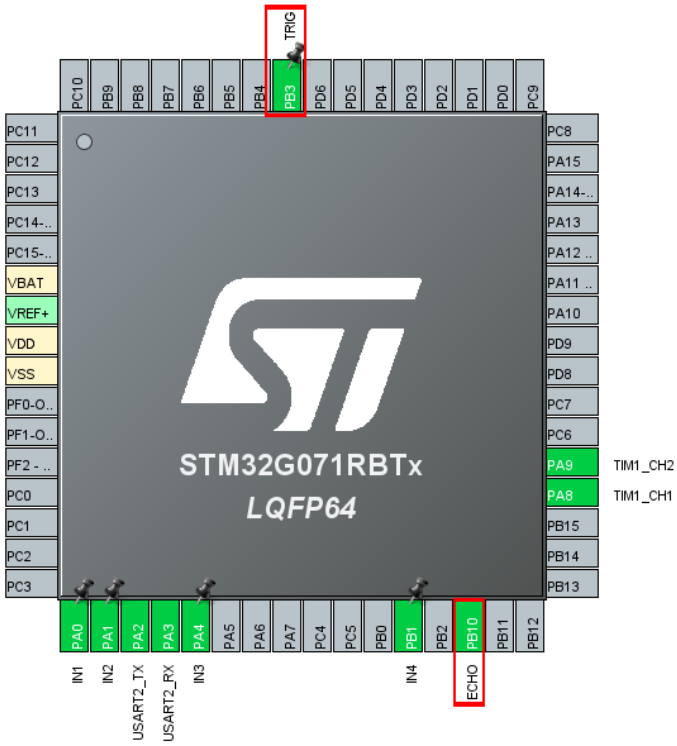
7.9 Konfiguracja Timer 2 w trybie Input Capture



