


Milestone 3





Link do repozytorium projektu: <https://github.com/kacprow21324/STM32F3Discovery-Autonomiczny-Pojazd>


1. Zwięzły opis robota

Projekt zakłada stworzenie zaawansowanego modelu pojazdu autonomicznego opartego na mikrokontrolerze STM32F3Discovery, który może funkcjonować w dwóch trybach działania: półautomatycznym oraz w pełni autonomicznym. W trybie półautomatycznym pojazd realizuje komendy wydawane przez użytkownika za pośrednictwem interfejsu UART, który umożliwia komunikację np. za pomocą modułu Bluetooth HC-05. W trybie automatycznym pojazd samodzielnie analizuje otoczenie przy użyciu zestawu sensorów ultradźwiękowych (HC-SR04) oraz optycznych (czujniki IR TCRT5000), co pozwala mu wykrywać i zatrzymać się, jeżeli wykryje daną przeszkodę oraz precyzyjnie podążać za namalowaną na podłożu linią trasy. Sterowanie napędem jest realizowane za pomocą sygnałów PWM generowanych przez wbudowane timery mikrokontrolera, umożliwiających precyzyjną regulację prędkości jazdy oraz płynność manewrów. System sterowania opiera się na efektywnej obsłudze przerwań oraz ewentualnie na systemie operacyjnym czasu rzeczywistego (RTOS), zapewniającym dokładne zarządzanie zadaniami. Użytkownik poprzez komendy UART może dynamicznie zmieniać parametry jazdy, odczytywać aktualny stan czujników oraz przełączać tryby pracy pojazdu, co umożliwia pełną kontrolę oraz analizę danych w czasie rzeczywistym.

2. Elementy wybrane do budowy robota

Lp.	Komponent	Ilość	Zdjęcie
1	HC-05 – bezprzewodowy moduł Bluetooth	1	
2	HC-SR04 czujnik ultradźwiękowy	2	
3	Koszyk na 6 baterii typu AA (R6)	1	

4	L298N – płyta sterownika silników DC	1	
5	Płytki stykowe justPi – 400 otworów	1	
6	Silniki DC TT z przekładnią i kołami (3–6 V, wał podwójny, 65 mm)	4	
7	STM32F3-Discovery (STM32F3DISCOVERY)	1	

8	TCRT5000 – czujnik odblaskowy IR	5		
---	----------------------------------	---	---	--

3. Mechanika robota

Deseczka jako podstawa robota

- Wymiary: 14,5 cm × 22 cm × 0,5 cm, cienka sklejka.
- Wywierciłem w niej otwór pod kable prowadzące do czujników TCRT5000 – dzięki temu przewody nie plączą się pod robotem.

Mocowanie silników DC

- Przyklejone bezpośrednio do spodniej strony deseczki klejem Pattex (do klejenia drewna i innych tworzyw sztucznych).
- Klej trzyma bardzo mocno, a nie dodaje praktycznie żadnej masy.

Sterownik silników (L298N)

- Przytwierdziłem do kawałka dokładnie dociętego styropianu – Całość przymocowałem do deseczki taśmą dwustronną.

Płytki STM32F3-Discovery

- Zamontowana jest na kawałku plastikowego bloczka z zestawu Jengi (oczywiście przyklejona z dwóch stron).

Czujniki TCRT5000

- Przykleiłem je na przedniej lewej i prawej stronie robota, za pomocą kleju Pattex (do klejenia drewna i innych tworzyw sztucznych).
- Przewody poprowadzone przez otwór w deseczce, dzięki czemu nie przeszkadzają w jeździe.
- W celu poprawy śledzenia linii zwiększyłem liczbę czujników TCRT5000 z 2 do 5, co znacznie poprawiło dokładność i stabilność jazdy.

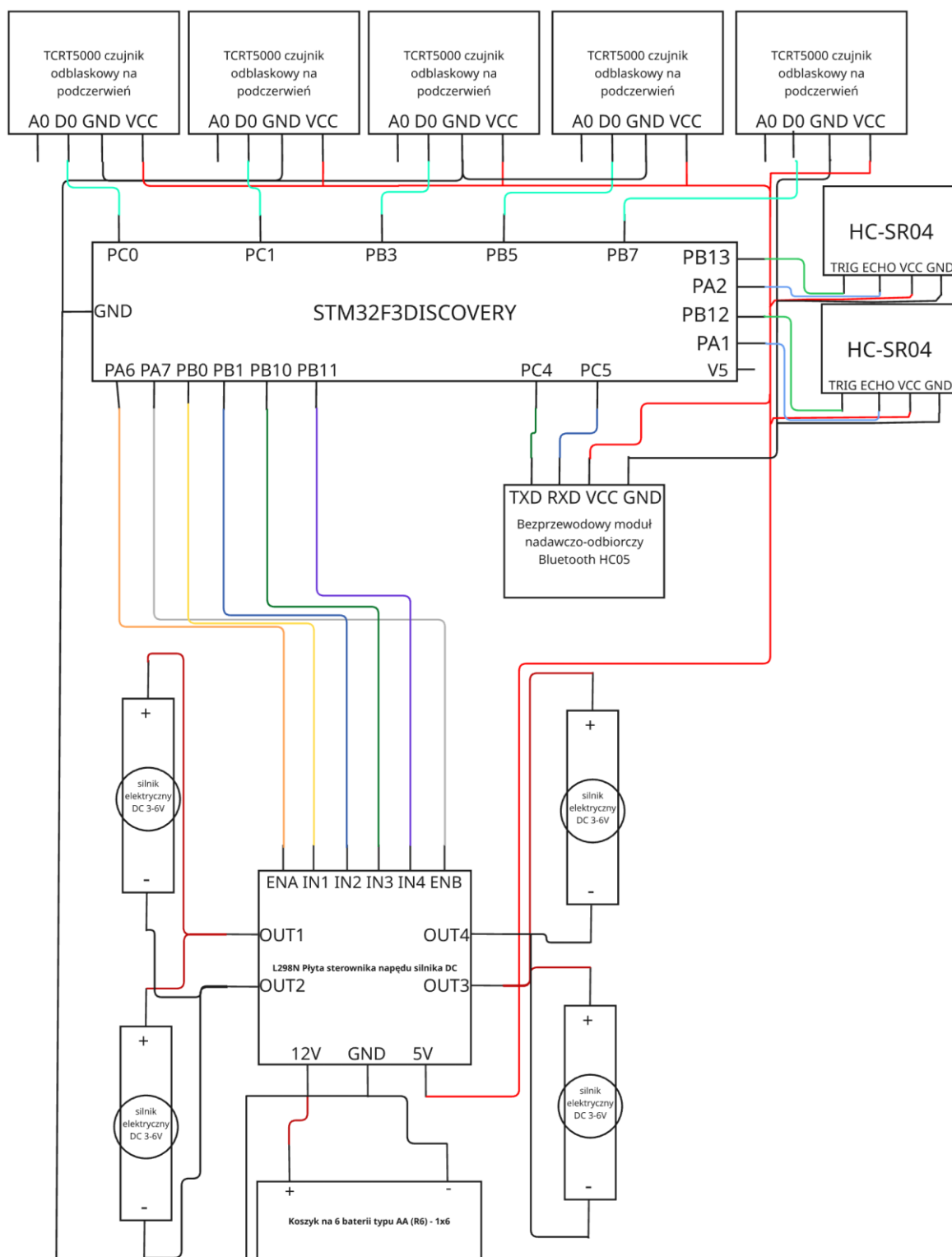
Czujniki ultradźwiękowe HC-SR04

- Przyklejona podstawa specjalnie do tych czujników za pomocą taśmy dwustronnej i żeby nie spadały przykleiłem je klejem.