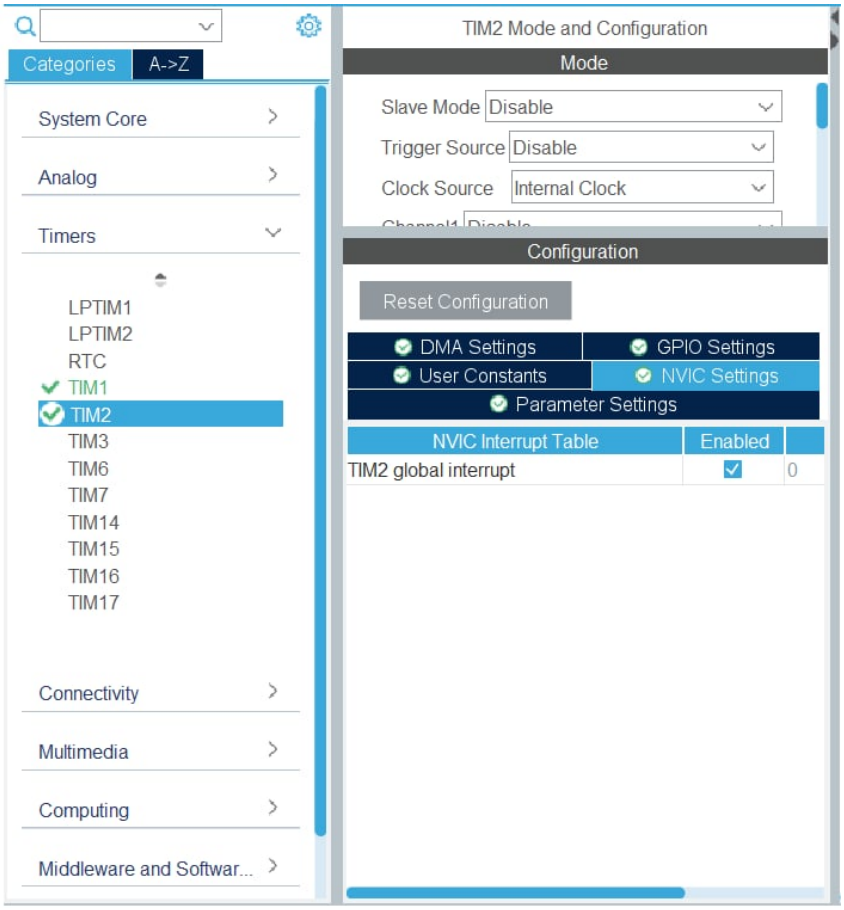
7.10 Prescaler dla Timer 2



7.11 Ustawienie nazw `TRIG` i `ECHO`



7.12 Włączenie przerwań dla Timer 2



8 Opis najważniejszych funkcji kodu

8.1 Funkcje sterujące ruchem robota

Poniższe funkcje odpowiadają za sterowanie kierunkiem ruchu robota poprzez ustawianie odpowiednich stanów logicznych na pinach mostka H, który zasila silniki robota. Każda funkcja realizuje inny manewr: jazdę do przodu, do tyłu, skręt w lewo, skręt w prawo oraz zatrzymanie.

```c
void jedz_tyl(void) {
    kierunek_aktualny = 1;

    HAL_GPIO_WritePin(IN1_GPIO_Port, IN1_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(IN2_GPIO_Port, IN2_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(IN3_GPIO_Port, IN3_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(IN4_GPIO_Port, IN4_Pin, GPIO_PIN_SET);
}

void jedz_przod(void) {
    HAL_GPIO_WritePin(IN1_GPIO_Port, IN1_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(IN2_GPIO_Port, IN2_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(IN3_GPIO_Port, IN3_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(IN4_GPIO_Port, IN4_Pin, GPIO_PIN_RESET);
}

void zatrzymaj(void) {
    kierunek_aktualny = 0;
    HAL_GPIO_WritePin(IN1_GPIO_Port, IN1_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(IN2_GPIO_Port, IN2_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(IN3_GPIO_Port, IN3_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(IN4_GPIO_Port, IN4_Pin, GPIO_PIN_RESET);
}

void hamuj(void) {
    HAL_GPIO_WritePin(IN1_GPIO_Port, IN1_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(IN2_GPIO_Port, IN2_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(IN3_GPIO_Port, IN3_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(IN4_GPIO_Port, IN4_Pin, GPIO_PIN_SET);
    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, 0);
    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, 0);
}

void skrec_lewo(void) {
    HAL_GPIO_WritePin(IN1_GPIO_Port, IN1_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(IN2_GPIO_Port, IN2_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(IN3_GPIO_Port, IN3_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(IN4_GPIO_Port, IN4_Pin, GPIO_PIN_SET);
}

void skrec_prawo(void) {
    HAL_GPIO_WritePin(IN1_GPIO_Port, IN1_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(IN2_GPIO_Port, IN2_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(IN3_GPIO_Port, IN3_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(IN4_GPIO_Port, IN4_Pin, GPIO_PIN_RESET);
}
```

8.2 Funkcje obsługi czujnika odległości

Te funkcje odpowiadają za obsługę czujnika ultradźwiękowego (np. HC-SR04), który mierzy odległość od przeszkody. **wyslij_trigger(void)** generuje krótki impuls na pinie TRIG, który inicjuje pomiar odległości przez czujnik. **HAL_TIM_IC_CaptureCallback(...)** to funkcja wywoływana w przerwaniu timera po wykryciu zbocza sygnału echa. Oblicza czas trwania impulsu i na tej podstawie wyznacza odległość. **`void zapisz_pomiar(uint32_t nowy)`** zapisuje najnowszy pomiar odległości do bufora cyklicznego, co umożliwia późniejszą filtrację lub uśrednianie wyników w celu zwiększenia odporności na zakłócenia.

```c
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim) {
    if (htim->Instance == TIM3) {
        static uint32_t start = 0;
        static uint8_t przechwycone = 0;
        if(!przechwycone) {
            start = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_3);
            __HAL_TIM_SET_CAPTUREPOLARITY(htim,
            TIM_CHANNEL_3, TIM_INPUTCHANNELPOLARITY_FALLING);
            przechwycone = 1;
        } else {
            uint32_t impuls = HAL_TIM_ReadCapturedValue(htim,
            TIM_CHANNEL_3) - start;
            odleglosc_cm = impuls / 58;
            zapisz_pomiar(odleglosc_cm);
            __HAL_TIM_SET_CAPTUREPOLARITY(htim,
            TIM_CHANNEL_3, TIM_INPUTCHANNELPOLARITY_RISING);
            przechwycone = 0;
        }
    }
}

void wyslij_trigger(void) {
    HAL_GPIO_WritePin(TRIG_GPIO_Port, TRIG_Pin, GPIO_PIN_SET);
    for (volatile int i = 0; i < 160; i++);
    HAL_GPIO_WritePin(TRIG_GPIO_Port, TRIG_Pin, GPIO_PIN_RESET);
}

void zapisz_pomiar(uint32_t nowy) {
    pomiary[idx++] = nowy;
    if (idx >= FILTR_OKIENKO) idx = 0;
}
```