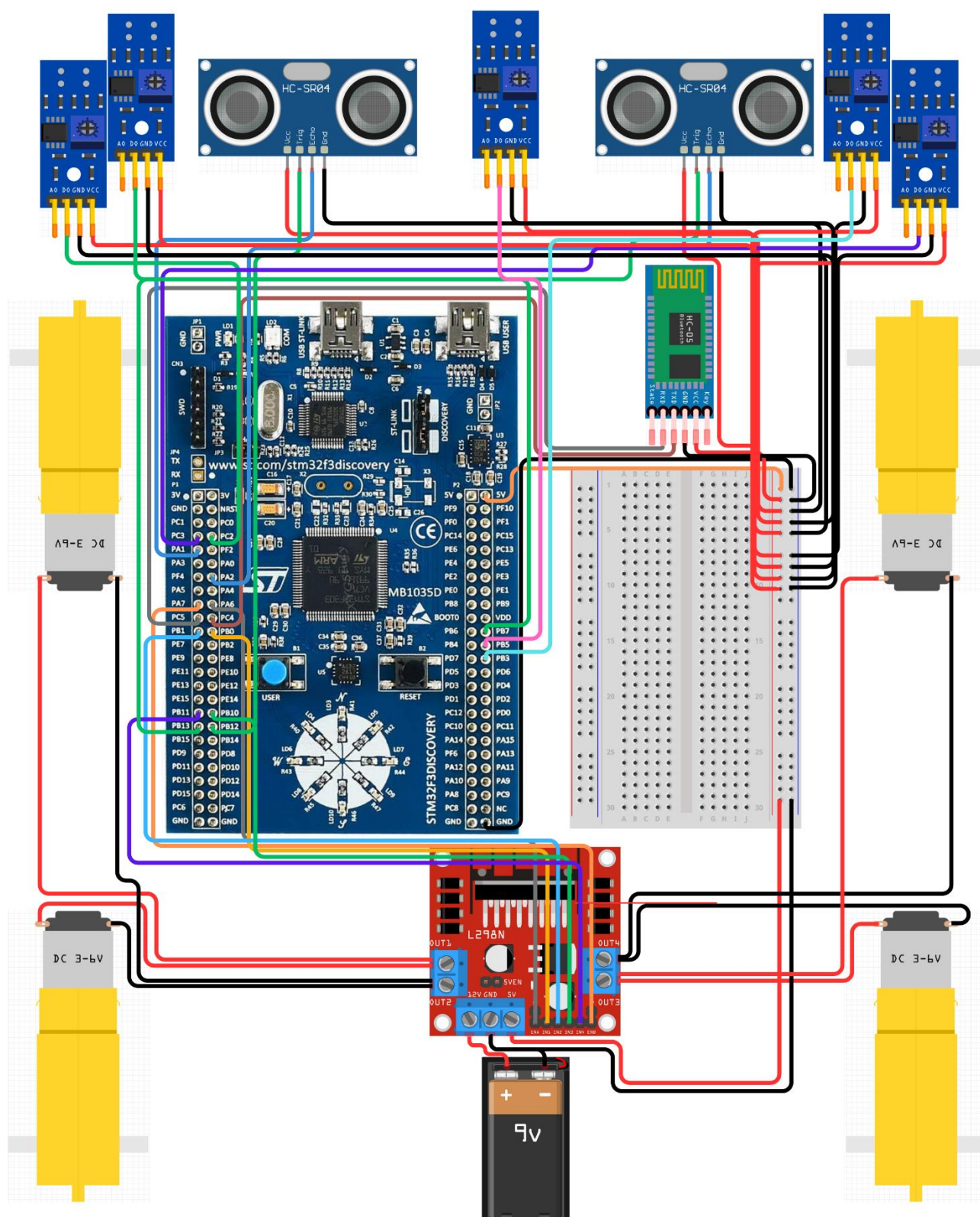
Okablowanie

- Wszystkie przewody sygnałowe i zasilające zakończyłem złączkami żeńsko-męskimi, żeńsko-żeńskimi i męsko-męskimi.
- Prowadzenie kabli wzdłuż krawędzi deseczki zabezpieczyłem punktowo taśmą, żeby nic się nie plątało pod kołami.