8.3 Funkcje komunikacji i obsługi poleceń

Funkcja obsługuje odbiór znaków przez UART, czyli komunikację z komputerem lub aplikacją sterującą. Na podstawie odebranego znaku wywołuje odpowiednią funkcję ruchu lub zmienia prędkość silników przez ustawienie wartości PWM.

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{  if (huart->Instance == USART2)
    {
        last_command_time = HAL_GetTick();
        switch (znak_odbioru){
        case 'F':
            ustaw_kierunek_i_predkosc(jedz_przod);
            break;
        case 'B':
            ustaw_kierunek_i_predkosc(jedz_tyl);
            break;
        case 'L':
            ustaw_kierunek_i_predkosc(skrec_lewo);
            break;
        case 'R':
            ustaw_kierunek_i_predkosc(skrec_prawo);
            break;
            case 'S':
                zatrzymaj();
                break;
            case '1':
                aktualny_pwm = 444;
                if (kierunek_aktualny != 0) {
                    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1,
                    aktualny_pwm);
                    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2,
                    aktualny_pwm);}
                break;
            case '3':
                aktualny_pwm = 666;
                if (kierunek_aktualny != 0) {
                    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1,
                    aktualny_pwm);
                    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2,
                    aktualny_pwm);
                }
                break;
            case 'q':
                aktualny_pwm = 999;
                if (kierunek_aktualny != 0) {
                    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1,
                    aktualny_pwm);
                    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2,
                    aktualny_pwm);}
                break;
            default:
                break;}
        znak_odbioru = 0;
        HAL_UART_Receive_IT(&huart2, &znak_odbioru, 1);}}
```

9 Pętla główna

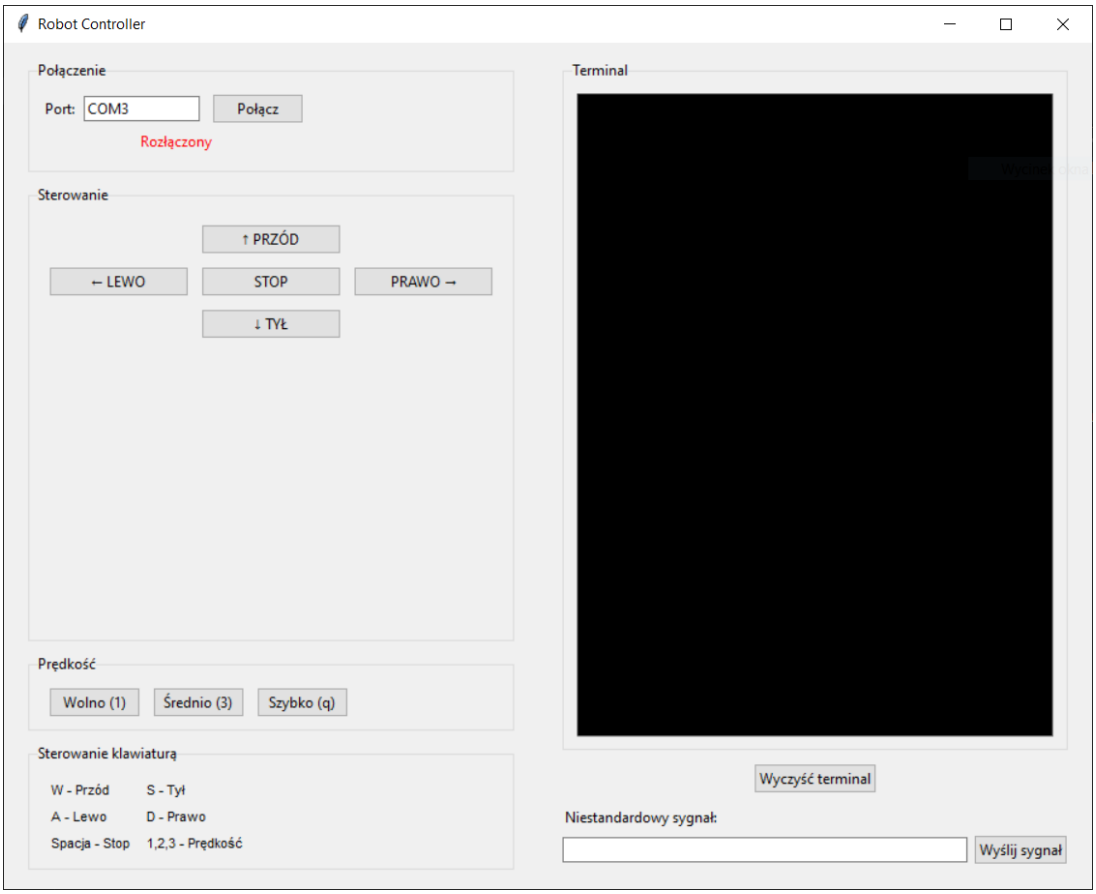
Pętla główna programu odpowiada za ciągłe monitorowanie otoczenia pojazdu oraz podejmowanie decyzji o manewrach w zależności od wykrytych przeszkód. Gdy nie trwa aktualnie manewr omijania (`stan_omijania == 0`), co określa zmienna stanu, wysyłany jest impuls `trigger` do czujnika ultradźwiękowego w celu wykonania pomiaru odległości. Jeśli wykryta zostanie przeszkoda w odległości mniejszej niż 20 cm, pojazd inicjuje procedurę omijania: zatrzymuje się, cofa, skręca i ewentualnie ponownie cofa w zależności od aktualnego kierunku jazdy. Cały manewr odbywa się przy pomocy prostych funkcji ruchu oraz odpowiednich opóźnień czasowych, a jego przebieg kontrolowany jest przez zmianę stanu omijania.