4. Schemat elektroniczny robota

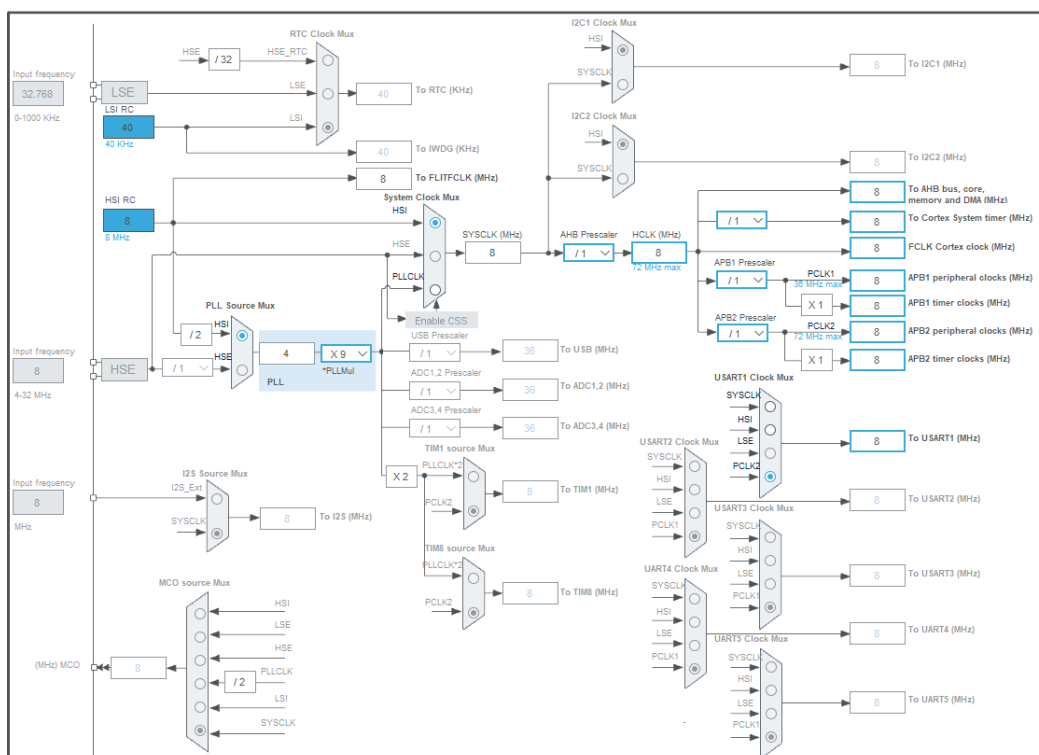
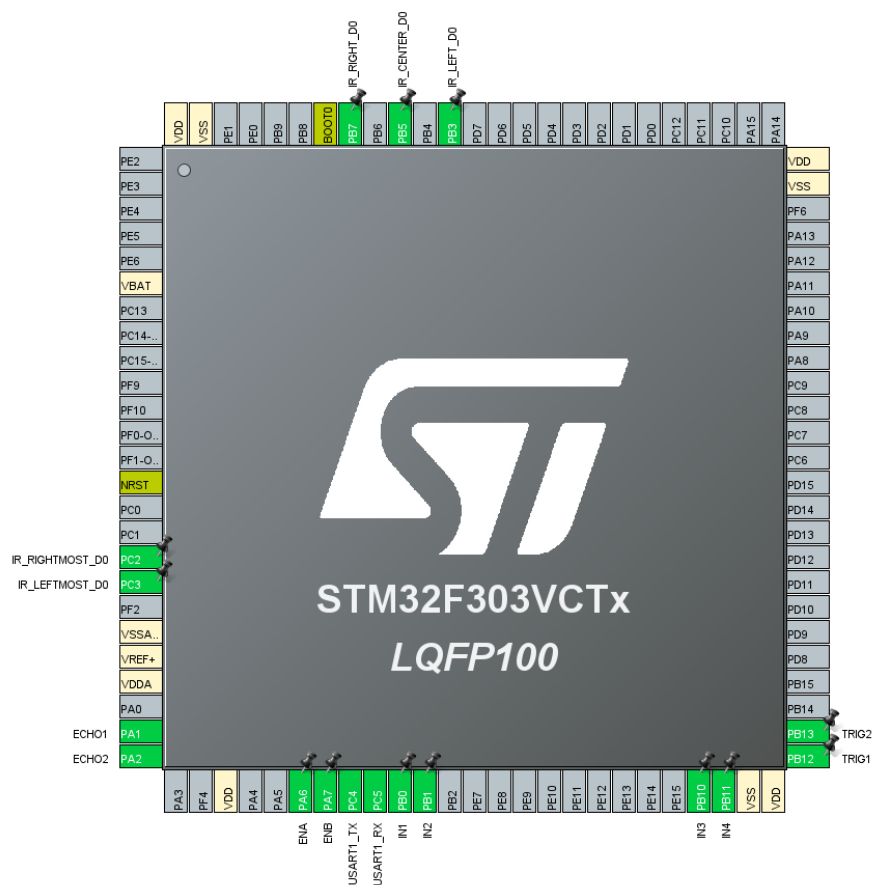


Rys. 4.1: Schemat uproszczony układu elektronicznego robota (zrobiony za pomocą programu Microsoft Paint).



Rys. 4.2: Schemat montażowy robota z płytką stykową (zrobiony za pomocą programu Canva).

5. Oprogramowanie sterujące



Categories

A-Z

System Core >

Analog >

Timers >

RTC

TIM1

TIM2

TIM3

TIM4

TIM6

TIM7

TIM8

TIM15

TIM16

TIM17

Connectivity >

CAN

I2C1

I2C2

IRTIM

SPI1

SPI2

SPI3

UART4

UART5

USART1

USART2

USART3

USB

Multimedia >

Computing >

Middleware and Software Packs >

Mode

Slave Mode

Disable

Trigger Source

Disable

Clock Source

Internal Clock

Channel1

PWM Generation CH1

Channel2

PWM Generation CH2

Channel3

Disable

Channel4

Disable

Combined Channels

Disable

☐ ETR IO as Clearing Source

☐ XOR activation

☐ One Pulse Mode

Configuration

Reset Configuration

DMA Settings

GPIO Settings

Parameter Settings

User Constants

NVIC Settings

Configure the below parameters :

Search (Ctrl+F)

Counter Settings

Prescaler (PSC - 16 bi...

7

Counter Mode

Up

Counter Period (AutoR...

99

Internal Clock Division ...

No Division

auto-reload preload

Disable

Trigger Output (TRGO) Param...

Master/Slave Mode (M...

Disable (Trigger input effect not d...

Trigger Event Selectio...

Reset (UG bit from TIMx_EGR)

Clear Input

Clear Input Source

Disable

PWM Generation Channel 1

Mode

PWM mode 1

Pulse (16 bits value)

0

Output compare preload

Enable

Fast Mode

Disable



USART1 Mode and Configuration

Categories A-Z

System Core

- DMA
- ✓ GPIO
- IWDG
- ✓ NVIC
- RCC
- ✓ SYS
- ⚠ TSC
- WWDG

Analog

Timers

- RTC
- TIM1
- ✓ TIM2
- ✓ TIM3
- TIM4
- TIM6
- TIM7
- TIM8
- TIM15
- TIM16
- TIM17

Connectivity

- CAN
- I2C1
- I2C2
- IRTIM
- SPI1
- SPI2
- SPI3
- UART4
- UART5
- ✓ USART1
- USART2
- USART3
- USB

Multimedia

Computing

Mode

Mode Asynchronous

Hardware Flow Control (RS232) Disable

☐ Hardware Flow Control (RS485)

Configuration

Reset Configuration

✓ NVIC Settings

✓ DMA Settings

✓ GPIO Settings

✓ Parameter Settings

✓ User Constants

Configure the below parameters :

Search (Ctrl+F)

Basic Parameters

- Baud Rate 9600 Bits/s
- Word Length 8 Bits (including Parity)
- Parity None
- Stop Bits 1

Advanced Parameters

- Data Direction Receive and Transmit
- Over Sampling 16 Samples
- Single Sample Disable

Advanced Features

- Auto Baudrate Disable
- TX Pin Active Level Inver... Disable
- RX Pin Active Level Inve... Disable
- Data Inversion Disable
- TX and RX Pins Swapping Disable
- Overrun Enable

[illegible]



Fragmenty kodu pochodzące z pliku main.c:

```
volatile uint8_t Auto = 0;
volatile uint8_t procentPredkosci = 10;
char znak;
int ostatniKierunek = 0;
uint32_t lewaOdleglosc = 0;
uint32_t prawaOdleglosc = 0;
volatile uint8_t pokazOdleglosc = 0;
```

Na początku ustawiono trzy makra: PREDKOSC_MIN i PREDKOSC_MAX określają minimalny i maksymalny procent wypełnienia PWM (0–100 %), a KROK_PREDKOSCI definiuje, o ile procent zmienia się prędkość przy każdej komendzie. Zadeklarowano dwie zmienne typu volatile uint8_t: Auto (flaga trybu automatycznego) i procentPredkosci (bieżący procent wypełnienia), oraz bufor na pojedynczy znak. Następnie int ostatniKierunek = 0; zapamiętuję, czy ostatnio jechałem prosto (0), skręcałem w lewo (1) czy w prawo (2). Zmienne uint32_t lewaOdleglosc = 0 i uint32_t prawaOdleglosc = 0 przechowują ostatnie zmierzone wartości odległości z dwóch czujników ultradźwiękowych (w centymetrach). Na koniec volatile uint8_t pokazOdleglosc = 0; to flaga, która w pętli głównej informuje, że należy wywołać funkcję wysyłającą zmierzone odległości przez UART.

```
void aktualizujPredkosc(void)
{
    uint32_t predkosc = (htim3.Init.Period + 1)*procentPredkosci/100;
    __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_1, predkosc);
    __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_2, predkosc);
}
```

Funkcja aktualizujPredkosc() przelicza wartość rejestru porównawczego: $(\text{htim3.Init.Period} + 1) * \text{procentPredkosci} / 100$, co zapewnia liniową zależność wypełnienia sygnału od procentu. Wynik przypisywany jest jednocześnie do obu kanałów timera TIM3 (kanały 1 i 2) przez makra HAL:

__HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_x, predkosc);.

```
void zatrzymajSilniki(void)
{
    __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_1, 0);
    __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_2, 0);
}
```

Funkcja zatrzymajSilniki() wyłącza sygnał PWM na obu kanałach silników (TIM_CHANNEL_1 i TIM_CHANNEL_2), ustawiając wartość porównania na 0. W praktyce oznacza to 0 % wypełnienia sygnału PWM, czyli silniki przestają dostawać sygnał i natychmiast się zatrzymują.

```
void jedzNaprzod(void)
{
    HAL_GPIO_WritePin(IN1_GPIO_Port, IN1_Pin, 1);
    HAL_GPIO_WritePin(IN2_GPIO_Port, IN2_Pin, 0);
    HAL_GPIO_WritePin(IN3_GPIO_Port, IN3_Pin, 1);
    HAL_GPIO_WritePin(IN4_GPIO_Port, IN4_Pin, 0);
    aktualizujPredkosc();
}
```



Funkcja jedzNaprzod() ustawia piny kierunkowe tak, aby oba silniki jechały naprzód (IN1=1, IN2=0; IN3=1, IN4=0), a następnie wywołuje aktualizujPredkosc(), która oblicza i uruchamia PWM o zadanym procentowym wypełnieniu – w efekcie robot rusza do przodu.

```
void jedzDoTyłu(void)
{
    HAL_GPIO_WritePin(IN1_GPIO_Port, IN1_Pin, 0);
    HAL_GPIO_WritePin(IN2_GPIO_Port, IN2_Pin, 1);
    HAL_GPIO_WritePin(IN3_GPIO_Port, IN3_Pin, 0);
    HAL_GPIO_WritePin(IN4_GPIO_Port, IN4_Pin, 1);
    aktualizujPredkosc();
}
```

Funkcja jedzDoTyłu() ustawia piny kierunkowe odwrotnie (IN1=0, IN2=1; IN3=0, IN4=1), dzięki czemu oba silniki kręcą się w tył, a potem wywołuje aktualizujPredkosc(), aby nadać im zadaną moc – robot jedzie do tyłu.

```
void skrecWLewo(void)
{
    HAL_GPIO_WritePin(IN1_GPIO_Port, IN1_Pin, 1);
    HAL_GPIO_WritePin(IN2_GPIO_Port, IN2_Pin, 0);
    HAL_GPIO_WritePin(IN3_GPIO_Port, IN3_Pin, 0);
    HAL_GPIO_WritePin(IN4_GPIO_Port, IN4_Pin, 1);
    aktualizujPredkosc();
}
```

Funkcja skrecWLewo() Ustawia lewy silnik do przodu (IN1=1, IN2=0) i prawy do tyłu (IN3=0, IN4=1), a następnie wywołuje aktualizujPredkosc(), co powoduje obrót robota w lewo z aktualnie ustawioną prędkością.

```
void skrecWPrawo(void)
{
    HAL_GPIO_WritePin(IN1_GPIO_Port, IN1_Pin, 0);
    HAL_GPIO_WritePin(IN2_GPIO_Port, IN2_Pin, 1);
    HAL_GPIO_WritePin(IN3_GPIO_Port, IN3_Pin, 1);
    HAL_GPIO_WritePin(IN4_GPIO_Port, IN4_Pin, 0);
    aktualizujPredkosc();
}
```

Funkcja skrecWPrawo() ustawia lewy silnik do tyłu (IN1=0, IN2=1) i prawy do przodu (IN3=1, IN4=0), a potem uruchamia aktualizujPredkosc(), wprowadzając robota w obrót w prawo.

```
void zatrzymajRobota(void)
{
    zatrzymajSilniki();
    HAL_GPIO_WritePin(IN1_GPIO_Port, IN1_Pin, 0);
    HAL_GPIO_WritePin(IN2_GPIO_Port, IN2_Pin, 0);
    HAL_GPIO_WritePin(IN3_GPIO_Port, IN3_Pin, 0);
    HAL_GPIO_WritePin(IN4_GPIO_Port, IN4_Pin, 0);
}
```