```
while (1)
 {
     if (stan_omijania == 0) {
             wyslij_trigger();
             HAL_Delay(60);
         }

     if (odleglosc_cm > 0 && odleglosc_cm < 20 && stan_omijania == 0) {
         stan_omijania = 1;

         zatrzymaj();
         HAL_Delay(100);

         __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, aktualny_pwm);
         __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, aktualny_pwm);
         jedz_tyl();
         HAL_Delay(500);

         zatrzymaj();
         HAL_Delay(100);

         skrec_prawo();
         HAL_Delay(500);

         zatrzymaj();
         HAL_Delay(100);

         if (kierunek_aktualny == 1) {
             jedz_tyl();
         } else {
             zatrzymaj();
         }

         HAL_Delay(200);
         stan_omijania = 0;
     }

 }
```

10 Aplikacja Sterująca

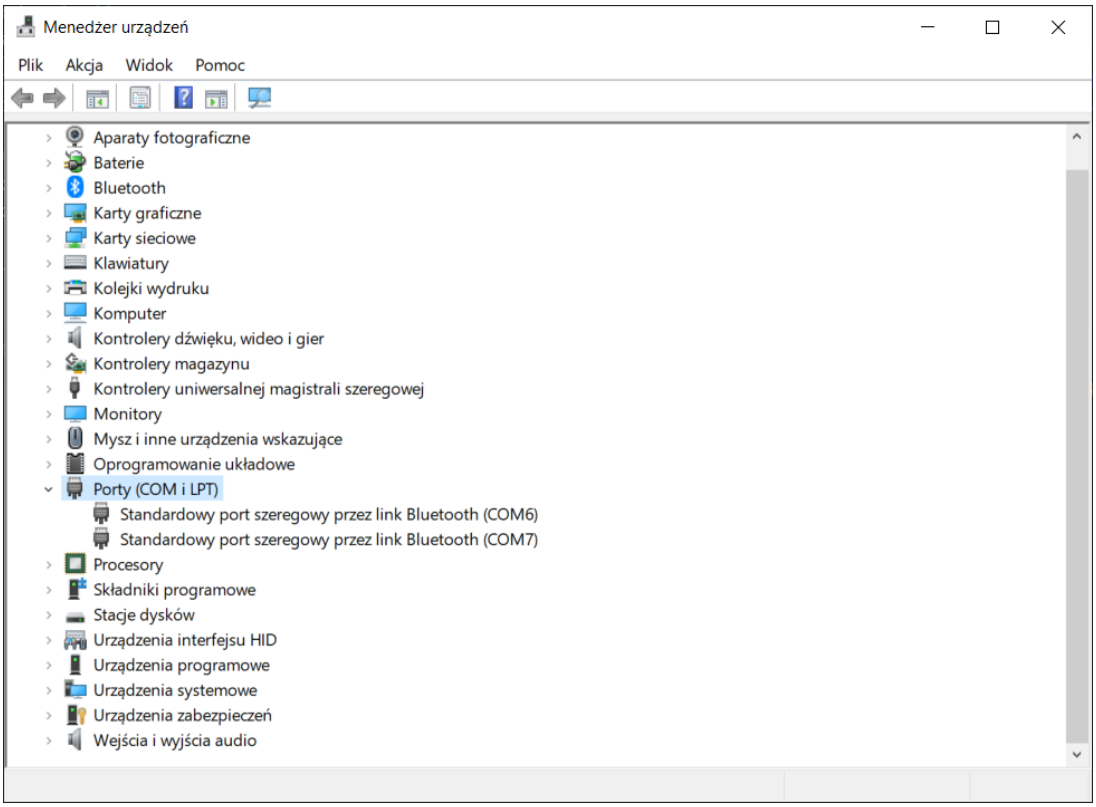
W większości podobnych projektów narzędzia do sterowania robotem dostępne są wyłącznie na urządzenia z systemem Android. Z tego względu zaprojektowaliśmy własną, uniwersalną aplikację działającą na systemie Windows. Program **rcar-control-app** został napisany w języku Python z użyciem biblioteki **Tkinter** i umożliwia sterowanie robotem przez port szeregowy (np. Bluetooth – moduł HC-05 lub USB). Intuicyjny interfejs pozwala na szybki wybór portu COM oraz łatwe nawiązanie połączenia z urządzeniem.

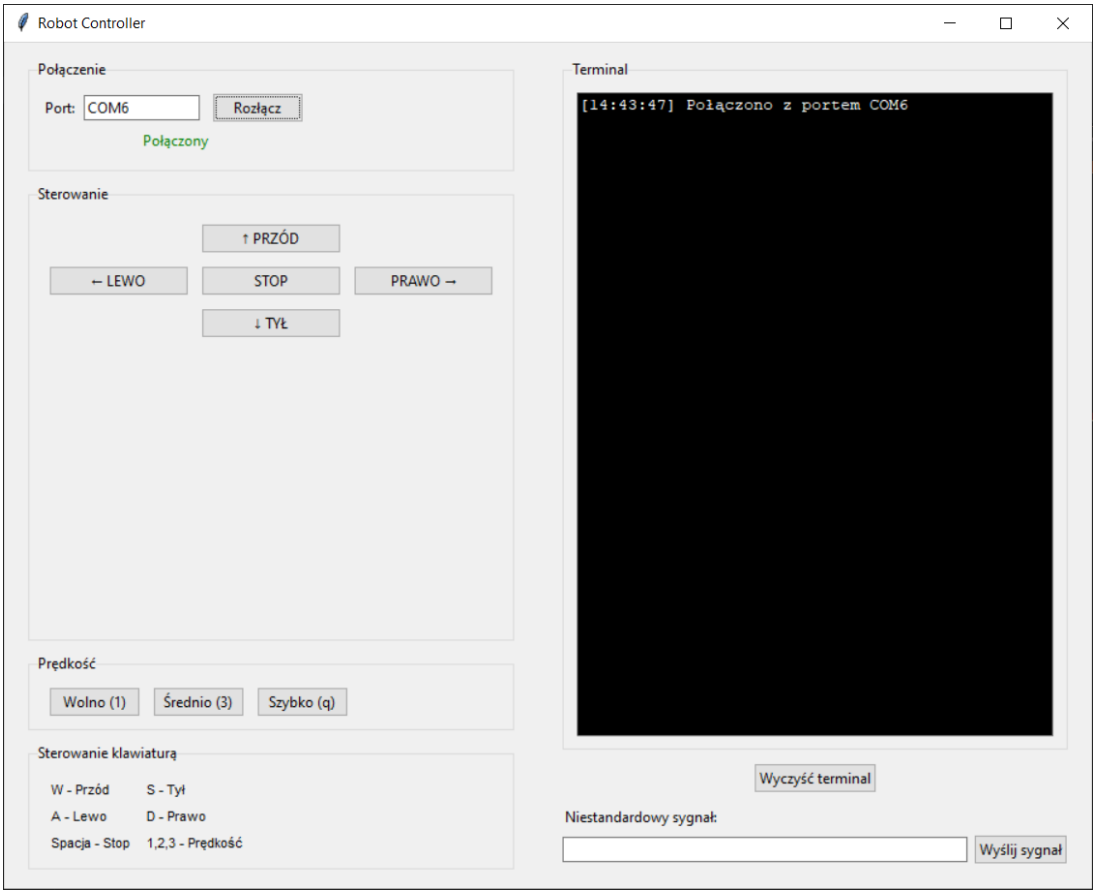
Kod aplikacji dostępny jest pod adresem: `https://github.com/szymon-tulodziecki/rcar-control-app/blob/main/rcar-control-app.py`



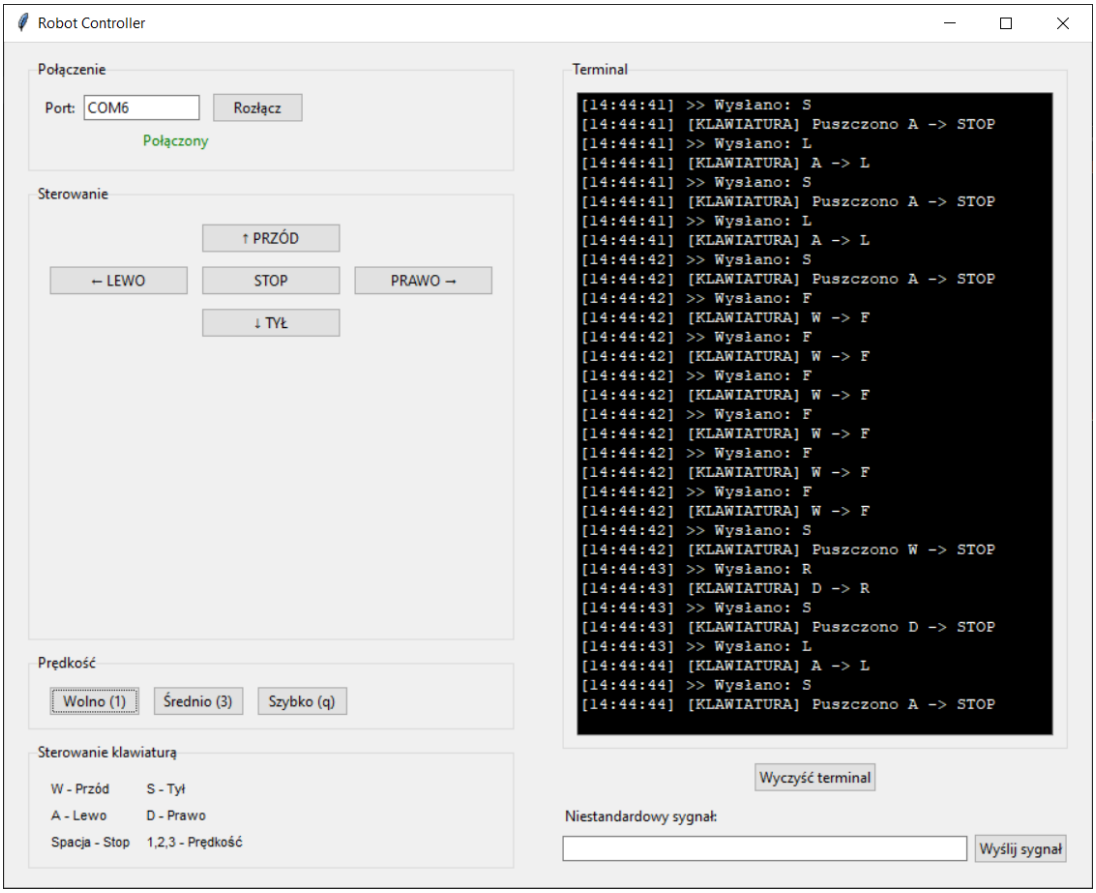
Aplikacja umożliwia sterowanie ruchem robota zarówno za pomocą klawiatury, jak i przycisków graficznych GUI. Dostępne są wszystkie podstawowe kierunki jazdy: przód, tył, lewo, prawo oraz zatrzymanie. Użytkownik może również zmieniać tryb prędkości (wolny, średni, szybki) oraz monitorować komunikację szeregową dzięki wbudowanemu terminalowi, który rejestruje wszystkie wysyłane i odbierane sygnały wraz ze znacznikami czasu.

Po uruchomieniu programu użytkownik wybiera odpowiedni port COM z listy i klika przycisk **Połącz**, co inicjuje komunikację z robotem. Interfejs jest przejrzysty i zawiera wszystkie niezbędne elementy do pełnej obsługi pojazdu, a wbudowany terminal znacząco ułatwia debugowanie połączenia.


Menedżer urządzeń
Plik
Akcja
Widok
Pomoc
Aparaty fotograficzne
Baterie
Bluetooth
Karty graficzne
Karty sieciowe
Klawiatury
Kolejki wydruku
Komputer
Kontrolery dźwięku, wideo i gier
Kontrolery magazynu
Kontrolery uniwersalnej magistrali szeregowej
Monitory
Mysz i inne urządzenia wskazujące
Oprogramowanie układowe
Porty (COM i LPT)
Standardowy port szeregowy przez link Bluetooth (COM6)
Standardowy port szeregowy przez link Bluetooth (COM7)
Procesory
Składniki programowe
Stacje dysków
Urządzenia interfejsu HID
Urządzenia programowe
Urządzenia systemowe
Urządzenia zabezpieczeń
Wejścia i wyjścia audio


Robot Controller
Połączenie
Port:
COM6
Rozłącz
Połączony
Sterowanie
↑ PRZÓD
← LEWO
STOP
PRAWO →
↓ TYŁ
Prędkość
Wolno (1)
Średnio (3)
Szybko (q)
Sterowanie klawiaturą
W - Przód
S - Tył
A - Lewo
D - Prawo
Spacja - Stop
1,2,3 - Prędkość
Terminal
[14:43:47] Połączono z portem COM6
Wyczyść terminal
Niestandardowy sygnał:
Wyślij sygnał