Funkcja zatrzymajSilniki() ustawia wszystkie piny kierunkowe IN1–IN4 na 0, odcinając zasilanie silników; nie wywołuje PWM, dzięki czemu robot natychmiast się zatrzymuje.



```
void jedzPoLinii(void)
{
    lewaOdleglosc = zmierzOdleglosc(GPIOB, TRIG1_Pin, GPIOA, GPIO_PIN_1);
    prawaOdleglosc = zmierzOdleglosc(GPIOB, TRIG2_Pin, GPIOA, GPIO_PIN_2);

    if (lewaOdleglosc < 15 || prawaOdleglosc < 15)
    {
        zatrzymajRobota();
        return;
    }

    aktualizujPredkosc();

    uint8_t skrajnieLewySensor = HAL_GPIO_ReadPin(IR_LEFTMOST_D0_GPIO_Port, IR_LEFTMOST_D0_Pin);
    uint8_t lewySensor = HAL_GPIO_ReadPin(IR_LEFT_D0_GPIO_Port, IR_LEFT_D0_Pin);
    uint8_t centralnySensor = HAL_GPIO_ReadPin(IR_CENTER_D0_GPIO_Port, IR_CENTER_D0_Pin);
    uint8_t prawySensor = HAL_GPIO_ReadPin(IR_RIGHT_D0_GPIO_Port, IR_RIGHT_D0_Pin);
    uint8_t skrajniePrawySensor = HAL_GPIO_ReadPin(IR_RIGHTMOST_D0_GPIO_Port,
    IR_RIGHTMOST_D0_Pin);

    if (centralnySensor == 1)
    {
        jedzNaprzod();
        ostatniKierunek = 0;
    }
    else if (lewySensor == 1)
    {
        skrecwLewo();
        ostatniKierunek = 1;
    }
    else if (prawySensor == 1)
    {
        skrecwPrawo();
        ostatniKierunek = 2;
    }
    else if (skrajnieLewySensor == 1)
    {
        skrecwLewo();
        ostatniKierunek = 1;
    }
    else if (skrajniePrawySensor == 1)
    {
        skrecwPrawo();
        ostatniKierunek = 2;
    }
    else
    {
        jedzDoTyłu();
        HAL_Delay(30);

        if (ostatniKierunek == 1)
        {
            skrecwLewo();
        }
        else if (ostatniKierunek == 2)
        {
            skrecwPrawo();
        }
    }
}
```



```
else
{
    jedzNaprzod();
}

HAL_Delay(150);
zatrzymajSilniki();
}
```

Funkcja `jedzPoLinii()` zaczyna od pomiaru odległości z dwóch czujników ultradźwiękowych – jeśli którakolwiek wartość jest mniejsza niż 15 cm, wywołuje `zatrzymajRobota()` i natychmiast kończy działanie. Następnie wywołuje `aktualizujPredkosc()`, aby ustawić PWM zgodnie z bieżącym procentem `Predkosc`. Odczytuje pięć czujników IR: skrajnie lewy, lewy, centralny, prawy i skrajnie prawy. Jeśli centralny wykryje linię (1), jedzie prosto i ustawia `ostatniKierunek = 0`. Gdy wykryje to lewy lub skrajnie lewy (1), wywołuje `skrecWLewo()` i ustawia `ostatniKierunek = 1`. Analogicznie, gdy wykryje prawy lub skrajnie prawy (1), wywołuje `skrecWPrawo()` i ustawia `ostatniKierunek = 2`. W sytuacji, gdy żaden czujnik IR nie wykrywa linii, cofa się przez 30 ms, a następnie w zależności od wartości `ostatniKierunek` wykonuje skręt w lewo, w prawo lub jedzie prosto przez 150 ms, po czym wyłącza silniki (`zatrzymajSilniki()`).

```
void wyswietlOdleglosc(void)
{
    lewaOdleglosc = zmierzOdleglosc(GPIOB, TRIG1_Pin, GPIOA, GPIO_PIN_1);
    prawaOdleglosc = zmierzOdleglosc(GPIOB, TRIG2_Pin, GPIOA, GPIO_PIN_2);
    char tekst[64];
    sprintf(tekst, "Lewy: %lu cm | Prawy: %lu cm\r\n", lewaOdleglosc, prawaOdleglosc);
    HAL_UART_Transmit(&huart1, (uint8_t*)tekst, strlen(tekst), HAL_MAX_DELAY);
}
```

Funkcja `wyswietlOdleglosc()` najpierw wywołuje `zmierzOdleglosc()` dla lewego i prawego czujnika ultradźwiękowego, zapisując wyniki w zmiennych `lewaOdleglosc` i `prawaOdleglosc`. Następnie formatuje tekst w buforze tekst w postaci "Lewy: <wartość> cm | Prawy: <wartość> cm\r\n", po czym za pomocą `HAL_UART_Transmit()` wysyła go przez UART1 do odbiornika.

```
uint32_t zmierzOdleglosc(GPIO_TypeDef* trigPort, uint16_t trigPin, GPIO_TypeDef* echoPort,
uint16_t echoPin)
{
    uint32_t start = 0, stop = 0;
    uint32_t timeout = HAL_GetTick();

    HAL_GPIO_WritePin(trigPort, trigPin, 0);
    HAL_Delay(1);
    HAL_GPIO_WritePin(trigPort, trigPin, 1);
    __HAL_TIM_SET_COUNTER(&htim2, 0);

    while (__HAL_TIM_GET_COUNTER(&htim2) < 10);
    HAL_GPIO_WritePin(trigPort, trigPin, 0);

    while (HAL_GPIO_ReadPin(echoPort, echoPin) == 0)
    {
        if (HAL_GetTick() - timeout > 10) return 0;
    }
    start = __HAL_TIM_GET_COUNTER(&htim2);
```




```
timeout = HAL_GetTick();
while (HAL_GPIO_ReadPin(echoPort, echoPin) == 1)
{
    if (HAL_GetTick() - timeout > 50) return 0;
}
stop = __HAL_TIM_GET_COUNTER(&htim2);

uint32_t czas = (stop > start) ? (stop - start) : 0;

return (uint32_t)(czas * 0.017f);
}
```

Funkcja `zmierzOdleglosc()` wysyła impuls ultradźwiękowy i mierzy czas powrotu echa. Najpierw obniża linię TRIG na 1 ms, potem podnosi ją na chwilę, a timer TIM2 zeruje licznik i czeka 10 cykli, po czym obniża TRIG. Następnie czeka, aż na pinie ECHO pojawi się stan wysoki, zrywając pomiar, i zapisuje wartość timera jako start. Kolejno czeka, aż ECHO wróci do stanu niskiego, zapisując stop. Jeśli oczekiwanie na ECHO trwa zbyt długo (powyżej 10 ms przed pierwszym zboczem lub 50 ms przed drugim), funkcja zwraca 0. Różnicę między stop a start (w mikrosekundach zgodnie z prescalerem) mnoży przez 0.017, co daje odległość w centymetrach.

```
void obslugaMenu(char znak)
{
    switch (znak)
    {
        case 'M':
            if (procentPredkosci + KROK_PREDKOSCI <= PREDKOSC_MAX) {
                procentPredkosci += KROK_PREDKOSCI;
            } else {
                procentPredkosci = PREDKOSC_MAX;
            }
            aktualizujPredkosc();
            break;

        case 'N':
            if (procentPredkosci >= KROK_PREDKOSCI) {
                procentPredkosci -= KROK_PREDKOSCI;
            } else {
                procentPredkosci = PREDKOSC_MIN;
            }
            aktualizujPredkosc();
            break;

        case 'G':
            jedzNaprzod();
            break;

        case 'T':
            jedzDoTyłu();
            break;

        case 'L':
            skrecwLewo();
            break;

        case 'P':
            skrecwPrawo();
            break;

        case 'S':
            break;
    }
}
```



```
    zatrzymajRobota();  
    break;  
case 'A':  
    Auto = 1;  
    break;  
case 'a':  
    Auto = 0;  
    zatrzymajRobota();  
    break;  
case '0':  
    pokazOdleglosc = 1;  
    break;  
default:  
    break;  
}  
}
```