Dodatkowo aplikacja umożliwia ręczne wpisywanie i wysyłanie niestandardowych komend, co pozwala na testowanie specyficznych funkcji robota lub obsługę niestandardowych protokołów komunikacyjnych. Obsługa robota możliwa jest także przy użyciu skrótów klawiaturowych: **W** – przód, **S** – tył, **A** – lewo, **D** – prawo, **Spacja** – stop, a także **1**, **2**, **3** do zmiany prędkości jazdy.



Aplikacja została zaprojektowana jako uniwersalne narzędzie do testowania i obsługi pojazdów RC, robotów mobilnych oraz innych urządzeń komunikujących się przez UART/Serial. Stanowi wygodną alternatywę dla aplikacji mobilnych dostępnych na Androida i znacząco zwiększa komfort pracy podczas rozwoju i testowania systemów robotycznych.

11 Wnioski

Nauczyliśmy się, jak zaprojektować i zbudować kompletny system wbudowany – od hardware'u, przez oprogramowanie mikrokontrolera, aż po własną aplikację sterującą na PC. Praca nad projektem pozwoliła zdobyć praktyczne doświadczenie z obsługą czujników, sterowaniem napędami, komunikacją UART/Bluetooth oraz integracją różnych podzespołów w jeden, działający układ. Poznaliśmy też realne wyzwania związane z montażem elektroniki, debugowaniem kodu i testowaniem całego systemu w praktyce.

Jednocześnie projekt pokazał, że STM32 jest fajny do nauki i prototypowania, ale trzeba z nim uważać – mikrokontrolery te są dość wrażliwe na błędy w zasilaniu czy podłączeniach i bardzo łatwo je uszkodzić. To uczy pokory, dokładności i planowania pracy, bo jeden drobny błąd może kosztować sporo czasu (i nerwów).

Podsumowując: projekt nauczył nas nie tylko technicznych podstaw systemów wbudowanych, ale też organizacji pracy, dokumentowania i radzenia sobie z typowymi problemami, które pojawiają się przy budowie własnych urządzeń elektronicznych.