Obsługa poleceń z UART została wykonana za pomocą funkcji obsługiMenu(char znak) w którym znajduje się switch, reagujący na znaki:

- M zwiększa procentPredkosci o KROK_PREDKOSCI, aż do PREDKOSC_MAX;
- N zmniejsza o krok, aż do minimum;
- G, T, L, P, S wywołują odpowiednio: jazdę przód, tył, skręt w lewo, skręt w prawo oraz zatrzymanie;
- A i a włączają lub wyłączają tryb automatyczny (Auto = 1/0) i zatrzymują robot w przypadku wyłączenia.
- O ustawia pokazOdleglosc = 1, co w pętli głównej spowoduje wywołanie funkcji wyswietlOdleglosc()

```
int main(void)  
{  
  
    /* USER CODE BEGIN 1 */  
    /* USER CODE END 1 */  
  
    /* MCU Configuration-----*/  
  
    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */  
    HAL_Init();  
  
    /* USER CODE BEGIN Init */  
    /* USER CODE END Init */  
  
    /* Configure the system clock */  
    SystemClock_Config();  
  
    /* USER CODE BEGIN SysInit */  
    /* USER CODE END SysInit */  
  
    /* Initialize all configured peripherals */  
    MX_GPIO_Init();  
    MX_TIM3_Init();  
    MX_TIM2_Init();  
    MX_USART1_UART_Init();  
    /* USER CODE BEGIN 2 */  
    HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_1);  
}
```



```
HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_2);

aktualizujPredkosc();

HAL_TIM_Base_Start(&htim2);
HAL_UART_Receive_IT(&huart1, (uint8_t*)&znak, 1);

char *w = "Robot Działa\r\n";
HAL_UART_Transmit(&huart1, (uint8_t*)w, strlen(w), HAL_MAX_DELAY);
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    if (Auto==1)
    {
        jedzPoLinii();
    }
    if (pokazOdleglosc==1)
    {
        wyswietlOdleglosc();
        pokazOdleglosc=0;
    }
}
/* USER CODE END WHILE */
/* USER CODE BEGIN 3 */
/* USER CODE END 3 */
}
```

Rozpaczynam od wywołań inicjalizujących HAL (HAL_Init()), ustawienia zegara (SystemClock_Config()) oraz konfiguracji peryferiów (MX_GPIO_Init(), MX_TIM3_Init(), MX_TIM2_Init(), MX_USART1_UART_Init()). Następnie uruchamiam PWM na kanałach 1 i 2 timera TIM3 i wywołuję aktualizujPredkosc(), by nadać początkowe wypełnienie, a także startuję licznik TIM2 (HAL_TIM_Base_Start(&htim2)). Aktywuję odbiór jednego bajtu przez UART w trybie przerwań (HAL_UART_Receive_IT(&huart1, (uint8_t*)&znak, 1)) i wysyłam komunikat "Robot Działa\r\n". W pętli while(1) sprawdzam dwie flagi: jeśli Auto == 1, wywołuję jedzPoLinii(), realizując automatyczną jazdę wzdłuż linii; jeśli pokazOdleglosc == 1, wywołuję wyswietlOdleglosc() i resetuję pokazOdleglosc do 0, aby na żądanie przesłać pomiar ultradźwiękowy przez UART.

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    if (huart->Instance == USART1)
    {
        obslugaMenu(znak);
        HAL_UART_Receive_IT(&huart1, (uint8_t*)&znak, 1);
    }
}
```

Funkcja HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) wyłapuje przerwanie po odebraniu znaku na USART1, przekazuje go do obslugaMenu(znak) i ponownie uruchamia odbiór kolejnego bajtu przerwami, co zapewnia ciągłą komunikację.



https://play.google.com/store/apps/details?id=de.kai_morich.serial_bluetooth_terminal&hl=pl

Serial Bluetooth Terminal

Kai Morich
Zakup w aplikacji

4.6★
3,14 tys. opinii

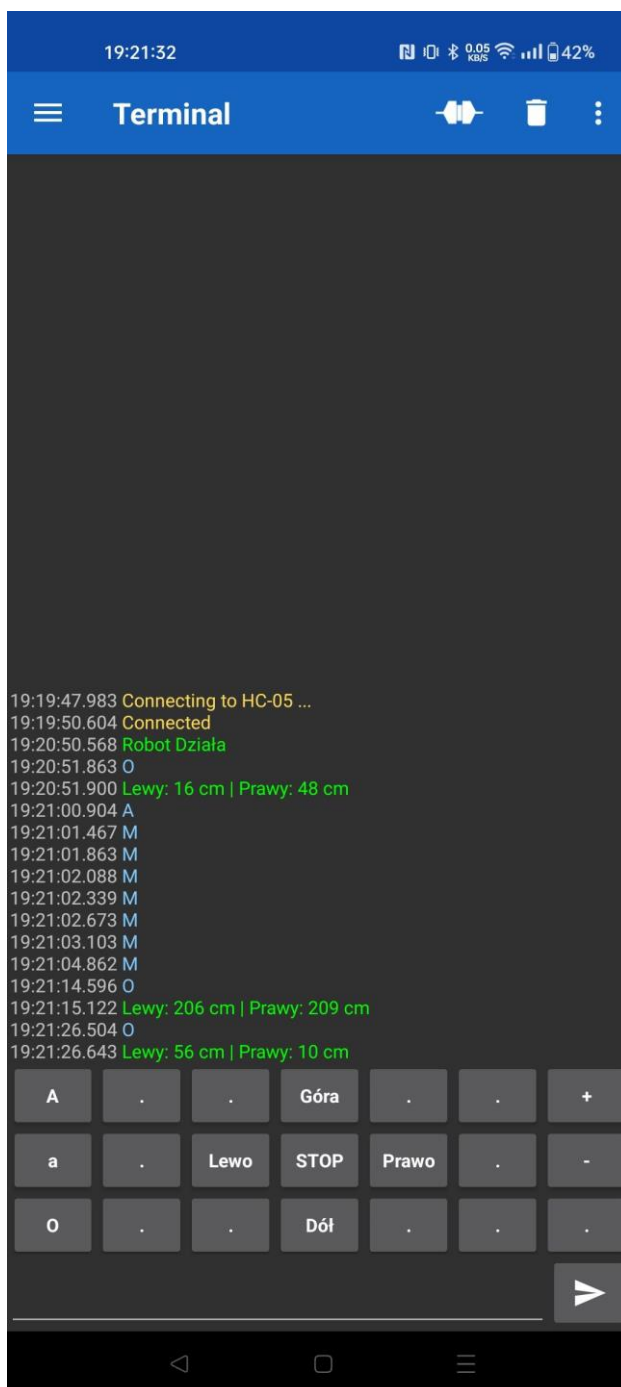
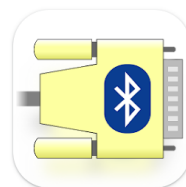
1 min+
Pobrania

PEGI 3

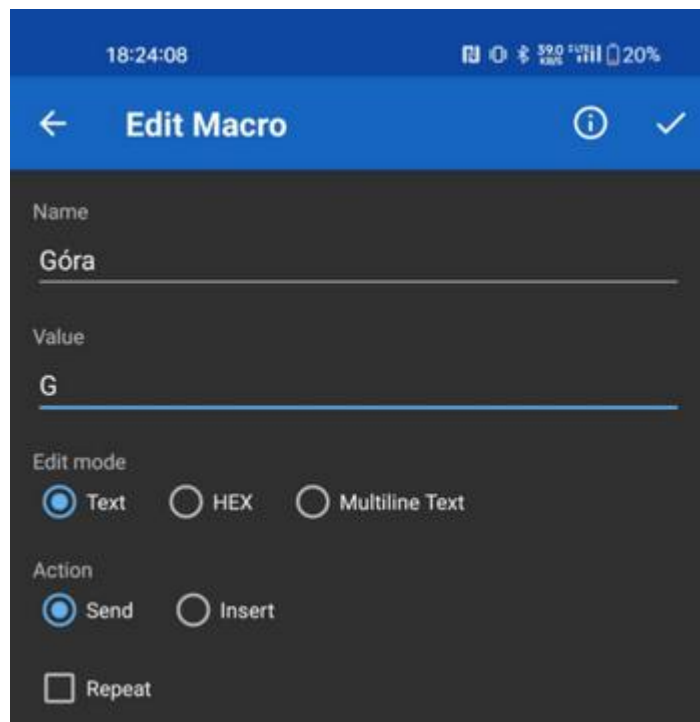
Zainstaluj

Udostępniaj

Dodaj do listy życzeń



Przykładowa konfiguracja przycisku powyżej aplikacji:



18:24:08 39.0 50% 20%

← Edit Macro ⓘ ✓

Name
Góra

Value
G

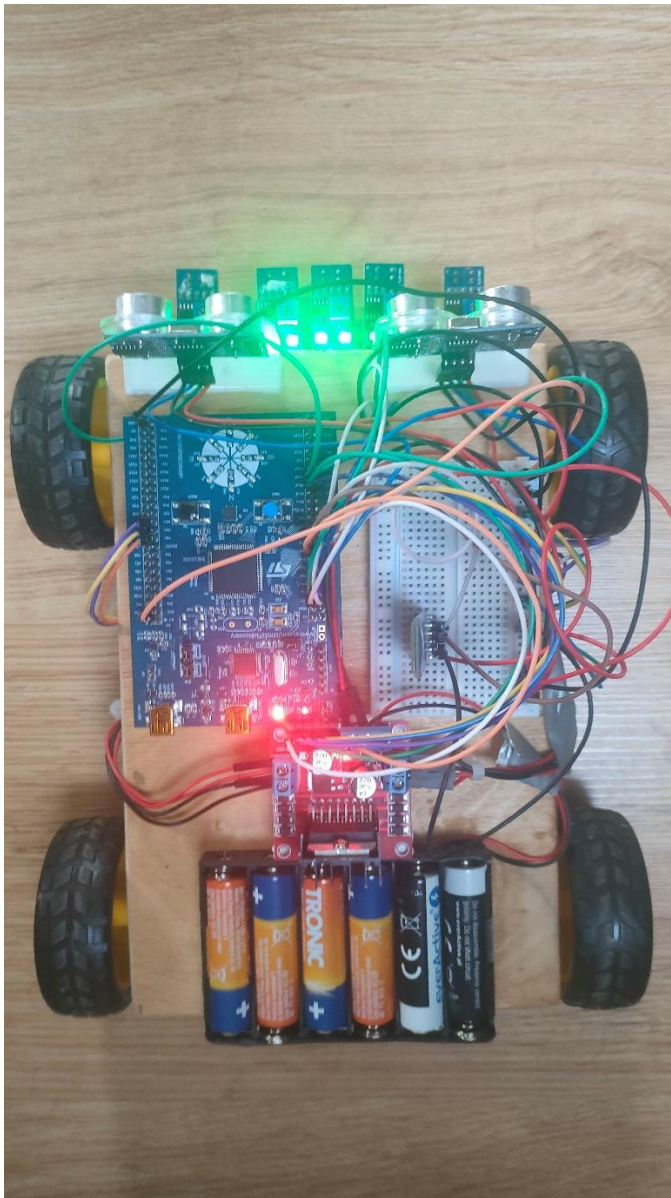
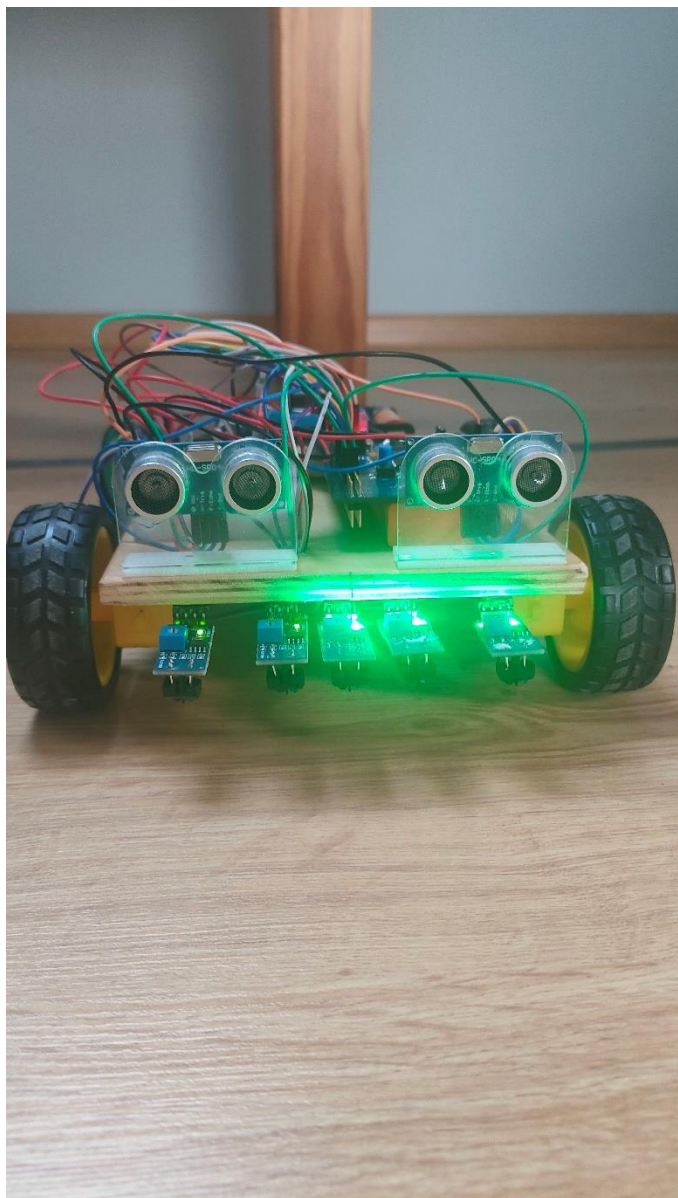
Edit mode
☒ Text ☐ HEX ☐ Multiline Text

Action
☒ Send ☐ Insert

☐ Repeat

W Serial Bluetooth Terminal przyciski makr zdefiniowano tak, by na każdy tap wysyłały jeden znak ASCII odpowiadający komendzie robota (np. „G” dla ruchu do przodu, „T” dla do tyłu, „L” / „P” dla skrętów, „S” dla stop, „A” / „a” dla włączenia/wyłączenia trybu automatycznego, „M” / „N” dla zmiany prędkości oraz „O” do wyświetlenia odległości). Każde makro w edycji ma ustawiony tryb „Text” i działanie „Send”, bez opcji powtarzania.

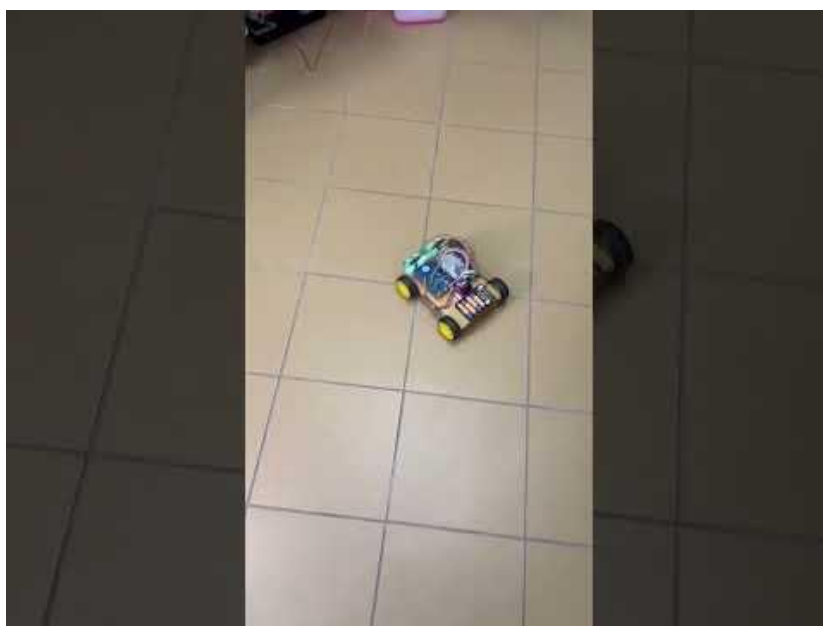
6. Zdjęcia opracowanego robota



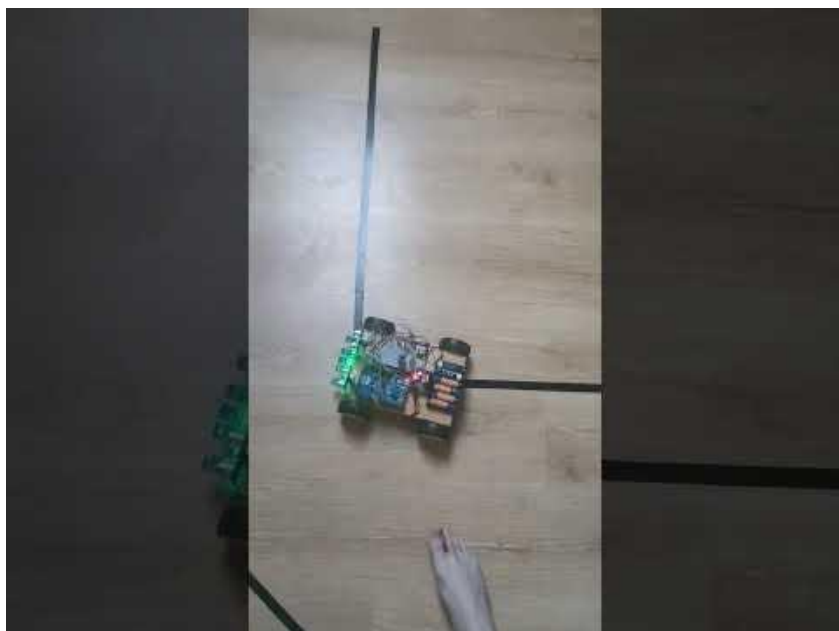
7. Filmy opracowanego robota



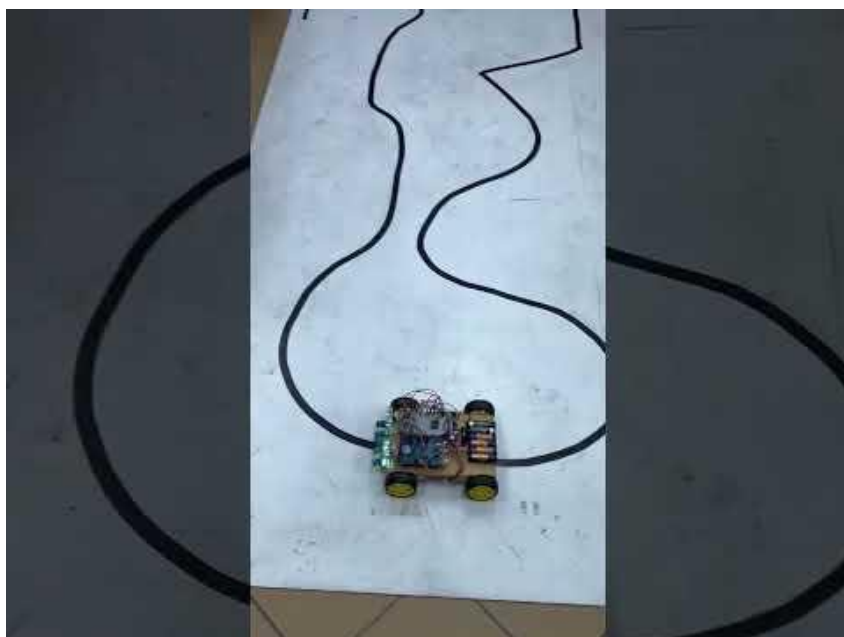
Jazda automatyczna (po linii z przeszkodą)



Jazda manualna



Jazda po linii (skręt 90 stopni)



Jazda automatyczna (Bonus)